NSCT Databases — Practice MCQs

nsctexam.com · 1798 original questions · Answer key at the back. Independent; not affiliated with HEC or VU. Not real NSCT questions.

Introduction to Database Systems MCQs for the ...

1. According to Elmasri and Navathe, what is a database management system (DBMS)? Understanding
   A. A collection of related data with an implicit meaning
   B. The database together with the software that manages it
   C. A file that stores the structure of every table
   D. Software that lets users define, construct, manipulate and share databases
2. In Elmasri-Navathe's terminology, what does the term 'database system' cover? Understanding
   A. Only the DBMS software
   B. The database plus the DBMS software (and, often, the applications)
   C. Only the stored data files
   D. Only the data dictionary
3. A DBMS stores the definitions of its tables, columns, types and constraints in a catalog inside the database. Which characteristic of the database approach is this? Understanding
   A. Self-describing nature of a database system
   B. Support of multiple views of the data
   C. Sharing of data and multiuser transaction processing
   D. Data redundancy
4. Which of these is metadata in a university database? Understanding
   A. A student's CGPA of 3.4
   B. The name 'Ayesha' stored in a student row
   C. The list of courses a student enrolled in this term
   D. The data type and maximum length of the StudentName column
5. A customer's address is stored in both the billing file and the delivery file, and the two copies now differ. Which file-processing problem is this? Understanding
   A. Data isolation
   B. Atomicity problem
   C. Data inconsistency caused by redundancy
   D. Concurrent-access anomaly
6. In Silberschatz's list of file-processing problems, what does 'data isolation' mean? Understanding
   A. Each transaction is hidden from other concurrent transactions
   B. Sensitive data is hidden from unauthorised users
   C. Each user sees only a private view of the data
   D. Data is scattered across files in different formats, so new programs are hard to write
7. Who is responsible for authorising access to the database, coordinating and monitoring its use, and acquiring hardware and software resources? Understanding
   A. The database designer
   B. The database administrator (DBA)
   C. The system analyst
   D. The DBMS implementer

8. Who determines the requirements of end users, especially naive users, and writes specifications for the canned transactions that meet them? Understanding
    A. Database administrators
    B. Tool developers
    C. System analysts
    D. Sophisticated end users

9. Who identifies the data to be stored and chooses appropriate structures for it, usually before the database is implemented? Understanding
    A. Database administrators
    B. Naive end users
    C. Operators and maintenance personnel
    D. Database designers

10. Which of these is a benefit of the database approach rather than a problem of file-processing systems? Understanding
    A. Data isolation
    B. Integrity problems
    C. Atomicity problems
    D. Program-data independence

11. What does program-data independence mean in a DBMS? Understanding
    A. Every program keeps its own private copy of the data it needs
    B. File structure is kept in the catalog, so changing it does not force changes to programs that do not use the changed part
    C. Programs can run without any database connection
    D. Data can be read only through stored procedures

12. A tax consultant keeps a personal database using a ready-made package with menus and graphical screens. Which category of end user is this in Elmasri-Navathe? Understanding
    A. Casual end user
    B. Naive (parametric) end user
    C. Standalone user
    D. Sophisticated end user

13. Who designs and builds the DBMS modules themselves, such as the catalog, query processor and concurrency control? Understanding
    A. DBMS system designers and implementers
    B. Application programmers
    C. Database administrators
    D. Tool developers

14. Which application is the strongest case for plain files rather than a full DBMS? Understanding
    A. A university registration system used by thousands of students at once
    B. A bank's account system that must survive crashes
    C. An online shop whose stock is updated by many staff
    D. Firmware on a small sensor that logs fixed-format readings for one program with very little memory

15. What does the database approach aim for regarding redundancy, according to Elmasri-Navathe? Understanding
    A. Zero redundancy in every case, with no value ever stored twice
    B. Redundancy managed separately by each application
    C. Controlled redundancy: any duplication is deliberate and kept consistent by the DBMS
    D. Redundancy only inside backup copies

16. In Elmasri-Navathe, what is program-operation independence? Understanding

    A. Each operation runs in its own transaction
    B. Programs and operations are stored in different files
    C. Programs call an operation through its interface, so its implementation can change without changing the programs
    D. Operations run without any program calling them

17. The hostel, library and examination offices each need to see a different subset of the same student data. Which characteristic of the database approach supports this? Application

    A. Self-describing nature
    B. Persistent storage of program objects
    C. Support of multiple views of the data
    D. Backup and recovery

18. A clerk at a utility office types a consumer number into a fixed screen and the system prints the bill. The clerk never writes queries. Which kind of user is the clerk? Application

    A. Sophisticated end user
    B. Naive (parametric) end user
    C. Casual end user
    D. Application programmer

19. What does the final SELECT return? Application

```
CREATE TABLE account (id INT PRIMARY KEY, balance INT);
INSERT INTO account VALUES (1, 5000), (2, 3000);
BEGIN TRANSACTION;
UPDATE account SET balance = balance - 2000 WHERE id = 1;
-- crash detected before the credit to account 2: abort
ROLLBACK;
SELECT id, balance FROM account ORDER BY id;
```

    A. (1, 5000) and (2, 3000)
    B. (1, 3000) and (2, 3000)
    C. (1, 3000) and (2, 5000)
    D. (1, 5000) and (2, 5000)

20. After this setup runs, which INSERT is rejected by the DBMS? Application

```
CREATE TABLE student (
  roll INT PRIMARY KEY,
  cgpa REAL CHECK (cgpa BETWEEN 0 AND 4)
);
INSERT INTO student VALUES (1, 3.2);
```

    A. INSERT INTO student VALUES (2, 4.0);
    B. INSERT INTO student VALUES (4, 0);
    C. INSERT INTO student VALUES (5, NULL);
    D. INSERT INTO student VALUES (3, 4.5);

21. SQLite keeps its catalog in the sqlite_master table. What does the last query return? Application

```
CREATE TABLE course (code TEXT PRIMARY KEY, title TEXT);
CREATE TABLE enrol (roll INT, code TEXT);
CREATE VIEW cs_courses AS
  SELECT * FROM course WHERE code LIKE 'CS%';
CREATE INDEX idx_enrol_roll ON enrol (roll);
SELECT COUNT(*) FROM sqlite_master WHERE type = 'table';
```

    A. 3
    B. 2
    C. 4
    D. 5

22. A college keeps each student's 30-byte name in four separate files (admissions, fee, hostel, exam). It has 10,000 students. How many bytes are spent on the redundant copies, beyond one copy per student? Application

    A. 900,000 bytes
    B. 1,200,000 bytes
    C. 300,000 bytes
    D. 600,000 bytes

23. Balance is 10,000. Clerk A withdraws 1,000 and clerk B withdraws 2,000 with no concurrency control. Both read 10,000 before either writes; A writes, then B writes. What is stored, and what should it be? Application

    A. 9,000 is stored; it should be 7,000
    B. 7,000 is stored; it should be 7,000
    C. 8,000 is stored; it should be 8,000
    D. 8,000 is stored; it should be 7,000

24. Someone decides to add an index on the CNIC column and move a busy table to faster disks, without changing any table definitions. Whose job is this in a typical organisation? Application

    A. The database administrator
    B. A naive end user
    C. A tool developer
    D. A system analyst

25. A head of department logs in a few times a term and runs a different ad-hoc query each time through a query interface. Which category of end user is this? Application

    A. Naive (parametric) end user
    B. Casual end user
    C. Standalone user
    D. DBMS implementer

26. Fee records are in fixed-width COBOL files, hostel records in CSV and exam results in spreadsheets. A report of students with unpaid fees who failed an exam takes weeks to write. Which file-processing problem is the main cause? Application

    A. Atomicity problems
    B. Data isolation
    C. Security problems
    D. Integrity problems

27. The rule 'balance must never fall below 500' is coded separately in six programs, and a new seventh program forgets it. What is the database-approach fix? Application
    A. Copy the check into the seventh program as well
    B. Store balances in a separate file for each program
    C. Run a nightly report that lists low balances
    D. Declare the rule once as a constraint in the schema so the DBMS enforces it for every program

28. Payroll clerks must see salaries but not exam results, while exam staff must see results but not salaries. Which DBMS facility handles this? Application
    A. An authorisation subsystem with views and privileges (GRANT/REVOKE)
    B. The recovery subsystem
    C. The query optimiser
    D. Storage of persistent program objects

29. Fee, hostel and exam offices each keep their own copy of every student's address. During a semester 120 students change address, and each change reaches only the one office that received it. How many stale address copies exist at the end? Application
    A. 240
    B. 120
    C. 360
    D. 480

30. A program updates 1,000 fee records inside one transaction. The system crashes after 400 updates. After the DBMS restarts and runs recovery, what state is the data in? Application
    A. None of the 1,000 updates is present; the partial work is undone
    B. The 400 updates stay and the DBMS automatically finishes the other 600
    C. The 400 updates stay and the other 600 are lost
    D. All 1,000 records are reset to the last weekly backup

31. Which pairing of a file-processing problem with the DBMS feature that solves it is WRONG? Analysis
    A. Atomicity problems → transactions with commit and rollback
    B. Integrity problems → constraints declared in the schema
    C. Concurrent-access anomalies → the backup and recovery subsystem
    D. Redundancy and inconsistency → one integrated schema with controlled redundancy

32. Which statement about databases is FALSE? Analysis
    A. Foreign-key values repeat across rows by design
    B. Some redundancy may be kept deliberately to speed up queries
    C. A well-designed database never stores any value twice
    D. The DBMS can enforce consistency between duplicated values

33. Which of these events is a concurrent-access anomaly rather than an atomicity, consistency or integrity problem? Analysis
    A. A transfer debits the source account, then the system crashes before crediting the target
    B. Two booking agents both read seat 12A as free and both sell it
    C. An address is changed in the hostel file but not in the fee file
    D. A program forgets the rule and saves a negative stock level

34. A university adds an Email field to the STUDENT record. Programs on its DBMS that use only Roll and Name keep working, but an old COBOL program reading the same data as a flat file breaks. What explains the difference? Analysis

    A. Program-data independence: the COBOL program has the record layout coded inside it, so its field positions are now wrong
    B. Multiple views: the COBOL program was not given its own view
    C. Data abstraction: COBOL does not support abstract data types
    D. Concurrency control: the COBOL program was reading while the field was added

35. Ahmed writes the Java code for an admission portal. Sana decides the tables and constraints before coding starts. Bilal grants privileges and schedules backups. Hira, an applicant, fills in the portal's forms. Which mapping is correct? Analysis

    A. Ahmed: database designer; Sana: application programmer; Bilal: DBA; Hira: casual end user
    B. Ahmed: application programmer; Sana: DBA; Bilal: database designer; Hira: naive end user
    C. Ahmed: application programmer; Sana: database designer; Bilal: DBA; Hira: naive end user
    D. Ahmed: DBMS implementer; Sana: database designer; Bilal: DBA; Hira: sophisticated end user

36. Each INSERT (a) to (d) is run separately against the table after the setup. Which ones does the DBMS reject? Analysis

```
CREATE TABLE student (
  roll INT PRIMARY KEY,
  name TEXT NOT NULL,
  cgpa REAL CHECK (cgpa BETWEEN 0 AND 4)
);
INSERT INTO student VALUES (1, 'Ali', 3.1);
-- (a) INSERT INTO student VALUES (2, 'Sara', NULL);
-- (b) INSERT INTO student VALUES (1, 'Omar', 3.0);
-- (c) INSERT INTO student VALUES (3, NULL, 2.5);
-- (d) INSERT INTO student VALUES (4, 'Hina', 4.2);
```

    A. (c) and (d) only
    B. (b), (c) and (d)
    C. (a), (c) and (d)
    D. All four

37. Balance is 1,000. Three clerks deposit 100, 200 and 300 with no concurrency control. All three read the balance before anyone writes, then they write in that order. How much money is lost? Analysis

    A. 600
    B. 300
    C. 0
    D. 500

38. Why is the catalog essential for a general-purpose DBMS such as PostgreSQL? Analysis

    A. The catalog holds the actual data rows, so without it there is no data
    B. The catalog exists mainly to make backups faster
    C. Users must query the catalog before they are allowed to write SQL
    D. The DBMS is not written for one application, so it reads the catalog to learn each database's structure before accessing data

39. A shop with one owner kept 200 products in a spreadsheet. It now has three branches and six cashiers updating stock at the same time. Which DBMS capability is the decisive reason to switch now? Analysis
    A. Persistent storage of data
    B. A natural-language query interface
    C. Concurrency control for multiuser transaction processing
    D. Storing data about data

40. Eight programs each code the rule 'age must be at least 17'. The rule changes to 18. How many places must be edited in the file-processing approach and in the DBMS approach with a declared constraint? Analysis
    A. 8 programs versus 1 constraint
    B. 1 program versus 8 constraints
    C. 8 programs versus 8 constraints
    D. 8 programs versus 0 changes

41. In Elmasri-Navathe's definition of a DBMS, what does 'constructing' a database mean? Understanding
    A. Storing the data on a storage medium that the DBMS controls
    B. Specifying the data types, structures and constraints of the data
    C. Querying the data and updating it to reflect changes in the miniworld
    D. Letting many users and programs access the data at the same time

42. In Elmasri-Navathe's terms, what does 'defining' a database involve? Understanding
    A. Loading the first batch of records into the database files
    B. Specifying the data types, structures and constraints of the data to be stored
    C. Writing the reports that end users will run
    D. Granting each user the privileges they need

43. A database represents some aspect of the real world, such as a university's students and courses. What do Elmasri-Navathe call that aspect? Understanding
    A. The schema
    B. The catalog
    C. The miniworld (universe of discourse)
    D. The external view

44. Which of these is one of Elmasri-Navathe's 'workers behind the scene' rather than an 'actor on the scene'? Understanding
    A. The database administrator
    B. A casual end user
    C. A database designer
    D. A tool developer who builds database design and performance-monitoring packages

45. A DBMS offers query languages for casual users, programming interfaces for programmers, fill-in forms for naive users and menus for browsing. Which advantage of the database approach is this? Understanding
    A. Providing multiple user interfaces
    B. Controlling redundancy
    C. Restricting unauthorised access
    D. Persistent storage for program objects

46. Which data model did E. F. Codd introduce in 1970? Understanding
    A. The hierarchical model
    B. The relational model
    C. The network model
    D. The object-oriented model

47. Which early DBMS product was based on the hierarchical data model? Understanding
    A. Oracle
    B. PostgreSQL
    C. IBM's IMS
    D. MongoDB

48. What is the job of operators and maintenance personnel in a database environment? Understanding
    A. Deciding which tables and constraints the database should contain
    B. Writing the canned transactions that tellers use
    C. Granting and revoking user privileges
    D. Running and maintaining the hardware and software environment of the database system

49. In Silberschatz's list of file-processing problems, what is 'difficulty in accessing data'? Understanding
    A. Each new information need requires a new application program, because there is no ad-hoc query facility
    B. Data is lost when the system crashes
    C. Users can read data they are not authorised to see
    D. Two users overwrite each other's updates

50. Which DBMS advantage lets an object from a Java or C++ program be stored so that it survives after the program ends and can be read back later? Understanding
    A. Multiple user views
    B. Persistent storage for program objects
    C. Concurrency control
    D. Restricting unauthorised access

51. Which person is best described as a sophisticated end user? Understanding
    A. A bank teller who uses the same deposit screen all day
    B. A manager who runs a different query once a month
    C. An engineer who writes complex queries and uses analysis tools directly on the DBMS
    D. A shopkeeper who keeps a personal customer list in a ready-made package

52. A DBMS can automatically insert a row into an audit table every time a salary is changed. Which DBMS feature provides this? Understanding
    A. Views
    B. Indexes
    C. The buffer manager
    D. Triggers (active rules)

53. In Elmasri-Navathe, what does data abstraction in the database approach refer to? Understanding
    A. The DBMS hides storage details and gives users a conceptual representation of the data through a data model
    B. Data is compressed so that it takes less disk space
    C. Each user sees only the rows they are allowed to see
    D. Old data is summarised into totals and the details are deleted

54. What are 'canned transactions' in Elmasri-Navathe? Understanding
    A. Transactions that the DBMS has rolled back after a failure
    B. Standard, pre-programmed queries and updates that naive users run again and again
    C. Ad-hoc queries typed by casual users
    D. Transactions that run only when the database is offline

55. Why do Elmasri-Navathe say a DBMS can reduce application development time? Understanding
    A. A DBMS writes application programs automatically from the schema
    B. Applications do not need testing when they use a DBMS
    C. Once the database is running, a new application needs less work because data definition, storage and access are already handled by the DBMS
    D. A DBMS removes the need for a database designer

56. Which of these is NOT one of the DBA's duties in Silberschatz's list? Understanding
    A. Schema definition
    B. Storage structure and access-method definition
    C. Granting authorisation for data access
    D. Writing every end user's ad-hoc queries for them

57. In Silberschatz's list of file-processing problems, what are 'integrity problems'? Understanding
    A. Consistency constraints are buried in program code, so they are hard to add or change
    B. Two users read the same record at the same time
    C. Files are stored in different formats
    D. A crash leaves an operation half done

58. Which two mechanisms do Elmasri-Navathe name for providing efficient query processing in a DBMS? Understanding
    A. Normalisation and triggers
    B. Indexes, and buffering (caching) of disk data in main memory
    C. Views and GRANT statements
    D. Backups and the transaction log

59. In Elmasri-Navathe's definition of a DBMS, what does 'sharing' a database mean? Understanding
    A. Copying the database to every user's computer
    B. Publishing the schema on a website
    C. Allowing multiple users and programs to access the database at the same time
    D. Splitting the database into one file per department

60. According to Elmasri-Navathe, which statement describes the implicit properties of a database? Understanding
    A. It is any collection of files stored on one disk
    B. It must use the relational model
    C. It must be stored on a single computer
    D. It is a logically coherent collection of data with inherent meaning, designed and populated for a specific purpose

61. What does 'standards enforcement' mean as an implication of the database approach? Understanding
    A. The DBA can define and enforce standards such as names, formats and report layouts for all users
    B. The DBMS refuses SQL that does not follow the ISO standard
    C. Every table must have the same number of columns
    D. Each department sets its own data formats

62. According to Elmasri-Navathe, why might a full DBMS be a poor choice for a system with strict real-time deadlines? Understanding
    A. A DBMS cannot store numeric sensor data
    B. The overhead of the DBMS's security, concurrency control, recovery and integrity functions may make the deadlines impossible to meet
    C. A DBMS can serve only one user at a time
    D. Real-time systems never need to store data

63. Which statement correctly distinguishes a casual end user from a sophisticated end user? Understanding
    A. Casual users only use canned transactions, while sophisticated users write SQL
    B. Casual users work behind the scene, while sophisticated users are actors on the scene
    C. Both may write queries; casual users use the database occasionally with different needs each time, while sophisticated users know the DBMS facilities thoroughly
    D. Casual users keep personal databases, while sophisticated users manage the whole database

64. Elmasri-Navathe contrast a relational DBMS's catalog with 'self-describing data' in XML and many NoSQL systems. What is the difference? Understanding
    A. XML data has no description at all, while relational data is described in the catalog
    B. Relational DBMSs store column names inside every row, while XML uses a separate catalog
    C. Only XML databases are self-describing; relational databases are not
    D. In XML and many NoSQL systems the data item names are stored with the data itself, while a relational DBMS keeps the description separately in the catalog

65. A data scientist connects to the sales database and writes complex SQL with window functions and statistical packages to study trends. Which category of end user is this? Application
    A. Sophisticated end user
    B. Naive (parametric) end user
    C. Standalone user
    D. Database designer

66. A file-based banking program debits Rs 2,000 from account A, and the power fails before it credits account B. The Rs 2,000 has vanished. Which Silberschatz file-processing problem is this? Application
    A. Data isolation
    B. Atomicity problem
    C. Security problem
    D. Difficulty in accessing data

67. The registrar asks for a list of students from Quetta with CGPA above 3.5. IT says it will take a week, because a new program must be written to read the student files. Which file-processing problem is this? Application
    A. Concurrent-access anomaly
    B. Atomicity problem
    C. Difficulty in accessing data
    D. Data redundancy

68. A company wants to lose at most one day of data if a disk fails. Which DBMS facility addresses this need? Application
    A. Multiple user views
    B. Triggers
    C. The query optimiser
    D. Backup and recovery

69. What does the final SELECT return? Application

```
CREATE TABLE acct (id INT PRIMARY KEY, bal INT);
INSERT INTO acct VALUES (1, 5000);
BEGIN;
UPDATE acct SET bal = bal - 1500 WHERE id = 1;
ROLLBACK;
SELECT bal FROM acct WHERE id = 1;
```

A. 5000
B. 3500
C. 1500
D. NULL

70. What does the final SELECT return? Application

```
CREATE TABLE emp (id INT, name TEXT, dept TEXT, salary INT);
INSERT INTO emp VALUES (1, 'Ayesha', 'Exams', 90000),
                       (2, 'Bilal', 'Payroll', 85000),
                       (3, 'Hamza', 'Exams', 70000);
CREATE VIEW exam_staff AS
  SELECT id, name FROM emp WHERE dept = 'Exams';
SELECT COUNT(*) FROM exam_staff;
```

A. 3
B. 2
C. 1
D. An error, because exam_staff has no salary column

71. A company stores each employee's phone number in 3 separate department files. During a year, 400 employees change their number. How many record updates are needed with the files, and how many in an integrated database that stores the number once? Application

A. 400 with files and 400 with the database
B. 1,200 with files and 400 with the database
C. 1,200 with files and 1,200 with the database
D. 1,200 with files and 3 with the database

72. Account A holds 8,000 and account B holds 2,000. A file-based program transfers 3,000 from A to B, but crashes after writing A's new balance and before writing B's. What is the total of the two balances after the crash? Application

A. 5,000
B. 7,000
C. 10,000
D. 13,000

73. In SQLite, pragma_table_info lists a table's columns from the catalog. What does the last query return? Application

```
CREATE TABLE student (roll INT, name TEXT, cgpa REAL);
ALTER TABLE student ADD COLUMN email TEXT;
SELECT COUNT(*) FROM pragma_table_info('student');
```

A. 4
B. 3
C. 2
D. 0

74. A DBA adds a column because the university now records each student's blood group, and changes a large table's storage parameters to speed it up. Which of Silberschatz's DBA duties is this? Application

    A. Routine maintenance
    B. Granting authorisation for data access
    C. Schema and physical-organisation modification
    D. Initial schema definition

75. A fee object's calculate_fee() operation changes from a flat fee to a per-credit-hour fee. Programs that call calculate_fee() with the same arguments keep working unchanged. Which property is shown? Application

    A. Physical data independence
    B. Controlled redundancy
    C. Self-describing nature
    D. Program-operation independence

76. A hospital must link patients, doctors, wards and appointments, where a doctor sees many patients and a patient sees many doctors. Which DBMS advantage in Elmasri-Navathe's list matters most here? Application

    A. Representing complex relationships among data
    B. Persistent storage for program objects
    C. Providing multiple user interfaces
    D. Restricting unauthorised access

77. A software company sells a package that helps DBAs monitor database performance and another that helps designers draw ER diagrams. In Elmasri-Navathe's roles, who builds these packages? Application

    A. DBMS system designers and implementers
    B. Tool developers
    C. System analysts
    D. Database administrators

78. How many rows are in salary_log after the script runs? Application

```
CREATE TABLE emp (id INT PRIMARY KEY, salary INT);
CREATE TABLE salary_log (id INT, old_sal INT, new_sal INT);
CREATE TRIGGER log_pay AFTER UPDATE OF salary ON emp
BEGIN
  INSERT INTO salary_log VALUES (OLD.id, OLD.salary, NEW.salary);
END;
INSERT INTO emp VALUES (1, 50000), (2, 60000);
UPDATE emp SET salary = salary + 5000 WHERE id = 1;
UPDATE emp SET salary = salary + 5000;
SELECT COUNT(*) FROM salary_log;
```

    A. 3
    B. 2
    C. 1
    D. 0

79. A clerk updates a student's address once in the university DBMS. Within seconds, the hostel, library and exam screens all show the new address. Which advantage in Elmasri-Navathe's list does this illustrate? Application

    A. Economies of scale
    B. Persistent storage for program objects
    C. Availability of up-to-date information
    D. Reduced application development time

80. Five university departments each used to buy their own small server and keep their own files. After moving to one central DBMS, the university buys one powerful server and removes duplicated work. Which implication of the database approach is this? Application

    A. Data isolation
    B. Program-operation independence
    C. Concurrent-access anomalies
    D. Economies of scale

81. The UPDATE has no WHERE clause and runs on its own (autocommit). What does the final SELECT return? Application

```
CREATE TABLE stock (item TEXT PRIMARY KEY, qty INT CHECK (qty >= 0));
INSERT INTO stock VALUES ('pen', 10), ('ink', 3);
UPDATE stock SET qty = qty - 5;
SELECT SUM(qty) FROM stock;
```

    A. 13
    B. 8
    C. 3
    D. -2

82. A flight has 3 seats left. With no concurrency control, 5 agents each read 'seats left = 3' at the same moment, each sells one ticket and writes back their own value minus 1. How many tickets were sold, and what does the file now show? Application

    A. 3 sold; it shows 0
    B. 5 sold; it shows 2
    C. 5 sold; it shows -2
    D. 1 sold; it shows 2

83. Four department files each store the full 200-byte profile of all 2,500 students. An integrated database stores each profile once, and each department keeps only an 8-byte roll-number reference per student. How many bytes does the integrated design save? Application

    A. 1,500,000
    B. 1,420,000
    C. 2,000,000
    D. 580,000

84. The second transaction's debit fails the CHECK constraint, and the application responds by issuing ROLLBACK. What are the final balances of accounts 1 and 2? Application

```
CREATE TABLE acct (id INT PRIMARY KEY, bal INT CHECK (bal >= 0));
INSERT INTO acct VALUES (1, 1000), (2, 500);
BEGIN;
UPDATE acct SET bal = bal + 700 WHERE id = 2;
UPDATE acct SET bal = bal - 700 WHERE id = 1;
COMMIT;
BEGIN;
UPDATE acct SET bal = bal + 700 WHERE id = 2;
UPDATE acct SET bal = bal - 700 WHERE id = 1;  -- fails
ROLLBACK;
SELECT bal FROM acct ORDER BY id;
```

A. 300 and 1,200
B. 300 and 1,900
C. 1,000 and 500
D. -400 and 1,900

85. SQLite enforces foreign keys only after the PRAGMA. Each statement (a) to (d) is run separately against the setup. Which ones does the DBMS reject? Application

```
PRAGMA foreign_keys = ON;
CREATE TABLE dept (id INT PRIMARY KEY);
CREATE TABLE emp (id INT PRIMARY KEY,
                  dept_id INT REFERENCES dept(id));
INSERT INTO dept VALUES (10), (20);
INSERT INTO emp VALUES (1, 10), (2, 20), (3, NULL);
-- (a) INSERT INTO emp VALUES (4, 30);
-- (b) DELETE FROM dept WHERE id = 20;
-- (c) INSERT INTO emp VALUES (5, NULL);
-- (d) UPDATE emp SET dept_id = 20 WHERE id = 3;
```

A. (a) and (b)
B. (a) only
C. (a), (b) and (d)
D. (b) and (c)

86. A student says: 'Now that we have moved from files to a DBMS, we no longer need to take backups.' What is wrong with this claim? Analysis

A. A DBMS provides backup and recovery facilities, but backups must still be taken because disks and sites can still fail
B. Nothing; a DBMS never loses data
C. A DBMS cannot be backed up, so the claim is irrelevant
D. Backups are only needed for file systems that use spreadsheets

87. When a student withdraws, a file-based program updates the admissions file and is then meant to update the exam file. The system crashes between the two writes, and the files now disagree. Which DBMS feature would have prevented this? Analysis

A. Concurrency control
B. Transactions, so both writes commit together or neither does
C. Views for each office
D. Authorisation with GRANT and REVOKE

88. A student claims that Silberschatz's 'data isolation' problem is the same as the 'I' in ACID. Which evaluation is correct? Analysis
    A. Correct: both mean transactions must not interfere
    B. Wrong: data isolation means a user cannot see other users' data
    C. Wrong: data isolation means data scattered across files in different formats, while ACID isolation means concurrent transactions do not see each other's partial work
    D. Correct: both mean data is kept on separate disks

89. Which of these is NOT an advantage of using a DBMS according to Elmasri-Navathe? Analysis
    A. Backup and recovery
    B. Enforcing integrity constraints
    C. Restricting unauthorised access
    D. A lower total cost than file processing for every application

90. Design X stores the department name in every employee record. Design Y stores a dept_id in each employee record, with a FOREIGN KEY to a DEPT table that holds the name once. Which statement best fits Elmasri-Navathe's idea of controlled redundancy? Analysis
    A. X is controlled because the names are typed by trained clerks
    B. Y has no redundancy at all, because dept_id is a number
    C. Y repeats dept_id deliberately, and the DBMS can keep it consistent with DEPT; X's repeated names can drift apart
    D. Both are uncontrolled, because any repeated value is a design error

91. In a file system, three offices each keep their own file with the student fields they need. In a DBMS, the three offices use three views of one STUDENT table. What is the key difference? Analysis
    A. The views are faster to update because each one is a separate copy
    B. There is no difference; a view is just a file with a different name
    C. The files are more secure because each office cannot see the others' files
    D. The views are derived from one stored copy, so they add no redundancy and always agree; the three files are separate copies that can drift apart

92. Which of these events is an integrity problem rather than a security, concurrency or atomicity problem? Analysis
    A. A clerk reads the salaries of all staff although his job does not need them
    B. Two clerks update the same balance and one update is lost
    C. A program stores a CGPA of 5.2 because no rule checks the 0 to 4 range
    D. A crash leaves a transfer debited but not credited

93. What does the final SELECT return, and why? Analysis

```
CREATE TABLE student (roll INT, name TEXT, city TEXT);
INSERT INTO student VALUES (1, 'Ali', 'Lahore'), (2, 'Sana', 'Quetta');
CREATE VIEW hostel_view AS SELECT roll, city FROM student;
UPDATE student SET city = 'Multan' WHERE roll = 1;
SELECT city FROM hostel_view WHERE roll = 1;
```

    A. Lahore, because the view stored a copy of the data when it was created
    B. An error, because hostel_view has no name column
    C. Both Lahore and Multan, because the old and new values are kept
    D. Multan, because a view is evaluated from the base table each time it is queried

94. Balance is 4,000. T1 deposits 1,000 and T2 withdraws 500, with no concurrency control. The log shows: T1 reads 4,000; T2 reads 4,000; T2 writes 3,500; T1 writes 5,000. Which diagnosis is correct? Analysis

    A. T1's deposit is a lost update; the stored 5,000 should be 3,500
    B. Nothing is wrong; 5,000 is correct because T1 wrote last
    C. T2's withdrawal is a lost update; the stored 5,000 should be 4,500
    D. It is an atomicity problem; the stored value should be 4,000

95. A company runs one central DBMS, but each department's application copies the CUSTOMER table every night into its own private file and works from that copy all day. Which file-processing problem has returned? Analysis

    A. Atomicity problems, because the nightly copy is a transaction
    B. Difficulty in accessing data, because SQL can no longer be used
    C. Data isolation in the ACID sense, because departments cannot see each other
    D. Redundancy and inconsistency, because the copies drift apart from the central data during the day

96. Each statement runs in autocommit mode, with no BEGIN. The second UPDATE fails the CHECK. What does the final SELECT return? Analysis

```
CREATE TABLE acct (id INT PRIMARY KEY, bal INT CHECK (bal >= 0));
INSERT INTO acct VALUES (1, 300), (2, 100);
UPDATE acct SET bal = bal + 500 WHERE id = 2;
UPDATE acct SET bal = bal - 500 WHERE id = 1;
SELECT bal FROM acct ORDER BY id;
```

    A. 300 and 100: the DBMS undoes both updates
    B. -200 and 600: the CHECK is ignored once the first update succeeds
    C. 300 and 600: the credit was already committed, so 500 has appeared from nowhere
    D. An error, so no rows are returned

97. Which pairing of a user group with the DBMS interface they would typically use is WRONG? Analysis

    A. Naive parametric users: forms and screens with a few function keys
    B. Casual users: an interactive query language or menu-based interface
    C. Application programmers: embedded SQL or a database API such as JDBC
    D. Naive parametric users: privileged commands to create accounts and set system parameters

98. Which scenario is NOT one of Elmasri-Navathe's situations where regular files may be preferable to a DBMS? Analysis

    A. A simple, well-defined application that is not expected to change
    B. An embedded system with very limited storage
    C. Many users update the same data at the same time
    D. An application with stringent real-time requirements

99. A student's address is held in 3 files. In one term 60 students move. The fee office updates all 60, the hostel office updates 40 of them, and the exam office updates 15, all of whom are among the hostel's 40. How many students now have at least one stale copy? Analysis

    A. 20
    B. 45
    C. 65
    D. 105

100. Classify these two changes. (1) In a DBMS, the stored format of the dob field changes from text to an integer day count, and programs that query dob as a date are unaffected. (2) A compute_gpa() operation gets a faster algorithm, and programs that call it with the same arguments are unaffected. Analysis
    A. (1) program-operation independence; (2) program-data independence
    B. Both show program-data independence
    C. Both show controlled redundancy
    D. (1) program-data independence; (2) program-operation independence

Database System Architecture MCQs for the NSCT

101. Which level of the three-schema architecture describes physical storage structures and access paths such as indexes? Understanding
    A. Internal level
    B. Conceptual level
    C. External level
    D. View level

102. What does the conceptual schema describe? Understanding
    A. How records are laid out in disk blocks
    B. The part of the database that one user group sees
    C. The structure of the whole database for all users: entities, data types, relationships and constraints, without storage details
    D. The SQL text of every application program

103. How many external schemas can a database have under the three-schema architecture? Understanding
    A. Exactly one
    B. Many, typically one for each user group
    C. Exactly three
    D. One for each physical file

104. Which body proposed the three-level (external, conceptual, internal) architecture for database systems in the 1970s? Understanding
    A. The IEEE 802 committee
    B. The ISO OSI committee
    C. The ANSI/SPARC study group
    D. The W3C

105. Silberschatz describes three levels of data abstraction. Which of his levels corresponds to Elmasri-Navathe's conceptual level? Understanding
    A. Physical level
    B. View level
    C. Logical level
    D. Storage level

106. What is physical data independence? Understanding
    A. The ability to change the conceptual schema without changing external schemas
    B. The ability to move data to another DBMS without any change
    C. The ability to change the internal schema without changing the conceptual schema
    D. The ability of users to choose their own storage devices

107. What is logical data independence? Understanding
    A. The ability to change the conceptual schema without changing external schemas or application programs
    B. The ability to change the internal schema without changing the conceptual schema
    C. The ability to change the data without changing the schema
    D. The ability to run the same query on different DBMS products

108. In the three-schema architecture, what is the process of transforming requests and results between levels called? Understanding
    A. Normalisation
    B. Serialisation
    C. Indexing
    D. Mapping

109. What do Elmasri-Navathe call the data in the database at a particular moment in time? Understanding
    A. The database schema
    B. The catalog
    C. The database state (also called an instance or snapshot)
    D. The internal schema

110. Where does the DBMS keep the three schemas and the mappings between them? Understanding
    A. Inside each application program
    B. In the transaction log
    C. In the DBMS catalog (data dictionary)
    D. In the buffer pool

111. In Elmasri-Navathe's terminology, which language specifies the internal schema when it is kept separate from the conceptual schema? Understanding
    A. Storage definition language (SDL)
    B. View definition language (VDL)
    C. Data manipulation language (DML)
    D. Data control language (DCL)

112. How is an SQL SELECT statement classified as a data manipulation language? Understanding
    A. Low-level (procedural) and record-at-a-time
    B. High-level (nonprocedural) and set-at-a-time
    C. High-level and record-at-a-time
    D. Low-level and set-at-a-time

113. In Silberschatz's DBMS structure, which component fetches data from disk into main memory and decides what to keep cached there? Understanding
    A. File manager
    B. Buffer manager
    C. Transaction manager
    D. DML compiler

114. Which component picks the lowest-cost evaluation plan from several equivalent plans for a query? Understanding
    A. The buffer manager
    B. The query optimiser
    C. The DDL interpreter
    D. The recovery manager

115. In a three-tier database application, where is the business logic usually executed? Understanding

A. On the database server only
B. In the client's web browser only
C. In the DBMS catalog
D. On the application server (middle tier)

116. According to Elmasri-Navathe, which kind of data independence is harder to achieve, and why? Understanding

A. Physical, because storage devices change often
B. Logical, because the internal schema cannot be changed
C. Physical, because indexes must be rebuilt by every application
D. Logical, because application programs depend heavily on the logical structure of the data they access

117. The DBA creates a B+ tree index on Student.CNIC. No table definitions or programs change, and queries return the same results faster. Which property does this show? Application

A. Logical data independence
B. Program-operation independence
C. Physical data independence
D. Referential integrity

118. An Email attribute is added to the STUDENT relation. Existing user views and the programs that use them keep working without change. Which property does this show? Application

A. Physical data independence
B. Atomicity
C. Logical data independence
D. Concurrency control

119. A table is moved from hard disks to SSDs, and its file organisation changes from a heap file to a hashed file. Programs are not changed. Which level was changed? Application

A. The conceptual level, so this is logical data independence
B. The external level, so user views were rewritten
C. Both the conceptual and internal levels
D. The internal level, so this is physical data independence

120. STUDENT is split into STUDENT and ADDRESS, and the views used by programs are redefined as a join so that programs see the same columns as before. Which mapping had to change? Application

A. The conceptual/internal mapping only
B. The external/conceptual mapping
C. No mapping changed
D. The mapping between the DBMS and the operating system

121. Query Q is run before and after an index is created. How do the two runs compare? Application

```
CREATE TABLE marks (roll INT, subject TEXT, score INT);
INSERT INTO marks VALUES (1, 'DB', 72), (2, 'DB', 85), (3, 'OS', 64), (4, 'DB', 91);
SELECT roll FROM marks WHERE score > 80 ORDER BY roll;   -- query Q, run 1
CREATE INDEX idx_score ON marks (score);
SELECT roll FROM marks WHERE score > 80 ORDER BY roll;   -- query Q, run 2
```

A. Run 1 returns 2 and 4; run 2 returns only 4
B. Run 2 fails because Q was written before the index existed
C. Run 2 returns the rows sorted by score instead of roll
D. Both return rolls 2 and 4; only the access path may differ

122. What does the final SELECT return? Application

```
CREATE TABLE student (roll INT, name TEXT, cgpa REAL);
INSERT INTO student VALUES (1, 'Ali', 3.1), (2, 'Sara', 3.6);
CREATE VIEW advisor_view AS SELECT roll, name FROM student;
ALTER TABLE student ADD COLUMN email TEXT;
SELECT * FROM advisor_view;
```

A. Two columns (roll, name) for both students
B. Three columns (roll, name, cgpa)
C. Four columns including email
D. An error, because the base table changed

123. A database has five external schemas, one for each department. Under the three-schema architecture, how many external/conceptual and conceptual/internal mappings are there? Application

A. 1 external/conceptual and 5 conceptual/internal
B. 5 of each
C. 1 of each
D. 5 external/conceptual and 1 conceptual/internal

124. What does the statement CREATE VIEW fee_view AS SELECT roll, amount FROM fee; define? Application

A. The internal schema
B. A new database state
C. The conceptual/internal mapping
D. An external schema (a user view)

125. A desktop application on each clerk's PC connects through JDBC straight to an Oracle server and runs its SQL there. Which architecture is this? Application

A. Two-tier client/server
B. Three-tier
C. Centralised mainframe with dumb terminals
D. Distributed database

126. A student portal works as browser → Django application server → PostgreSQL. Which tier sends SQL to the database? Application

A. The browser
B. The database server sends SQL to itself
C. The application server
D. Each tier sends its own SQL

127. Which of the statements in this script changes the schema rather than only the database state? Application

```
CREATE TABLE course (code TEXT, title TEXT);
INSERT INTO course VALUES ('CS101', 'Programming');
UPDATE course SET title = 'Programming I' WHERE code = 'CS101';
ALTER TABLE course ADD COLUMN credits INT;
DELETE FROM course WHERE code = 'CS101';
```

A. INSERT INTO course VALUES ('CS101', 'Programming');
B. ALTER TABLE course ADD COLUMN credits INT;
C. UPDATE course SET title = 'Programming I' WHERE code = 'CS101';
D. DELETE FROM course WHERE code = 'CS101';

128. In what order does a DBMS process an SQL query? Application

A. Optimisation, then parsing, then evaluation
B. Parsing and translation, then optimisation, then evaluation
C. Evaluation, then optimisation, then parsing
D. Parsing, then evaluation, then optimisation

129. Which change can data independence NOT hide from the programs that use the database? Application

A. Adding a hash index on Phone
B. Deleting the Phone attribute that a report program prints
C. Adding a new attribute Email that no program uses
D. Splitting a table and redefining the views to join the parts

130. A three-tier system has 200 browser clients and 4 application servers, each with a database connection pool of 25. What is the largest number of simultaneous connections to the database server? Application

A. 200
B. 25
C. 800
D. 100

131. After the ALTER TABLE, program A breaks but program B keeps working. Why? Analysis

```
CREATE TABLE student (roll INT, name TEXT, cgpa REAL);
INSERT INTO student VALUES (1, 'Ali', 3.1);
ALTER TABLE student ADD COLUMN email TEXT;
SELECT * FROM student;                  -- program A: unpacks exactly 3 values
SELECT roll, name, cgpa FROM student;   -- program B
```

A. SELECT * now returns 4 columns, while B names its 3 columns explicitly
B. ALTER TABLE deleted the existing row
C. B uses an index created by ALTER TABLE
D. SELECT * is not allowed once a table has been altered

132. Both SELECT statements on emp_view return the same rows. What does this demonstrate? Analysis

```
-- Before: one table, and programs query the view
CREATE TABLE emp_old (eid INT, ename TEXT, dno INT, dname TEXT);
INSERT INTO emp_old VALUES (1, 'Ali', 10, 'IT'), (2, 'Sara', 20, 'HR');
CREATE VIEW emp_view AS SELECT eid, ename, dname FROM emp_old;
SELECT * FROM emp_view ORDER BY eid;
-- After: the DBA splits the table and redefines the view
CREATE TABLE emp (eid INT, ename TEXT, dno INT);
CREATE TABLE dept (dno INT, dname TEXT);
INSERT INTO emp SELECT eid, ename, dno FROM emp_old;
INSERT INTO dept SELECT DISTINCT dno, dname FROM emp_old;
DROP VIEW emp_view;
DROP TABLE emp_old;
CREATE VIEW emp_view AS
  SELECT e.eid, e.ename, d.dname FROM emp e JOIN dept d ON e.dno = d.dno;
SELECT * FROM emp_view ORDER BY eid;
```

A. Logical data independence: the conceptual schema changed and only the external/conceptual mapping (the view definition) was updated
B. Physical data independence: only the storage structures changed
C. Neither: programs had to be rewritten to use the new tables
D. Atomicity: the restructuring ran as one transaction

133. Which statement about the three-schema architecture is FALSE? Analysis

A. Creating an index can change which rows a SELECT returns
B. Only the internal level describes data that physically exists on disk
C. A user view can hide columns that the conceptual schema contains
D. Each extra level of mapping adds processing overhead

134. Why do Elmasri-Navathe say few commercial DBMSs implement the full three-schema architecture? Analysis

A. The two levels of mapping add overhead at compile or run time, so many systems merge some levels
B. The ANSI/SPARC standard forbids implementing the internal level
C. SQL cannot express views, so the external level is impossible
D. Relational systems do not store any metadata

135. A user queries a view. In which order does the DBMS apply the mappings to reach the stored data? Analysis

A. Conceptual/internal, then external/conceptual
B. External/conceptual, then conceptual/internal
C. Only external/internal, skipping the conceptual level
D. No mapping; views store their own copy of the data

136. Classify these changes: (1) add a hash index; (2) add an attribute to an entity; (3) compress a table's data blocks; (4) merge two tables and redefine the views. Which classification is correct? Analysis

A. Physical: 1 and 2. Logical: 3 and 4.
B. Physical: 1 and 3. Logical: 2 and 4.
C. Physical: 3 only. Logical: 1, 2 and 4.
D. Physical: 1, 3 and 4. Logical: 2 only.

137. A bank wants 5,000 mobile clients, must not expose database credentials on phones and wants business rules in one place. Which design meets all three needs best? Analysis

A. Two-tier: each app connects directly to the database with its own login
B. Two-tier with the business rules copied into every app
C. Centralised: phones act as dumb terminals logged into the database host
D. Three-tier: apps call an application server, which alone connects to the database

138. Straight after the CREATE TABLE statements for a new database have run, and before any INSERT, what is true? Analysis

A. Neither a schema nor a state exists yet
B. The database state is the same as the schema
C. The internal schema does not exist until data is inserted
D. The schema exists and the database is in the empty state

139. An UPDATE would set a cgpa to 4.6 and is rejected by a CHECK constraint. In Silberschatz's DBMS structure, which component tests this rule? Analysis

A. The authorisation and integrity manager
B. The buffer manager
C. The file manager
D. The DDL interpreter

140. The database moves to a new server address. A two-tier version of the system has 300 desktop clients, each with the address in its own settings; a three-tier version has 2 application servers. How many configurations must be edited in each version? Analysis

A. 2 in the two-tier version and 300 in the three-tier version
B. 300 in both versions
C. 300 in the two-tier version and 2 in the three-tier version
D. 1 in both versions

141. Elmasri-Navathe give another name for the database schema. What is it? Understanding

A. The intension
B. The extension
C. The snapshot
D. The instance

142. What does a schema diagram show? Understanding

A. The current rows stored in every table
B. The structure of each record type, such as relation and attribute names, but not the actual data
C. The physical disk blocks where each table is stored
D. The list of users and their passwords

143. In Elmasri-Navathe's terms, what is each object in a schema, such as STUDENT or COURSE, called? Understanding

A. A database state
B. An external view
C. A schema construct
D. A mapping

144. In Elmasri-Navathe's terminology, which language specifies user views and their mappings to the conceptual schema? Understanding

A. Storage definition language (SDL)
B. Data manipulation language (DML)
C. Host language
D. View definition language (VDL)

145. When SQL statements are embedded in a Java or C program, what is Java or C called? Understanding

A. The host language
B. The data sublanguage
C. The storage definition language
D. The view definition language

146. How does a low-level (procedural) DML retrieve data? Understanding

A. As a whole set of rows from one declarative statement
B. One record at a time, usually inside loops in a programming language
C. Only through views
D. By reading the catalog instead of the data

147. In Silberschatz's DBMS structure, which storage-manager component allocates disk space and manages the data structures that represent information on disk? Understanding

A. Buffer manager
B. Transaction manager
C. File manager
D. Query evaluation engine

148. In Silberschatz's DBMS structure, which component keeps the database consistent despite system failures and makes concurrent transactions run without conflicting? Understanding
    A. File manager
    B. DDL interpreter
    C. Authorisation and integrity manager
    D. Transaction manager

149. Which DBMS utility helps the DBA bring existing data, such as text files, into the database? Understanding
    A. The loading utility
    B. The performance-monitoring utility
    C. The query optimiser
    D. The precompiler

150. What do Elmasri-Navathe call a DBMS whose database and software are spread over many sites connected by a network? Understanding
    A. A centralised DBMS
    B. A distributed DBMS
    C. A single-user DBMS
    D. A special-purpose DBMS

151. Which standard API lets a client program call any DBMS that provides the right driver? Understanding
    A. HTTP
    B. SMTP
    C. ODBC (Open Database Connectivity)
    D. FTP

152. Which category of data model, such as the ER model, uses concepts close to the way users perceive data? Understanding
    A. Representational (implementation) data models
    B. Low-level (physical) data models
    C. Storage definition models
    D. High-level (conceptual) data models

153. Which statement describes representational (implementation) data models such as the relational model? Understanding
    A. They are understood by end users but are not too far from how data is organised in storage, so they can be implemented directly
    B. They describe how records are stored as files and blocks on disk
    C. They exist only on paper and are never implemented
    D. They are used only for designing user views

154. What do Elmasri-Navathe call changes made to a schema after the database is in operation, such as adding an attribute? Understanding
    A. State transition
    B. Schema evolution
    C. Mapping
    D. Normalisation

155. In Elmasri-Navathe's DBMS component modules, which module controls access to DBMS information on disk, whether it is part of the database or the catalog? Understanding
    A. The query compiler
    B. The precompiler
    C. The stored data manager
    D. The DDL compiler

156. Which DBMS utility changes a set of database files to a different file organisation to improve performance? Understanding
    A. The loading utility
    B. The backup utility
    C. The DDL compiler
    D. The file reorganisation utility

157. What does the runtime database processor do in Elmasri-Navathe's DBMS structure? Understanding
    A. Executes privileged commands, executable query plans and canned transactions with runtime parameters
    B. Translates ER diagrams into tables
    C. Removes SQL from host-language programs before compilation
    D. Stores the schema definitions in the catalog

158. In client/server terms, what is a thin client? Understanding
    A. A client that holds all business rules and SQL in its own program
    B. A client that runs only the user interface, with most processing done on servers
    C. A server with very little memory
    D. A client that stores a full copy of the database

159. In Silberschatz's description of disk storage, which item holds metadata about the structure of the database, in particular its schema? Understanding
    A. The data files
    B. The indices
    C. The data dictionary
    D. The buffer pool

160. What characterises a federated database system? Understanding
    A. One database copied identically to every site
    B. A single-user database on one desktop
    C. A database that uses only the hierarchical model
    D. Autonomous, often heterogeneous, pre-existing databases that are loosely integrated

161. What does the DDL compiler do in Elmasri-Navathe's DBMS structure? Understanding
    A. Processes schema definitions and stores their descriptions (metadata) in the catalog
    B. Executes SELECT statements and returns rows
    C. Chooses the cheapest plan for each query
    D. Writes data pages from memory to disk

162. According to Elmasri-Navathe, how is the internal schema usually specified in today's relational DBMSs? Understanding
    A. Through a separate storage definition language that every relational DBMS provides
    B. There is no separate SDL; the DBA controls storage through functions, parameters and specifications such as index choices
    C. Through CREATE VIEW statements
    D. It is never specified; relational DBMSs have no internal level

163. Which criterion do Elmasri-Navathe call the main one for classifying DBMSs? Understanding
    A. The number of users it supports
    B. The number of sites it runs on
    C. The data model the DBMS is based on
    D. Its cost

164. What is the difference between a homogeneous and a heterogeneous distributed DBMS? Understanding

A. Homogeneous stores the same rows at every site; heterogeneous stores different rows
B. Homogeneous has one user; heterogeneous has many
C. Homogeneous uses only SQL; heterogeneous uses no query language
D. Homogeneous uses the same DBMS software at every site; heterogeneous uses different DBMS software at different sites

165. An admissions clerk enters each new applicant by filling in boxes on a screen that looks like the paper admission form. Which kind of DBMS interface is this? Application

A. A forms-based interface
B. A natural language interface
C. A privileged DBA interface
D. An embedded SQL interface

166. A manager types 'show all students from Lahore with CGPA above 3.5' in plain English, and the system works out the query. Which kind of interface is this? Application

A. A forms-based interface
B. A natural language interface
C. An interface for parametric users
D. A menu-based interface

167. A bank teller uses a screen where one function key starts a deposit and another starts a withdrawal, keeping keystrokes to a minimum. Which interface is this, in Elmasri-Navathe's list? Application

A. A natural language interface
B. An interface for the DBA
C. An interface for parametric users
D. A graphical query interface

168. The view is created before any course is inserted. What does the final SELECT return? Application

```
CREATE TABLE course (code TEXT, title TEXT, credits INT);
CREATE VIEW cs_view AS
  SELECT code, title FROM course WHERE code LIKE 'CS%';
INSERT INTO course VALUES ('CS101', 'Programming', 4),
                          ('MA101', 'Calculus', 3),
                          ('CS201', 'Data Structures', 4);
SELECT COUNT(*) FROM cs_view;
```

A. 0
B. 3
C. An error, because the view was created on an empty table
D. 2

169. A student portal runs as: browser → application server → database server. In the three-tier architecture, which tier is the browser? Application

A. The presentation (client) tier
B. The business logic tier
C. The database server tier
D. The storage manager

170. SQLite's EXPLAIN QUERY PLAN shows how a query will be run. How does the plan for the same SELECT change after the index is created? Application

```
CREATE TABLE student (roll INT, name TEXT, city TEXT);
EXPLAIN QUERY PLAN SELECT roll FROM student WHERE city = 'Lahore';
CREATE INDEX idx_city ON student(city);
EXPLAIN QUERY PLAN SELECT roll FROM student WHERE city = 'Lahore';
```

A. It changes from SEARCH using idx_city to SCAN
B. It changes from SCAN of the whole table to SEARCH using idx_city
C. It does not change, because indexes are used only for ORDER BY
D. The query is rejected until the index is rebuilt

171. Which statements act only on the database state rather than on the schema? Application

```
CREATE TABLE fee (roll INT, amount INT);        -- 1
INSERT INTO fee VALUES (1, 5000);               -- 2
ALTER TABLE fee ADD COLUMN paid_on TEXT;        -- 3
UPDATE fee SET amount = 5500 WHERE roll = 1;    -- 4
CREATE INDEX idx_fee_roll ON fee(roll);         -- 5
DELETE FROM fee WHERE roll = 1;                 -- 6
```

A. 2, 4 and 6
B. 2 and 4 only
C. 2, 4, 5 and 6
D. 1, 3 and 5

172. A database has 4 user views. The DBA adds 2 more views and also changes one table from a heap file to a hashed file. Which mappings are added or changed? Application

A. 6 external/conceptual mappings are changed, and the conceptual/internal mapping is unchanged
B. 2 conceptual/internal mappings are added
C. 2 external/conceptual mappings are added, and the conceptual/internal mapping is changed
D. Only the conceptual schema changes; no mapping is affected

173. The phone column is renamed to mobile, and the contact view is redefined as shown. A program runs SELECT phone FROM contact. What does it get? Application

```
CREATE TABLE student (roll INT, phone TEXT);
INSERT INTO student VALUES (1, '0300-1111');
CREATE VIEW contact AS SELECT roll, phone FROM student;
DROP VIEW contact;
ALTER TABLE student RENAME COLUMN phone TO mobile;
CREATE VIEW contact AS SELECT roll, mobile AS phone FROM student;
SELECT phone FROM contact;
```

A. An error, because phone no longer exists
B. NULL, because the old column was removed
C. 'mobile', the new column name
D. '0300-1111', exactly as before the rename

174. A desktop application on each of 50 PCs contains all the business rules and SQL, and it connects directly to the database server. What kind of client is this, and where is the business logic? Application

A. A fat client in a two-tier design; the logic is on each PC
B. A thin client in a three-tier design; the logic is on an application server
C. A thin client in a two-tier design; the logic is in the database
D. A fat client in a three-tier design; the logic is on an application server

175. A CREATE TABLE statement is submitted to the DBMS, and its definitions are recorded in the data dictionary. Which component in Silberschatz's structure does this? Application

    A. The query evaluation engine
    B. The DDL interpreter
    C. The buffer manager
    D. The transaction manager

176. A C program contains embedded SQL statements. Before compilation, a module pulls out the SQL statements and sends them to the DML compiler, leaving the rest for the C compiler. Which module is this? Application

    A. The query optimiser
    B. The stored data manager
    C. The precompiler
    D. The runtime database processor

177. Every month a DBA collects statistics on table sizes and access frequencies to decide whether to reorganise files or add indexes. Which DBMS utility supports this? Application

    A. The loading utility
    B. The precompiler
    C. The view definition language
    D. The performance-monitoring utility

178. A pharmacy chain runs the same DBMS product at each of its 20 branches. Each branch stores its own stock, and the branch databases work together as one logical database over the network. How is this DBMS classified? Application

    A. Homogeneous distributed
    B. Heterogeneous distributed
    C. Centralised
    D. Federated with full autonomy

179. A query on a view finds the matching stored records. In which order does the DBMS transform the results before showing them to the user? Application

    A. External view → conceptual format → internal format
    B. Internal format → conceptual format → external view
    C. Internal format → external view, skipping the conceptual level
    D. Conceptual format → internal format → external view

180. Which level of the three-schema architecture does each statement define? Application

```
CREATE TABLE emp (id INT, name TEXT, salary INT);   -- (1)
CREATE INDEX idx_emp_name ON emp(name);              -- (2)
CREATE VIEW emp_public AS SELECT id, name FROM emp;   -- (3)
```

    A. (1) internal, (2) conceptual, (3) external
    B. (1) conceptual, (2) external, (3) internal
    C. (1) conceptual, (2) internal, (3) external
    D. (1) external, (2) internal, (3) conceptual

181. In SQLite 3.35 or later, what happens when the ALTER TABLE runs? Application

```
CREATE TABLE student (roll INT, name TEXT, phone TEXT);
CREATE VIEW v AS SELECT roll, phone FROM student;
ALTER TABLE student DROP COLUMN phone;
```

A. It succeeds, and the view returns NULL for phone
B. It succeeds, and the view silently drops the phone column
C. It succeeds, and the view keeps showing the old phone values
D. It is rejected with an error, because the view v still uses phone

182. A database has 15 views. STUDENT is split into STUDENT and ADDRESS, stored as two files, and only the 4 views that use address columns are redefined. Which mappings change? Application

A. The 4 external/conceptual mappings of those views, and the conceptual/internal mapping
B. All 15 external/conceptual mappings, and nothing else
C. Only the 4 external/conceptual mappings
D. Only the conceptual/internal mapping

183. 200 clients each make 10 requests a minute, and each request needs 3 SQL queries. In the two-tier design, clients send SQL to the database across the WAN. In the three-tier design, each request is one round trip over the WAN to an application server in the same data centre as the database. How many round trips cross the WAN per minute? Application

A. 2,000 in two-tier and 6,000 in three-tier
B. 6,000 in two-tier and 2,000 in three-tier
C. 6,000 in both
D. 2,000 in both

184. A SELECT runs, and the page it needs is not in main memory. In Silberschatz's structure, in which order do these components take part? Application

A. Query evaluation engine → DML compiler → file manager → buffer manager
B. DML compiler (with optimiser) → query evaluation engine → buffer manager → file manager
C. Buffer manager → DML compiler → query evaluation engine → file manager
D. DDL interpreter → query evaluation engine → transaction manager → buffer manager

185. The view computes age from dob as of 1 January 2026. Which rolls does the final SELECT return? Application

```
CREATE TABLE student (roll INT, dob TEXT);
INSERT INTO student VALUES (1, '2004-03-10'), (2, '2006-11-30');
CREATE VIEW student_age AS
  SELECT roll,
         CAST((julianday('2026-01-01') - julianday(dob)) / 365.25 AS INT) AS age
  FROM student;
SELECT roll FROM student_age WHERE age >= 20 ORDER BY roll;
```

A. 1 only
B. 1 and 2
C. 2 only
D. Neither, because age is not a stored column

186. A student lists four facts about the levels of the three-schema architecture. Which one is FALSE? Analysis

A. There is one conceptual schema for the whole database
B. There can be many external schemas
C. Each user group has its own internal schema
D. The internal schema describes physical storage

187. An old program navigates a file by keeping stored record addresses in its variables. After the DBA reorganises the file, the program reads the wrong records. What does this show? Analysis
    A. Logical data independence failed, because an attribute was added
    B. The reorganisation changed the conceptual schema
    C. The DBA should never reorganise files
    D. The program bypassed the conceptual level and depended on internal details, so it lost physical data independence

188. Which statement about database states is correct, according to Elmasri-Navathe? Analysis
    A. A valid state satisfies the structure and constraints in the schema, and the DBMS is partly responsible for keeping every state valid
    B. The schema changes every time a row is inserted
    C. Only the initial state needs to satisfy the constraints
    D. An empty database has no state

189. A DBA says: 'I created a view for each department, so we now have physical data independence.' Which evaluation is correct? Analysis
    A. Correct: views hide how data is stored on disk
    B. Wrong: views define external schemas and support logical data independence; physical data independence comes from the conceptual/internal mapping
    C. Wrong: views provide neither kind of independence
    D. Correct: views store the data in a separate physical file

190. Program P loops with 'GET NEXT student' and tests city = 'Lahore' itself. Program S runs SELECT * FROM student WHERE city = 'Lahore'. Which comparison is correct? Analysis
    A. Both are set-at-a-time, because both return all Lahore students
    B. P is declarative and S is procedural
    C. P uses a procedural, record-at-a-time DML and fixes the access order itself; S is declarative and set-at-a-time, so the optimiser can choose an index
    D. S must read every row, while P can use an index automatically

191. A college moves its business rules from 300 fat desktop clients to one application server. Which consequence is correct? Analysis
    A. Clients must now send SQL directly to the database
    B. The database server must now run the business rules
    C. The clients become fat clients
    D. A rule change is deployed once on the server instead of on 300 PCs

192. Which pairing of a change with the mapping that must be updated is WRONG? Analysis
    A. Moving a table to SSDs → the conceptual/internal mapping
    B. Adding an index → the external/conceptual mapping
    C. Splitting a table and redefining its views → the external/conceptual mapping
    D. Changing a heap file to a hashed file → the conceptual/internal mapping

193. What does the final SELECT return? Analysis

```
CREATE TABLE s (roll INT, name TEXT, cgpa REAL);
INSERT INTO s VALUES (1, 'Ali', 3.2), (2, 'Sara', 3.8), (3, 'Omar', 2.9);
CREATE VIEW v1 AS SELECT roll, cgpa FROM s WHERE cgpa >= 3.0;
CREATE VIEW v2 AS SELECT roll FROM v1 WHERE cgpa < 3.5;
UPDATE s SET cgpa = 3.1 WHERE roll = 3;
SELECT roll FROM v2 ORDER BY roll;
```

A. 1 and 3
B. 1 only
C. 1, 2 and 3
D. 3 only

194. In SQLite, the query below returned rolls 1, 3, 4 before the index and 4, 3, 1 after it. Which conclusion is correct? Analysis

```
CREATE TABLE s (roll INT, city TEXT, name TEXT);
INSERT INTO s VALUES (1, 'Lahore', 'Zara'), (2, 'Quetta', 'Ali'),
                     (3, 'Lahore', 'Bilal'), (4, 'Lahore', 'Amna');
SELECT roll FROM s WHERE city = 'Lahore';
CREATE INDEX ix ON s(city, name);
SELECT roll FROM s WHERE city = 'Lahore';
```

A. The index changed the query's result, so physical data independence failed
B. The index corrupted the table and should be dropped
C. The set of rows is the same; without ORDER BY, the order is not guaranteed and can change with the access path
D. SQLite always returns rows in reverse after an index is created

195. All programs query base tables directly, and no views are defined. Which kind of data independence is weakened, and which still holds? Analysis

A. Physical independence is lost, because there is no external level; logical independence still holds
B. Both kinds are lost, because views are needed for any independence
C. Neither is affected, because the catalog provides both
D. Logical independence is weakened, because conceptual changes reach programs directly; physical independence still holds

196. A table is changed from a B+ tree file to a hashed file on roll. A query with roll BETWEEN 100 AND 200 still returns the same rows but becomes much slower. Which analysis is correct? Analysis

A. Physical data independence failed, because performance changed
B. Physical data independence held, but hashing does not support range searches, so the DBA chose a poor organisation for this workload
C. Logical data independence failed, because the query had to be rewritten
D. Hashing changed the rows returned, so the results only look the same

197. A program needs to UPDATE rows through a view that joins two tables and computes SUM(salary) per department. Which limit of the external level does this show? Analysis

A. Views cannot be queried by programs
B. Views always store their own copy of the data, so updates are lost
C. Views with joins and aggregates are generally not updatable, because an update cannot be mapped back to unique base rows
D. Any view can be updated as long as the user has privileges

198. A program reads view student_advisor(roll, advisor), assuming one advisor per student. Which conceptual change is hardest to hide from it by redefining the view? Analysis
    A. Adding an email attribute to STUDENT
    B. Moving STUDENT to a faster disk
    C. Splitting STUDENT into two tables that can be joined back on roll
    D. Changing the student-advisor relationship from one-to-many to many-to-many

199. Which classification of these three specifications is correct? (1) An ER diagram of students and courses. (2) CREATE TABLE statements for those relations. (3) A specification of a B+ tree index and record layout on disk. Analysis
    A. (1) representational, (2) conceptual, (3) physical
    B. (1) conceptual, (2) physical, (3) representational
    C. (1) conceptual (high-level), (2) representational, (3) physical
    D. All three are representational

200. After the optimiser's statistics in the catalog are deleted, a query returns the same rows but runs ten times slower. Which explanation is most likely? Analysis
    A. The evaluation engine now returns wrong rows more slowly
    B. The buffer manager deleted all cached pages permanently
    C. The DDL interpreter must recompile the table definitions on every query
    D. The query optimiser lacks the statistics it uses to estimate costs, so it picked a poor plan

Data Models MCQs for the NSCT

201. What is a data model, in Elmasri-Navathe's sense? Understanding
    A. The data stored in a database at one moment
    B. A diagram of the hardware a DBMS runs on
    C. A program that converts files from one format to another
    D. A collection of concepts for describing the structure of a database (data types, relationships, constraints) and, usually, basic operations

202. In Elmasri-Navathe's classification of data models, which category does the relational model belong to? Understanding
    A. High-level (conceptual) data model
    B. Low-level (physical) data model
    C. Representational (implementation) data model
    D. Self-describing data model

203. In Chen-style ER notation as used by Elmasri-Navathe, what does a double oval represent? Understanding
    A. A derived attribute
    B. A key attribute
    C. A weak entity type
    D. A multivalued attribute

204. Age is computed from BirthDate and today's date. How is Age drawn in Elmasri-Navathe's ER notation? Understanding
    A. As a double oval (multivalued attribute)
    B. As an underlined oval (key attribute)
    C. As a dashed oval (derived attribute)
    D. As a diamond (relationship)

205. In Elmasri-Navathe's ER notation, what do a double rectangle and a double diamond represent? Understanding

A. A strong entity type and an M:N relationship
B. A weak entity type and its identifying relationship
C. A multivalued attribute and a composite attribute
D. An entity set and a recursive relationship

206. In Chen-style ER diagrams, what does a double line between an entity type and a relationship type mean? Understanding

A. Partial participation
B. A 1:1 cardinality ratio
C. Total participation: every entity must take part in the relationship
D. A derived relationship

207. How is a key attribute shown in a Chen-style ER diagram? Understanding

A. It is drawn as a double oval
B. It is drawn as a dashed oval
C. Its name is underlined inside the oval
D. It is drawn inside a diamond

208. In the formal relational model, what is a row of a table called? Understanding

A. An attribute
B. A domain
C. A relation schema
D. A tuple

209. What is the degree (arity) of a relation? Understanding

A. The number of tuples it currently holds
B. The number of candidate keys it has
C. The number of attributes in its schema
D. The number of relations it references

210. Which data model organises records as trees of parent-child relationships, as in IBM's IMS? Understanding

A. The network model
B. The relational model
C. The semi-structured model
D. The hierarchical model

211. How is a weak entity identified? Understanding

A. By combining its owner's key with its own partial key
B. By its own primary key alone
C. By a system-generated row number
D. By all of its attributes together, without reference to any other entity

212. Cardinality ratios and participation constraints together are called what? Understanding

A. Integrity rules of the relational model
B. Structural constraints of a relationship type
C. Mapping cardinalities of a schema
D. Functional dependencies

213. In Elmasri-Navathe's (min, max) notation, what does min = 0 mean? Understanding
    A. Partial participation: an entity need not take part in the relationship
    B. Total participation
    C. The relationship is 1:1
    D. The entity type is weak

214. Which of these is NOT a property of a relation in the formal relational model? Understanding
    A. Its tuples are kept in a fixed order that queries depend on
    B. Its tuples are all distinct
    C. Each attribute value is atomic in the basic (flat) model
    D. Each value comes from the attribute's domain or is NULL

215. EMPLOYEE supervises EMPLOYEE. What kind of relationship type is this, and what does it need in the ER diagram? Understanding
    A. A ternary relationship that needs three entity types
    B. A recursive (unary) relationship that needs role names such as supervisor and supervisee
    C. A weak relationship that needs a double diamond
    D. An identifying relationship that needs a partial key

216. An app stores products as JSON documents, and different products have different fields. Which data model best describes this? Understanding
    A. The hierarchical model
    B. The semi-structured data model
    C. The network model
    D. The flat relational model

217. A STUDENT may have several phone numbers. How should PhoneNumbers be modelled in an ER diagram? Application
    A. As a composite attribute
    B. As a derived attribute
    C. As the key attribute
    D. As a multivalued attribute (double oval)

218. Name is broken into FirstName, MiddleName and LastName. What type of attribute is Name? Application
    A. Multivalued
    B. Derived
    C. Composite
    D. Simple (atomic)

219. STUDENT (key Roll) and COURSE (key Code) have an M:N relationship ENROLS with attribute Grade. Under standard ER-to-relational mapping, what is the primary key of the ENROLS relation? Application
    A. Roll
    B. Code
    C. (Roll, Code, Grade)
    D. (Roll, Code)

220. DEPARTMENT (1) to EMPLOYEE (N) through WORKS_FOR. Using the foreign-key approach, where does the foreign key go? Application
    A. In EMPLOYEE, referring to DEPARTMENT's key
    B. In DEPARTMENT, referring to EMPLOYEE's key
    C. In both relations
    D. In a new WORKS_FOR relation only

221. DEPENDENT is a weak entity owned by EMPLOYEE (key Ssn), with partial key DependentName. What is the primary key of the DEPENDENT relation? Application

    A. (Essn, DependentName), where Essn references EMPLOYEE
    B. DependentName
    C. Essn
    D. A new surrogate key only

222. DEPARTMENT (key Dnumber) has a multivalued attribute Locations. How is it mapped to relations? Application

    A. A Locations column holding a comma-separated list
    B. Columns Location1, Location2 and Location3 in DEPARTMENT
    C. A new relation DEPT_LOCATIONS(Dnumber, Location) with key (Dnumber, Location)
    D. A new relation with key Location only

223. Strong entity types: STUDENT, COURSE, TEACHER, DEPARTMENT. Relationships: STUDENT M:N COURSE; TEACHER 1:N COURSE; TEACHER N:1 DEPARTMENT; COURSE N:1 DEPARTMENT. STUDENT has one multivalued attribute, Phones. What is the minimum number of relations after standard mapping? Application

    A. 5
    B. 8
    C. 6
    D. 9

224. Elmasri-Navathe notation: EMPLOYEE (1,1) — WORKS_FOR — (4,N) DEPARTMENT. There are 12 departments. What is the smallest possible number of employees? Application

    A. 12
    B. 4
    C. 16
    D. 48

225. DEPARTMENT 1:N EMPLOYEE (WORKS_FOR). There are 5 departments and 40 employees. What is the largest possible number of WORKS_FOR relationship instances? Application

    A. 200
    B. 5
    C. 40
    D. 45

226. MANAGES is 1:1 between EMPLOYEE (partial participation) and DEPARTMENT (total participation). In the foreign-key approach, which placement avoids NULLs? Application

    A. Put the manager's employee key in DEPARTMENT
    B. Put the department number in EMPLOYEE
    C. Put each key in the other relation
    D. Create a separate MANAGES relation, which is required for 1:1

227. What are the degree and the cardinality of the course relation produced here? Application

```
CREATE TABLE course (
  code TEXT PRIMARY KEY, title TEXT, credits INT, dept TEXT
);
INSERT INTO course VALUES
  ('CS101', 'Programming', 3, 'CS'), ('CS201', 'Data Structures', 3, 'CS'),
  ('CS301', 'Databases', 3, 'CS'), ('MT101', 'Calculus', 3, 'MATH'),
  ('MT201', 'Linear Algebra', 3, 'MATH'), ('EN101', 'English', 2, 'HUM');
SELECT * FROM course;
```

A. Degree 6, cardinality 4
B. Degree 4, cardinality 6
C. Degree 4, cardinality 4
D. Degree 6, cardinality 6

228. SUPPLY is a ternary relationship among SUPPLIER, PART and PROJECT, with M:N:P cardinality and attribute Quantity. What is the primary key of the SUPPLY relation after mapping? Application

A. The supplier key only
B. The combination of the three participating keys
C. The supplier and part keys only
D. Quantity together with the project key

229. EMPLOYEE (composite Name, multivalued Skills, derived Age), DEPARTMENT (multivalued Locations), PROJECT, weak DEPENDENT. Relationships: WORKS_FOR (N:1), MANAGES (1:1), CONTROLS (DEPARTMENT 1:N PROJECT), WORKS_ON (M:N), SUPERVISION (recursive 1:N), DEPENDENTS_OF (identifying). Minimum relations after standard mapping? Application

A. 6
B. 7
C. 12
D. 9

230. STUDENT (0,6) — ENROLS — (5,40) SECTION. There are 10 sections and 70 students. What are the smallest and largest possible numbers of ENROLS instances? Application

A. 0 and 420
B. 50 and 420
C. 5 and 400
D. 50 and 400

231. Chen's 1:N label on DEPARTMENT—WORKS_FOR—EMPLOYEE puts '1' beside DEPARTMENT and 'N' beside EMPLOYEE. Each employee works in exactly one department. In Elmasri-Navathe's (min, max) notation, which pair is written beside EMPLOYEE? Analysis

A. (1,1)
B. (1,N)
C. (0,N)
D. (0,1)

232. PERSON and PASSPORT_RECORD are linked by a 1:1 relationship with total participation on both sides, and neither takes part in any other relationship. What is the minimum number of relations needed? Analysis

A. 2
B. 3
C. 0
D. 1

233. E1 (key a1) and E2 (key b1) are linked by R1, which is 1:N from E1 to E2, and by R2, which is M:N. Neither has multivalued attributes. What is the minimum number of relations in 1NF? Analysis

    A. 3
    B. 2
    C. 4
    D. 5

234. WORKS_ON is M:N between EMPLOYEE and PROJECT, with attribute Hours. Why can Hours not be moved into EMPLOYEE? Analysis

    A. An employee works on several projects with different hours, so one value per employee cannot hold them
    B. Attributes are never allowed on relationship types
    C. Hours is a derived attribute
    D. EMPLOYEE is a weak entity type

235. SUPPLY is replaced by its three binary projections SP, SJ and PJ. How many tuples does the join of the three return, compared with the 3 original SUPPLY tuples? Analysis

```
CREATE TABLE supply (s TEXT, p TEXT, j TEXT);
INSERT INTO supply VALUES ('s1','p1','j2'), ('s1','p2','j1'), ('s2','p1','j1');
CREATE TABLE sp AS SELECT DISTINCT s, p FROM supply;
CREATE TABLE sj AS SELECT DISTINCT s, j FROM supply;
CREATE TABLE pj AS SELECT DISTINCT p, j FROM supply;
SELECT COUNT(*) FROM sp
  JOIN sj ON sp.s = sj.s
  JOIN pj ON sp.p = pj.p AND sj.j = pj.j;
```

    A. 3, exactly the original tuples
    B. 2, because one tuple is lost
    C. 9, every combination
    D. 4, including one spurious tuple (s1, p1, j1)

236. The visit table has no key. What do the two COUNT queries return, and what is the cardinality of the formal relation with the same rows? Analysis

```
CREATE TABLE visit (patient TEXT, ward TEXT);
INSERT INTO visit VALUES ('Ali', 'W1'), ('Sara', 'W2'), ('Ali', 'W1');
SELECT COUNT(*) FROM visit;
SELECT COUNT(*) FROM (SELECT DISTINCT patient, ward FROM visit);
```

    A. 3 and 2; the formal relation has 2 tuples
    B. 3 and 3; the formal relation has 3 tuples
    C. 2 and 2; SQL drops duplicates automatically
    D. 3 and 2; the formal relation has 3 tuples

237. Why did the hierarchical model struggle with M:N relationships such as students and courses? Analysis

    A. It could not store numeric data
    B. A tree gives each child record one parent, so M:N needs duplicated records or virtual pointers
    C. It had no way to represent 1:N relationships
    D. It required every query to be written in SQL

238. Each loan's payments are numbered 1, 2, 3, … and the numbering restarts for every loan. How should PAYMENT be modelled? Analysis
    A. As a strong entity with PaymentNo as its key
    B. As a weak entity owned by LOAN through an identifying relationship, with PaymentNo as a partial key and total participation
    C. As a multivalued attribute of LOAN with no further structure
    D. As a strong entity linked to LOAN by an M:N relationship

239. A student writes: 'The cardinality of ENROLS is M:N, and the cardinality of the STUDENT table is 500.' Which assessment is correct? Analysis
    A. Only the first is correct; tables do not have a cardinality
    B. Both are standard uses of 'cardinality', with different meanings: a relationship's cardinality ratio, and a relation's number of tuples
    C. Only the second is correct; relationships have degree, not cardinality
    D. Both are wrong; 'cardinality' means the number of attributes

240. EMPLOYEE (1,1) — WORKS_FOR — (4,N) DEPARTMENT, and the company has 30 employees. What is the largest number of departments that can exist without breaking the constraints? Analysis
    A. 8
    B. 30
    C. 7
    D. 4

241. In Elmasri-Navathe's terms, how does a database schema differ from a database state? Understanding
    A. The schema is the data stored at a moment; the state is its description
    B. The schema is the physical file layout; the state is the ER diagram
    C. Both mean the set of tables, but the state also includes the indexes
    D. The schema is the description, which rarely changes; the state is the data stored at a particular moment

242. Which category of data model describes how data is stored in files, including record formats, record orderings and access paths? Understanding
    A. A conceptual (high-level) data model
    B. A representational (implementation) data model
    C. A semi-structured data model
    D. A physical (low-level) data model

243. In the network (CODASYL) model, what links an owner record type to its member record type? Understanding
    A. A foreign key column that matches a primary key
    B. A set type, which represents a 1:N relationship
    C. A tree in which each child has exactly one parent segment
    D. A relationship diamond with participation lines

244. SUPPLIER, PART and PROJECT all take part in one relationship type, SUPPLY. What is the degree of SUPPLY? Understanding
    A. 2, so it is binary
    B. 3, so it is ternary
    C. 1, so it is unary
    D. It depends on how many SUPPLY instances exist

245. Elmasri-Navathe write complex attributes in text form, for example {AddressPhone({Phone(AreaCode, Number)}, Address(Street, City))}. What do braces { } and parentheses ( ) mark? Understanding
    A. Braces mark composite attributes; parentheses mark multivalued attributes
    B. Braces mark multivalued attributes; parentheses group the components of a composite attribute
    C. Braces mark key attributes; parentheses mark derived attributes
    D. Braces mark optional (NULL-able) attributes; parentheses mark stored attributes

246. In Elmasri-Navathe's terms, what is the collection of all EMPLOYEE entities stored in the database at a given moment called? Understanding
    A. The entity set (the extension of the entity type)
    B. The entity type (the intension)
    C. The key attribute of EMPLOYEE
    D. The relationship set of EMPLOYEE

247. The attributes of a 1:N relationship type may be moved into one of the participating entity types. Into which one? Understanding
    A. The entity type on the 1-side
    B. Either side, with no difference in meaning
    C. Neither; they must always stay in a separate relation
    D. The entity type on the N-side

248. Why does a weak entity type always have total participation in its identifying relationship? Understanding
    A. Its partial key is unique across all entities of the weak type
    B. Identifying relationships are always M:N
    C. Weak entity types have no attributes of their own
    D. A weak entity cannot be identified without its owner, so every weak entity must be related to one

249. In an EER specialization, what does a circle containing the letter 'd' mean? Understanding
    A. Disjoint: an entity can belong to at most one of the subclasses
    B. Derived: subclass membership is computed from an attribute
    C. Overlapping: an entity can belong to several subclasses
    D. Total: every superclass entity must belong to some subclass

250. How does generalization differ from specialization in EER modelling? Understanding
    A. Generalization is top-down and specialization is bottom-up
    B. Generalization applies only to weak entity types
    C. Specialization creates new relationship types, while generalization creates new attributes
    D. Generalization is bottom-up: it finds a common superclass for existing entity types. Specialization is top-down: it defines subclasses of an existing entity type

251. ENGINEER is a subclass of EMPLOYEE in an EER schema. What does ENGINEER inherit? Understanding
    A. Only the key attribute of EMPLOYEE
    B. Only EMPLOYEE's relationships, not its attributes
    C. Nothing; a subclass must redeclare every attribute
    D. All the attributes and relationships of EMPLOYEE

252. A vehicle's OWNER can be a PERSON, a BANK or a COMPANY, and these three have different keys. Which EER construct models OWNER? Understanding
    A. A specialization of OWNER into the subclasses PERSON, BANK and COMPANY
    B. A category (union type) whose superclasses are PERSON, BANK and COMPANY
    C. A weak entity type owned by all three
    D. An overlapping specialization of PERSON

253. Why is XML or JSON data described as self-describing? Understanding

A. Schema information, such as tag names or key names, is stored together with the data values
B. It always comes with a separate DDL file that defines its structure
C. Each document must first be validated against a relational schema
D. Its values are kept sorted, so the structure can be inferred

254. In the formal relational model, what does R(A1, A2, ..., An) denote? Understanding

A. A relation state holding n tuples
B. A single tuple with n values
C. A domain containing n values
D. A relation schema: the relation name R with the attributes A1 to An

255. Who proposed the relational model of data, and in which year? Understanding

A. Peter Chen, in 1976
B. Charles Bachman, in 1964
C. E. F. Codd, in 1970
D. Elmasri and Navathe, in 1989

256. MANAGES between EMPLOYEE and DEPARTMENT has the cardinality ratio 1:1. What does this ratio state? Understanding

A. Every employee must manage exactly one department
B. The relationship set contains exactly one instance
C. Each department has at least one manager and may have several
D. An employee manages at most one department, and a department has at most one manager

257. When a weak entity type is mapped to a relation, which referential triggered action do Elmasri-Navathe say is commonly chosen for its foreign key to the owner? Understanding

A. CASCADE, because a weak entity cannot exist without its owner
B. SET NULL, so the weak entity survives with no owner
C. SET DEFAULT, so the weak entity moves to a default owner
D. None, because the weak relation has no foreign key

258. Which of Elmasri-Navathe's approaches for mapping a binary 1:1 relationship creates a third relation that holds the keys of both participating relations? Understanding

A. The foreign key approach
B. The merged relation approach
C. The cross-reference (relationship relation) approach
D. The multivalued attribute approach

259. EMPLOYEE is specialised into SECRETARY, ENGINEER and TECHNICIAN. Which relations does Elmasri-Navathe's mapping option 8A create? Understanding

A. Only one relation per subclass, each repeating all the EMPLOYEE attributes
B. A single EMPLOYEE relation with a JobType attribute and every subclass attribute
C. A single EMPLOYEE relation with one Boolean flag per subclass
D. An EMPLOYEE relation, plus one relation per subclass holding the EMPLOYEE key and that subclass's own attributes

260. How is a composite attribute such as Address(Street, City, Zip) handled in the standard ER-to-relational mapping? Understanding

A. It becomes a separate ADDRESS relation keyed by Address
B. It becomes one column holding the joined string
C. It is dropped, because composite attributes cannot be mapped
D. Only its simple components, Street, City and Zip, become columns

261. How is a derived attribute such as Age usually handled when an ER schema is mapped to relations? Understanding

A. It becomes a separate relation keyed by the entity's key
B. It is not stored; it is computed from the stored attribute, such as BirthDate, when needed
C. It becomes part of the entity's primary key
D. It is mapped like a multivalued attribute

262. How is the partial key of a weak entity type drawn in Chen-style notation? Understanding

A. Underlined with a solid line
B. Inside a double oval
C. Inside a dashed oval
D. Underlined with a dashed (or dotted) line

263. In the network (CODASYL) model, every set type is 1:N. How is an M:N relationship such as STUDENT–COURSE represented? Understanding

A. With an extra link (intersection) record type that is the member of two set types
B. With one set type whose member records have several owners in that set
C. With a foreign key stored in each of the two record types
D. It cannot be represented in the network model at all

264. In the ER model, what is the value set of an attribute? Understanding

A. The set of values that may be assigned to that attribute for each entity
B. The values currently stored in the attribute
C. The set of entities that have the attribute
D. The list of attributes that form the key

265. A single-family house has no apartment number, so ApartmentNumber is NULL for it. Which meaning of NULL in Elmasri-Navathe applies? Application

A. Unknown: the value exists but is missing
B. Not applicable
C. Unknown: it is not known whether the value exists
D. Zero

266. CAR has VehicleIdNumber and the composite attribute Registration(State, Number). Each one on its own identifies a car. How is this shown in the ER diagram? Application

A. Only one may be underlined; the other must be drawn as a derived attribute
B. Together they form one composite key, (VehicleIdNumber, Registration)
C. CAR must become a weak entity type
D. Both are key attributes and both are underlined, because an entity type may have more than one key

267. SUPERVISION is a recursive 1:N relationship on EMPLOYEE (key Ssn). How is it mapped using the foreign-key approach? Application

A. Create SUPERVISION(Ssn, SuperSsn) with the key (Ssn, SuperSsn)
B. Add a multivalued SupervisedSsns column to EMPLOYEE
C. Add SuperSsn to EMPLOYEE as a foreign key that references EMPLOYEE's Ssn
D. Create a new SUPERVISOR relation that copies EMPLOYEE

268. COURSE (key CourseNo) has a recursive M:N relationship PREREQUISITE: a course can have many prerequisites and can be a prerequisite for many courses. What relation does the mapping create? Application

A. A PrereqNo column added to COURSE
B. PREREQ(CourseNo, PrereqNo), with key (CourseNo, PrereqNo) and both columns referencing COURSE
C. PREREQ(CourseNo, PrereqNo), with key CourseNo alone
D. No relation, because recursive relationships need no mapping

269. EMPLOYEE has Ssn (key), the composite Name(Fname, Lname), Salary, BirthDate, the derived Age and the multivalued Skills. After standard mapping, what is the degree of the EMPLOYEE relation? Application

A. 4
B. 5
C. 6
D. 7

270. BUILDING (key Bno) owns the weak entity type ROOM (partial key RoomNo). ROOM in turn owns the weak entity type SEAT (partial key SeatNo). What is the primary key of the SEAT relation? Application

A. (RoomNo, SeatNo)
B. (Bno, RoomNo, SeatNo)
C. SeatNo
D. (Bno, SeatNo)

271. Entity types: AUTHOR, BOOK and PUBLISHER. AUTHOR M:N BOOK (WRITES), BOOK N:1 PUBLISHER, and PUBLISHER has a multivalued attribute Phones. There are no other relationships. What is the minimum number of relations after standard mapping? Application

A. 5
B. 4
C. 6
D. 3

272. Entity types: EMPLOYEE, DEPARTMENT and the weak DEPENDENT (owner EMPLOYEE). MANAGES is 1:1, WORKS_FOR is N:1, MENTORS is a recursive M:N on EMPLOYEE, DEPARTMENT has a multivalued Locations, and EMPLOYEE has a composite Name. What is the minimum number of relations? Application

A. 6
B. 4
C. 5
D. 7

273. Elmasri-Navathe notation: PROFESSOR (1,3) — TEACHES — (1,1) COURSE. There are 25 courses. What are the smallest and largest possible numbers of professors? Application

A. 9 and 25
B. 8 and 25
C. 9 and 75
D. 1 and 25

274. STUDENT (0,2) — JOINS — (3,10) CLUB, with 40 students and 6 clubs. What are the smallest and largest possible numbers of JOINS instances? Application

A. 0 and 80
B. 18 and 60
C. 18 and 80
D. 0 and 60

275. A UML class diagram shows 1 next to DEPARTMENT and 1..* next to EMPLOYEE on the WORKS_FOR association. What does it say? Application

    A. Each employee works for exactly one department, and each department has one or more employees
    B. Each department has exactly one employee, and each employee works for one or more departments
    C. Each employee works for one or more departments
    D. Each department has at most one employee, and each employee has at most one department

276. An early ER draft gives EMPLOYEE an attribute Department that holds the department's name, while DEPARTMENT is also an entity type. What do Elmasri-Navathe's refinement guidelines do with this attribute? Application

    A. Replace it with a relationship type between EMPLOYEE and DEPARTMENT
    B. Make Department a multivalued attribute of EMPLOYEE
    C. Make DEPARTMENT a weak entity type owned by EMPLOYEE
    D. Keep it, since attributes that refer to other entity types are standard

277. DEPARTMENT has a multivalued attribute Locations. Department 1 is in {Lahore}, department 4 in {Karachi, Lahore} and department 5 in {Lahore, Multan, Islamabad}. After standard mapping, how many tuples does DEPT_LOCATIONS(Dnumber, Location) hold? Application

    A. 3
    B. 4
    C. 9
    D. 6

278. WORKS_ON is mapped from an M:N relationship. Each INSERT runs as its own statement, and a statement that fails changes nothing. How many rows does works_on hold at the end? Application

```
CREATE TABLE works_on (
  essn INT, pno INT, hours REAL,
  PRIMARY KEY (essn, pno)
);
INSERT INTO works_on VALUES (1, 10, 20), (1, 20, 10), (2, 10, 15);
INSERT INTO works_on VALUES (2, 20, 5);
INSERT INTO works_on VALUES (1, 10, 8);
SELECT COUNT(*) FROM works_on;
```

    A. 5
    B. 4
    C. 3
    D. 2

279. EMPLOYEE(Ssn, Name) is specialised disjointly into SECRETARY (TypingSpeed), TECHNICIAN (TGrade) and ENGINEER (EngType). Using option 8C, a single relation with one type attribute, what is the degree of the resulting EMPLOYEE relation? Application

    A. 5
    B. 3
    C. 6
    D. 8

280. The same EMPLOYEE specialization into SECRETARY, TECHNICIAN and ENGINEER is total and disjoint. How many relations does option 8B (subclass relations only) create? Application

    A. 4
    B. 1
    C. 3
    D. 2

281. Each passenger can hold many tickets, and each ticket belongs to exactly one passenger. What is the cardinality ratio of PASSENGER : TICKET in HOLDS? Application
    A. N:1
    B. M:N
    C. 1:N
    D. 1:1

282. In HOLDS, every ticket must belong to some passenger, but a registered passenger need not have any ticket. What are the participation constraints? Application
    A. Total for PASSENGER, partial for TICKET
    B. Total for both
    C. Total for TICKET, partial for PASSENGER
    D. Partial for both

283. Each passenger may hold zero or many tickets, and each ticket belongs to exactly one passenger. In Elmasri-Navathe's (min, max) notation for HOLDS, which pairs are written? Application
    A. PASSENGER (0,N) and TICKET (1,1)
    B. PASSENGER (1,1) and TICKET (0,N)
    C. PASSENGER (1,N) and TICKET (0,1)
    D. PASSENGER (0,1) and TICKET (1,N)

284. Entity types: CUSTOMER, ACCOUNT, BRANCH and the weak TRANSACTION (owner ACCOUNT). CUSTOMER M:N ACCOUNT (DEPOSITOR) and ACCOUNT N:1 BRANCH. CUSTOMER has a composite Address and a multivalued Emails. What is the minimum number of relations? Application
    A. 5
    B. 7
    C. 6
    D. 8

285. Entity types: PERSON, PASSPORT, COUNTRY and AIRPORT. PERSON 1:1 PASSPORT has total participation on both sides, and PASSPORT is in no other relationship. AIRPORT N:1 COUNTRY. There is a ternary TRAVELS among PERSON, COUNTRY and AIRPORT. There are no multivalued attributes. What is the minimum number of relations? Application
    A. 5
    B. 3
    C. 4
    D. 6

286. Each student's university is stored as an attribute, University, of STUDENT. Now every university also needs its city, its ranking and a list of its programmes. What is the better design? Analysis
    A. Make University a composite attribute (Name, City, Ranking, Programmes)
    B. Make University a multivalued attribute of STUDENT
    C. Make UNIVERSITY an entity type and relate STUDENT to it
    D. Make UNIVERSITY a weak entity type owned by STUDENT

287. A designer marks DEPENDENT as a weak entity owned by EMPLOYEE. But every dependent also has a national ID number (CNIC) that is unique and always recorded. Which assessment is correct? Analysis
    A. DEPENDENT has its own key, so it is a strong entity type, linked to EMPLOYEE by an ordinary relationship
    B. It stays weak, because a dependent cannot exist without an employee
    C. It must be weak, because it is on the N-side of a 1:N relationship
    D. It must be modelled as a multivalued attribute of EMPLOYEE

288. Elmasri-Navathe notation: STUDENT (2,3) — TAKES — (5,5) LAB. There are 10 students and 7 labs. Can a valid TAKES relationship set exist? Analysis
    A. Yes, with 30 instances
    B. No: the labs need exactly 35 instances, but the students can supply at most 30
    C. Yes, with 35 instances, since (5,5) constrains only the labs
    D. No, because a student's max must equal a lab's min

289. A college must record which instructor taught which course in which semester. Design 1 uses a ternary TEACHES(Instructor, Course, Semester). Design 2 uses two binary relationships, INSTRUCTOR–COURSE and COURSE–SEMESTER. What is the problem with Design 2? Analysis
    A. Binary relationships cannot carry attributes
    B. When a course has several instructors across several semesters, it cannot say who taught it in which semester
    C. It needs more relations, so its queries are slower
    D. Ternary relationships cannot be mapped to relations, so Design 1 is impossible

290. EMPLOYEE is specialised disjointly into three subclasses, each with six specific attributes, and most employees belong to none of them. Which mapping option leaves the most NULLs? Analysis
    A. Option 8A: a superclass relation plus one relation per subclass
    B. Option 8B: one relation per subclass only
    C. Mapping each subclass attribute to its own relation
    D. Option 8C: one relation with a type attribute and all 18 subclass columns

291. VEHICLE is specialised into CAR and TRUCK, and the specialization is partial: some vehicles, such as motorbikes, are neither. Why is option 8B (relations for the subclasses only) a poor choice here? Analysis
    A. 8B cannot handle disjoint specializations
    B. 8B fills the CAR and TRUCK relations with NULLs
    C. Vehicles that are in no subclass would have no relation to store them, so they would be lost
    D. 8B needs a type attribute, which a partial specialization does not have

292. A team moves an application from a network (CODASYL) database to a relational one. What changes in how its queries are written? Analysis
    A. Queries must now follow pointers from parent records down to child records
    B. Queries can no longer combine data from two record types
    C. Nothing changes, because both models use the same navigational access
    D. Queries state what data is wanted by matching values, instead of following owner-member links one record at a time

293. EMPLOYEE has 500 rows and PARKING_SPOT has 50. ASSIGNED is 1:1, with partial participation for EMPLOYEE and total for PARKING_SPOT. A designer merges both into one relation. What is the main consequence? Analysis
    A. About 450 employee rows carry NULL in every parking column
    B. Nothing, since merging always suits a 1:1 relationship
    C. Parking spots with no employee are lost
    D. The merged relation needs the composite key (Ssn, SpotNo)

294. SUPPLY is a ternary relationship among SUPPLIER, PART and PROJECT. In (min, max) notation, PROJECT takes part with (0,1). What can the primary key of the SUPPLY relation be? Analysis
    A. All three foreign keys, as for every ternary relationship
    B. The supplier and part foreign keys together
    C. The foreign key that references PROJECT, on its own
    D. Nothing, because a relationship relation needs no key

295. Rule: each course section is taught by exactly one teacher, and a teacher teaches many sections. A designer creates TEACHES(TeacherId, SectionId) with key (TeacherId, SectionId). Which assessment is correct? Analysis

    A. It is correct and required, because every relationship needs its own relation
    B. It is wrong because the key should be TeacherId alone
    C. It is not needed: the relationship is 1:N, so TeacherId can be a foreign key in SECTION. Even in TEACHES, SectionId alone would be the key
    D. It is wrong because a 1:N relationship must be merged into the TEACHER relation

296. Sample data: one department has no employees, and every employee has a department. Which conclusion about WORKS_FOR participation is valid? Analysis

    A. Total participation for EMPLOYEE is proven by the data
    B. Total participation for DEPARTMENT is ruled out; total participation for EMPLOYEE fits the data but is not proven
    C. The relationship must be M:N
    D. DEPARTMENT participation is total, since most departments have employees

297. There are 12 teachers. TEACHER (2,4) — TEACHES — (1,1) SECTION. Which range gives every possible number of sections? Analysis

    A. 3 to 12
    B. 12 to 48
    C. 24 to 48
    D. 6 to 24

298. A designer maps the multivalued attribute Phones of STUDENT as three columns, Phone1, Phone2 and Phone3. Which problems does the standard mapping, STUDENT_PHONE(Roll, Phone), avoid? Analysis

    A. Entity integrity violations on Roll
    B. A hard limit of three numbers, many NULLs, and searches that must test three columns
    C. The need for any foreign key at all
    D. Loss of the STUDENT primary key

299. TotalCredits of STUDENT can be computed from ENROLS and COURSE, but a designer stores it as a column. What risk does this create? Analysis

    A. It can disagree with the enrolment data unless every change to ENROLS also updates it
    B. None, because derived attributes must always be stored
    C. It breaks entity integrity
    D. It makes STUDENT a weak entity type

300. DEPENDENT is mapped from a weak entity type. Foreign keys are enforced. What does the final query return, and why? Analysis

```
CREATE TABLE employee (ssn INT PRIMARY KEY, name TEXT);
CREATE TABLE dependent (
  essn INT REFERENCES employee(ssn) ON DELETE CASCADE,
  dep_name TEXT,
  PRIMARY KEY (essn, dep_name)
);
INSERT INTO employee VALUES (1, 'Ali'), (2, 'Sara');
INSERT INTO dependent VALUES (1, 'Zara'), (1, 'Omar'), (2, 'Zara');
DELETE FROM employee WHERE ssn = 1;
SELECT COUNT(*) FROM dependent;
```

A. 1: Sara's dependent Zara remains, because the key is (essn, dep_name), so the two Zaras are different tuples
B. 0: deleting employee 1 removes every dependent named Zara
C. 2: CASCADE removes only one matching dependent
D. 3: CASCADE needs a trigger before it deletes anything

Relational Database Concepts MCQs for the NSCT

301. In the formal relational model, what is a single row of a relation called? Understanding

A. An attribute
B. A degree
C. A tuple
D. A domain

302. What does the degree of a relation measure? Understanding

A. The number of tuples in its current state
B. The number of candidate keys it has
C. The number of relations that reference it
D. The number of attributes in its schema

303. Which term means the set of atomic values that an attribute is allowed to take? Understanding

A. Tuple
B. Superkey
C. Relation state
D. Domain

304. Which definition of a candidate key is correct? Understanding

A. A minimal superkey: removing any attribute stops it from uniquely identifying tuples
B. Any set of attributes that contains the primary key
C. Any attribute that has no NULL values in the current table
D. An attribute that refers to the primary key of another relation

305. What does the entity integrity constraint state? Understanding

A. Every attribute value must come from its domain
B. No two relations may share an attribute name
C. No primary key value can be NULL
D. Every foreign key must match an existing primary key

306. Under the referential integrity constraint, which values may a foreign key hold? Understanding

A. Any value from the same domain as the referenced primary key
B. A value that matches an existing primary key value in the referenced relation, or NULL
C. Only NULL until the referenced row is updated
D. Only values that do not already appear in the referencing relation

307. What is a composite key? Understanding

A. A primary key that is also a foreign key in every table
B. A key whose values are generated by the DBMS
C. A key made up of two or more attributes
D. A key that allows duplicate values

308. A relation currently holds 250 tuples. Which term describes this number? Understanding

A. Domain size
B. Arity
C. Cardinality
D. Degree

309. Which statement about superkeys and candidate keys is correct? Understanding

A. Every candidate key is a superkey, but a superkey need not be a candidate key
B. Superkeys and candidate keys are the same thing under different names
C. Every superkey is a candidate key, but a candidate key need not be a superkey
D. A relation can have only one superkey but several candidate keys

310. Which statement about a relation in the formal relational model is correct? Understanding

A. Each tuple must have a different number of attribute values
B. Two identical tuples may exist if they were inserted at different times
C. Its tuples have no inherent order
D. An attribute may store a set of values, such as several phone numbers

311. In EMPLOYEE(ssn, name, super_ssn), super_ssn holds the ssn of the employee's supervisor. What is super_ssn? Understanding

A. A composite primary key
B. A foreign key that references the primary key of its own relation
C. A derived attribute that needs no constraint
D. A candidate key of EMPLOYEE

312. What is a prime attribute? Understanding

A. An attribute that is never NULL
B. An attribute that belongs to the primary key only
C. An attribute with an integer domain
D. An attribute that belongs to at least one candidate key

313. A relation has three candidate keys, and the designer picks one as the primary key. What are the other two usually called? Understanding

A. Partial keys
B. Alternate keys (Elmasri-Navathe call them unique keys)
C. Superkeys that are not minimal
D. Foreign keys

314. According to Elmasri-Navathe, which operation on a referenced relation can violate only referential integrity? Understanding
    A. DELETE
    B. INSERT
    C. Changing a non-key attribute with UPDATE
    D. SELECT

315. The rule 'an employee's salary must not exceed their supervisor's salary' is which kind of constraint in Elmasri-Navathe's classification? Understanding
    A. A key constraint
    B. An application-based (semantic) constraint
    C. A domain constraint
    D. An entity integrity constraint

316. According to Elmasri-Navathe, which constraints can an INSERT operation violate? Understanding
    A. Domain, key, entity integrity and referential integrity
    B. Only referential integrity
    C. Only key and entity integrity
    D. Only domain constraints

317. Relation COURSE has the attributes (code, title, credits, dept, semester) and currently stores 120 tuples. What are its degree and cardinality? Application
    A. Degree 1, cardinality 120
    B. Degree 120, cardinality 5
    C. Degree 5, cardinality 600
    D. Degree 5, cardinality 120

318. STUDENT(roll_no PRIMARY KEY, name, cgpa) already holds roll_no 501. Which constraint does inserting another tuple with roll_no 501 violate? Application
    A. Key constraint
    B. Domain constraint
    C. Entity integrity
    D. Referential integrity

319. Attribute age is defined on the domain of integers from 17 to 60. Which constraint does inserting age = 'twenty' violate? Application
    A. Referential integrity
    B. Entity integrity
    C. Domain constraint
    D. Key constraint

320. A relation has three attributes whose domains contain 2, 3 and 4 values. What is the largest number of distinct tuples a state of this relation can hold? Application
    A. 4
    B. 24
    C. 9
    D. 12

321. Based only on this instance, which attribute set is definitely NOT a key of STUDENT? Application

```
roll | name | dept | email
101  | Ali  | CS   | ali@x.pk
102  | Sara | EE   | sara@x.pk
103  | Ali  | EE   | ali2@x.pk
104  | Omar | CS   | omar@x.pk
```

A. {email}
B. {roll}
C. {name, dept}
D. {name}

322. R(A, B, C, D) has exactly one candidate key, {A}. How many superkeys does R have? Application

A. 1
B. 8
C. 4
D. 15

323. R(A, B, C, D) has the composite primary key {A, B} and no other candidate key. How many superkeys does R have? Application

A. 8
B. 16
C. 4
D. 2

324. ENROL(student_id, course_id, grade) has foreign key student_id referencing STUDENT. STUDENT has no tuple with id 9001. What happens to INSERT INTO ENROL VALUES (9001, 'CS201', 'A')? Application

A. It violates referential integrity and is rejected
B. It is accepted because grade is not part of a key
C. It is accepted and a STUDENT tuple 9001 is created automatically
D. It violates entity integrity and is rejected

325. STUDENT has primary key roll_no. Which constraint does inserting a tuple with roll_no = NULL violate? Application

A. Domain constraint
B. Entity integrity
C. Key constraint
D. Referential integrity

326. Foreign keys are enforced. After the DELETE, how many rows remain in emp? Application

```
CREATE TABLE dept(dno INT PRIMARY KEY, dname TEXT);
CREATE TABLE emp(eid INT PRIMARY KEY, name TEXT,
  dno INT REFERENCES dept(dno) ON DELETE CASCADE);
INSERT INTO dept VALUES (10,'CS'), (20,'EE');
INSERT INTO emp VALUES (1,'Ali',10), (2,'Sara',20),
  (3,'Omar',10), (4,'Hina',10), (5,'Bilal',20);
DELETE FROM dept WHERE dno = 10;
```

A. 0
B. 5
C. 2
D. 3

327. Foreign keys are enforced. What is the state of emp after the DELETE? Application

```
CREATE TABLE dept(dno INT PRIMARY KEY, dname TEXT);
CREATE TABLE emp(eid INT PRIMARY KEY, name TEXT,
  dno INT REFERENCES dept(dno) ON DELETE SET NULL);
INSERT INTO dept VALUES (10,'CS'), (20,'EE');
INSERT INTO emp VALUES (1,'Ali',10), (2,'Sara',20), (3,'Omar',10);
DELETE FROM dept WHERE dno = 10;
```

A. 3 rows; Ali and Omar have dno 20
B. 1 row: Sara only
C. 3 rows, unchanged, because the DELETE is rejected
D. 3 rows; Ali and Omar have dno NULL

328. R(A, B, C) has the functional dependencies A → B and B → C. Which is the only candidate key? Application

A. {A, B}
B. {A}
C. {A, C}
D. {B}

329. R(A, B, C, D) has exactly two candidate keys, {A} and {B}. How many superkeys does R have? Application

A. 16
B. 12
C. 8
D. 14

330. ENROL(student_id, course_id, grade) has primary key (student_id, course_id), and student_id is also a foreign key to STUDENT. Can a tuple have student_id = NULL? Application

A. Yes, because foreign keys may always be NULL
B. Yes, as long as course_id is not NULL
C. No, because foreign keys may never be NULL
D. No, because student_id is part of the primary key and entity integrity forbids NULL there

331. R(A, B, C, D) has exactly two candidate keys: {A, B} and {C}. How many superkeys does R have? Analysis

A. 10
B. 6
C. 8
D. 12

332. R(A, B, C, D) has the functional dependencies A → B, B → C and C → A. What are its candidate keys? Analysis

A. {D} only
B. {A, D} only
C. {A, D}, {B, D} and {C, D}
D. {A}, {B} and {C}

333. In this instance A has a duplicate value and B does not. Which conclusion is valid? Analysis

```
A | B | C
1 | p | 10
2 | q | 10
1 | r | 20
```

A. {B} is the primary key of R
B. {A} becomes a key once the duplicate row is deleted
C. {A} is not a key, but the instance cannot prove that {B} is a key
D. Neither {A} nor {B} can ever be a key

334. Considering only this instance, which attribute sets are minimal sets whose values are unique across all four tuples? Analysis

```
A | B | C
1 | x | p
2 | x | q
1 | y | q
3 | y | p
```

A. {A, B} only
B. {A}, {B} and {C}
C. {A, B, C} only
D. {A, B}, {A, C} and {B, C}

335. In the formal relational model, a student claims: 'If a relation has no declared primary key, it has no superkey at all.' Which response is correct? Analysis

A. Wrong: since a relation is a set with no duplicate tuples, the set of all its attributes is always a superkey
B. Right: superkeys exist only after a PRIMARY KEY is declared
C. Wrong: every single attribute is always a superkey
D. Right: without a key the relation may contain duplicate tuples

336. Foreign keys are enforced. The four statements run in order. Which dno values remain in dept at the end? Analysis

```
CREATE TABLE dept(dno INT PRIMARY KEY, dname TEXT);
CREATE TABLE emp(eid INT PRIMARY KEY, name TEXT,
  dno INT REFERENCES dept(dno) ON DELETE RESTRICT);
INSERT INTO dept VALUES (10,'CS'), (20,'EE'), (30,'ME');
INSERT INTO emp VALUES (1,'Ali',10), (2,'Sara',10), (3,'Omar',30);
-- statements:
DELETE FROM dept WHERE dno = 20;
DELETE FROM dept WHERE dno = 10;
UPDATE emp SET dno = 10 WHERE eid = 3;
DELETE FROM dept WHERE dno = 30;
```

A. 10 only
B. None
C. 30 only
D. 10 and 30

337. Foreign keys are enforced. How many rows in total are removed from the three tables by DELETE FROM dept WHERE dno = 1? Analysis

```
CREATE TABLE dept(dno INT PRIMARY KEY);
CREATE TABLE emp(eid INT PRIMARY KEY,
  dno INT REFERENCES dept(dno) ON DELETE CASCADE);
CREATE TABLE dependent(eid INT REFERENCES emp(eid) ON DELETE CASCADE,
  dname TEXT, PRIMARY KEY (eid, dname));
INSERT INTO dept VALUES (1), (2);
INSERT INTO emp VALUES (11,1), (12,1), (13,2);
INSERT INTO dependent VALUES (11,'Amna'), (11,'Hamza'),
  (12,'Zoya'), (13,'Ahmed');
```

A. 4
B. 1
C. 3
D. 6

338. DEPARTMENT(dnum PK, mgr_ssn NOT NULL FK → EMPLOYEE) and EMPLOYEE(ssn PK, dno NOT NULL FK → DEPARTMENT) are both empty, and constraints are checked immediately. Why can no first row be inserted, and what is a standard fix? Analysis

A. The primary keys clash; give both tables the same primary key
B. Each insert needs a row that does not exist yet; allow NULL in one FK (or defer the check), insert, then update
C. The domains differ; change dno and mgr_ssn to the same data type
D. Circular foreign keys are illegal in the relational model; delete one of them

339. EMP.dno is a foreign key to DEPT.dnumber with no ON UPDATE action, so the default (reject) applies. Employees exist in department 5. Which update on DEPT is rejected? Analysis

A. Changing department 5's location
B. Changing the dnumber of a department that no employee references
C. Changing department 5's name to 'Research'
D. Changing department 5's dnumber to 7

340. EMP(ssn, name, phones) stores values such as '0300-1234567, 0321-7654321' in phones. What is the problem, and which design follows the relational model? Analysis

A. phones is not atomic; move it to EMP_PHONE(ssn, phone) with primary key (ssn, phone)
B. phones should be a foreign key to a PHONE_COMPANY table
C. phones is too long; change its type to a larger VARCHAR
D. ssn is not unique; make (ssn, phones) the primary key of EMP

341. In Elmasri-Navathe's terms, what does a relational database schema S consist of? Understanding

A. A set of relation states at one moment
B. A single table holding every attribute
C. The index files kept by the DBMS
D. A set of relation schemas together with a set of integrity constraints

342. 'A relation cannot contain duplicate tuples.' In Elmasri-Navathe's classification, what kind of constraint is this? Understanding

A. A schema-based (explicit) constraint
B. An inherent model-based (implicit) constraint
C. An application-based (semantic) constraint
D. A transition constraint

343. A column is declared NOT NULL in CREATE TABLE. In Elmasri-Navathe's classification, what kind of constraint is this? Understanding

A. An inherent model-based (implicit) constraint
B. A schema-based (explicit) constraint
C. An application-based (semantic) constraint
D. A transition (dynamic) constraint

344. Elmasri-Navathe list several meanings a NULL can carry. Which set matches theirs? Understanding

A. Zero, an empty string, or false
B. A default value, a deleted value, or a duplicate value
C. Only 'the attribute does not apply'
D. The value is unknown, it exists but is withheld, or the attribute does not apply to this tuple

345. According to Elmasri-Navathe, updating an attribute that is neither part of the primary key nor a foreign key can violate which constraint? Understanding

A. Only the domain constraint: the new value must have the right type and lie in the domain
B. The key constraint and entity integrity
C. Referential integrity
D. All four: domain, key, entity integrity and referential integrity

346. A DELETE would break referential integrity. Which of these is NOT one of the options Elmasri-Navathe list for handling it? Understanding

A. Restrict: reject the delete
B. Automatically rename the referenced key to a new value
C. Cascade: also delete the referencing tuples
D. Set null or set default in the referencing tuples

347. Must a foreign key have the same attribute name as the primary key it references? Understanding

A. Yes, the names must be identical
B. No, and the domain may differ as well
C. No. It must have the same domain, but the name may differ, for example Dno referencing Dnumber
D. Yes, unless the foreign key references its own relation

348. In SQL (SQLite included), which columns of the parent table may a FOREIGN KEY reference? Understanding

A. Only its PRIMARY KEY
B. Its PRIMARY KEY, or a column set declared UNIQUE
C. Any column, whether unique or not
D. Only an INTEGER PRIMARY KEY column

349. Rule: 'an employee's salary may only increase.' What kind of constraint is this? Understanding

A. A state (static) constraint
B. A domain constraint
C. A key constraint
D. A transition (dynamic) constraint, because it compares old and new values

350. In SQL, which constraint is used to declare an alternate (secondary candidate) key? Understanding

A. FOREIGN KEY
B. CHECK
C. UNIQUE
D. DEFAULT

351. A relation has the candidate keys {cnic} and {name, father_name, dob}. Which should be the primary key, following Elmasri-Navathe's advice? Understanding
    A. {name, father_name, dob}, since more attributes give stronger uniqueness
    B. {cnic}, since a key with one attribute or few attributes is usually better
    C. Either one, but entity integrity then requires the larger key
    D. Neither, because a surrogate key is compulsory

352. What do Elmasri-Navathe call a database state that satisfies every integrity constraint in the schema? Understanding
    A. A consistent schema
    B. An empty state
    C. An initial state
    D. A valid state

353. Under Elmasri-Navathe's alternative definition, a tuple is a set of (attribute, value) pairs. What does this definition make unnecessary? Understanding
    A. The ordering of tuples within a relation
    B. The ordering of values within a tuple
    C. Primary keys
    D. Domains

354. EMP.dno references DEPT, and no relation references EMP. Deleting a tuple from EMP can violate which constraint? Understanding
    A. Referential integrity
    B. Entity integrity
    C. None, because removing a referencing tuple cannot break referential integrity
    D. The key constraint

355. DEPT(dno PK) is referenced by EMP.dno, and DEPT has no foreign keys of its own. Which constraint can inserting a new DEPT tuple NOT violate? Understanding
    A. Referential integrity
    B. The key constraint
    C. Entity integrity
    D. The domain constraint

356. How does a SQL table differ from a relation in the formal relational model? Understanding
    A. Table rows have no order, but relation tuples are ordered
    B. Tables cannot contain NULLs
    C. Relations allow duplicate tuples, but tables do not
    D. A SQL table can hold duplicate rows unless a key or UNIQUE constraint prevents it

357. The attributes Home_phone and Office_phone both take values from the domain Local_phone_numbers. Is this allowed? Understanding
    A. No, each domain may belong to only one attribute
    B. Yes. Several attributes may share a domain, and their names show the different roles
    C. Only if the two attributes are in different relations
    D. Only if one of them is a foreign key

358. How are application-based (semantic) constraints usually enforced, according to Elmasri-Navathe? Understanding
    A. In application programs, or with triggers and assertions in SQL
    B. By declaring them as PRIMARY KEY constraints
    C. Automatically, by the relational model itself
    D. By choosing suitable domains for the attributes

359. A relation has degree 6 and currently holds no tuples. Which statement is true? Understanding

A. Its cardinality is 0, and its degree is still 6
B. Its degree is 0
C. It is not a valid relation
D. Its cardinality is 6

360. What does ON DELETE SET DEFAULT do when a referenced row is deleted? Understanding

A. It restores the deleted row with default values
B. It sets the foreign key in each referencing row to that column's default value
C. It sets every column of the referencing rows to its default
D. It deletes the referencing rows whose foreign key equals the default

361. What does declaring a foreign key DEFERRABLE INITIALLY DEFERRED change? Understanding

A. The constraint is never checked
B. The constraint is checked at COMMIT, not after each statement
C. The constraint is checked only on DELETE
D. The constraint is checked before each statement starts

362. Which attributes does the entity integrity constraint apply to? Understanding

A. Every attribute of every candidate key
B. Only the first attribute of a composite primary key
C. Every attribute of the relation
D. Every attribute of the primary key, but not alternate (UNIQUE) keys

363. EMP.dno is a foreign key that references DEPT.dno. Which relation is the referencing relation? Understanding

A. DEPT
B. Both
C. EMP
D. Neither, because the term applies only to self-references

364. What are a relation schema and a relation state also called? Understanding

A. The intension and the extension
B. The extension and the intension
C. The domain and the degree
D. The superkey and the key

365. No natural attribute of STUDENT is reliably unique, so a designer adds student_id, a number the system generates automatically. What is student_id? Application

A. A surrogate key
B. A foreign key
C. A natural key
D. A partial key

366. Foreign keys are enforced. What does the final query return? Application

```
CREATE TABLE dept (dno INT PRIMARY KEY, dname TEXT);
CREATE TABLE emp (eid INT PRIMARY KEY, name TEXT,
  dno INT REFERENCES dept(dno) ON UPDATE CASCADE);
INSERT INTO dept VALUES (10, 'CS'), (20, 'EE');
INSERT INTO emp VALUES (1, 'Ali', 10), (2, 'Sara', 20),
  (3, 'Omar', 10), (4, 'Hina', 10), (5, 'Bilal', 20);
UPDATE dept SET dno = 30 WHERE dno = 10;
SELECT dno, COUNT(*) FROM emp GROUP BY dno ORDER BY dno;
```

A. (10, 3) and (20, 2)
B. (20, 2) and (NULL, 3)
C. (20, 2) and (30, 3)
D. The UPDATE is rejected, so (10, 3) and (20, 2)

367. R(A, B, C, D, E, F) has exactly one candidate key, {A, B, C}. How many superkeys does R have? Application

A. 64
B. 3
C. 7
D. 8

368. R(A, B, C, D, E) has exactly two candidate keys, {A} and {B}. How many superkeys does R have? Application

A. 32
B. 16
C. 8
D. 24

369. R(A, B, C, D, E) has exactly two candidate keys, {A, B} and {B, C}. How many superkeys does R have? Application

A. 16
B. 8
C. 4
D. 12

370. R(A, B, C, D) has exactly three candidate keys, {A}, {B} and {C}. How many superkeys does R have? Application

A. 24
B. 14
C. 15
D. 12

371. R(A, B, C, D) has exactly two candidate keys, {A, B} and {C, D}. How many superkeys does R have? Application

A. 8
B. 6
C. 4
D. 7

372. R(A, B, C, D, E) has the functional dependencies A → B, BC → E and ED → A. What are its candidate keys? Application

A. {C, D} only
B. {A, C, D}, {B, C, D} and {C, D, E}
C. {A, C, D} only
D. {A, B, C, D, E} only

373. R(A, B, C, D) has the functional dependencies AB → C, C → D and D → A. What are its candidate keys? Application

A. {A, B}, {B, C} and {B, D}
B. {A, B} only
C. {B} only
D. {A, B} and {C, D}

374. R(A, B, C, D, E) has the functional dependencies A → BC, CD → E, B → D and E → A. How many candidate keys does R have? Application

A. 1
B. 2
C. 5
D. 4

375. R(A, B, C, D) has the functional dependencies A → B, B → C and C → D. Which set is a superkey but NOT a candidate key? Application

A. {A}
B. {B, C, D}
C. {A, C}
D. {B, C}

376. Which constraint does the UPDATE violate? Application

```
CREATE TABLE student (roll INT PRIMARY KEY, name TEXT);
INSERT INTO student VALUES (501, 'Ali'), (502, 'Sara');
UPDATE student SET roll = 501 WHERE roll = 502;
```

A. Entity integrity
B. The key constraint
C. Referential integrity
D. The domain constraint

377. Foreign keys are enforced, and no ON DELETE action is declared. Each statement runs on its own, and a failing statement changes nothing. What does the final query return? Application

```
CREATE TABLE dept (dno INT PRIMARY KEY, dname TEXT);
CREATE TABLE emp (eid INT PRIMARY KEY, name TEXT,
  dno INT REFERENCES dept(dno));
INSERT INTO dept VALUES (10, 'CS'), (20, 'EE');
INSERT INTO emp VALUES (1, 'Ali', 10), (2, 'Omar', 10);
DELETE FROM dept WHERE dno = 10;
DELETE FROM dept WHERE dno = 20;
SELECT COUNT(*) FROM dept;
```

A. 0
B. 2
C. 1
D. The script stops at the first DELETE, so nothing is returned

378. Foreign keys are enforced. Each INSERT runs on its own, and a failing statement changes nothing. What does the final query return? Application

```
CREATE TABLE dept (dno INT PRIMARY KEY, dname TEXT);
CREATE TABLE emp (eid INT PRIMARY KEY, name TEXT,
  dno INT REFERENCES dept(dno));
INSERT INTO dept VALUES (10, 'CS');
INSERT INTO emp VALUES (1, 'Ali', 10);
INSERT INTO emp VALUES (2, 'Sara', NULL);
INSERT INTO emp VALUES (3, 'Omar', 99);
SELECT COUNT(*) FROM emp;
```

A. 2
B. 3
C. 1
D. 0

379. Foreign keys are enforced. Each INSERT runs on its own, and a failing statement changes nothing. What does the final query return? Application

```
CREATE TABLE emp (eid INT PRIMARY KEY, name TEXT,
  super_id INT REFERENCES emp(eid));
INSERT INTO emp VALUES (1, 'Ali', NULL);
INSERT INTO emp VALUES (2, 'Sara', 3);
INSERT INTO emp VALUES (3, 'Omar', 1);
INSERT INTO emp VALUES (2, 'Sara', 3);
SELECT COUNT(*) FROM emp;
```

A. 3
B. 4
C. 2
D. 1

380. Foreign keys are enforced. How many rows remain in emp after the DELETE? Application

```
CREATE TABLE emp (eid INT PRIMARY KEY, name TEXT,
  super_id INT REFERENCES emp(eid) ON DELETE CASCADE);
INSERT INTO emp VALUES (1, 'Ali', NULL), (2, 'Sara', 1), (3, 'Omar', 1),
  (4, 'Hina', 2), (5, 'Bilal', 2), (6, 'Zara', 3), (7, 'Usman', 4);
DELETE FROM emp WHERE eid = 2;
SELECT COUNT(*) FROM emp;
```

A. 3
B. 4
C. 5
D. 6

381. Foreign keys are enforced. What does the final query return? Application

```
CREATE TABLE dept (dno INT PRIMARY KEY, dname TEXT);
CREATE TABLE emp (eid INT PRIMARY KEY, name TEXT,
  dno INT DEFAULT 0 REFERENCES dept(dno) ON DELETE SET DEFAULT);
INSERT INTO dept VALUES (0, 'Unassigned'), (10, 'CS'), (20, 'EE');
INSERT INTO emp VALUES (1, 'Ali', 10), (2, 'Sara', 10),
  (3, 'Omar', 20), (4, 'Hina', 10);
DELETE FROM dept WHERE dno = 10;
SELECT dno, COUNT(*) FROM emp GROUP BY dno ORDER BY dno;
```

A. (20, 1) only
B. (NULL, 3) and (20, 1)
C. (0, 3) and (20, 1)
D. (10, 3) and (20, 1)

382. Foreign keys are enforced. Each INSERT into enrol runs on its own, and a failing statement changes nothing. What does the final query return? Application

```
CREATE TABLE section (course TEXT, sec_no INT,
  PRIMARY KEY (course, sec_no));
CREATE TABLE enrol (sid INT, course TEXT, sec_no INT,
  PRIMARY KEY (sid, course),
  FOREIGN KEY (course, sec_no) REFERENCES section(course, sec_no));
INSERT INTO section VALUES ('CS101', 1), ('CS101', 2), ('CS102', 2);
INSERT INTO enrol VALUES (1, 'CS101', 1);
INSERT INTO enrol VALUES (2, 'CS102', 1);
INSERT INTO enrol VALUES (3, 'CS101', 2);
INSERT INTO enrol VALUES (4, 'CS103', 2);
SELECT COUNT(*) FROM enrol;
```

A. 3
B. 2
C. 1
D. 4

383. emp.dno is meant to reference dept.dno, but no FOREIGN KEY was declared. Which emp row breaks referential integrity? Application

```
CREATE TABLE dept (dno INT PRIMARY KEY, dname TEXT);
CREATE TABLE emp (eid INT PRIMARY KEY, name TEXT, dno INT);
INSERT INTO dept VALUES (10, 'CS'), (20, 'EE'), (30, 'ME');
INSERT INTO emp VALUES (1, 'Ali', 10), (2, 'Sara', NULL),
  (3, 'Omar', 40), (4, 'Hina', 30);
```

A. Sara (eid 2)
B. Omar (eid 3)
C. Hina (eid 4)
D. Ali (eid 1)

384. enrol should have the primary key (sid, cid), but the constraint was never declared. What is the smallest number of rows that must be removed before the key can be added? Application

```
CREATE TABLE enrol (sid INT, cid TEXT, grade TEXT);
INSERT INTO enrol VALUES
  (1, 'C1', 'A'), (1, 'C2', 'B'), (2, 'C1', 'A'),
  (1, 'C1', 'C'), (NULL, 'C3', 'B'), (2, 'C2', 'A');
```

A. 1
B. 2
C. 3
D. 0

385. STUDENT holds 120 tuples. Five INSERTs are attempted, and one is rejected for a duplicate key. Then 3 tuples are deleted. What is the cardinality of STUDENT now? Application

A. 122
B. 117
C. 125
D. 121

386. What is the largest number of candidate keys a relation with 4 attributes can have? Analysis

A. 6
B. 4
C. 15
D. 16

387. In a STUDENT instance, {roll} has unique values, and so does {roll, name}. A student concludes that {roll, name} is a candidate key. Which assessment is correct? Analysis

A. Wrong: if {roll} is a key, {roll, name} is not minimal. And an instance can never prove a key anyway
B. Correct, because its values are unique in the instance
C. Correct, because composite keys are preferred
D. Wrong, because names may repeat in the future

388. ORDERS.cust_id references CUSTOMER. Orders must never be lost, and every order must always show the customer who placed it. Which ON DELETE policy fits? Analysis

A. CASCADE
B. SET NULL
C. RESTRICT (or NO ACTION): refuse to delete a customer who still has orders
D. SET DEFAULT

389. DEPENDENT(essn, dep_name, ...) has primary key (essn, dep_name), and essn references EMPLOYEE. A dependent means nothing without its employee. Which ON DELETE action fits essn? Analysis

A. SET NULL
B. RESTRICT
C. CASCADE
D. SET DEFAULT

390. ENROL(student_id, course_id, grade) has the primary key (student_id, course_id). student_id also references STUDENT with ON DELETE SET NULL. Under the relational model, what is wrong with this design? Analysis

    A. SET NULL would put NULL into a primary key attribute, so it conflicts with entity integrity. CASCADE or RESTRICT fits
    B. Nothing; SET NULL suits every foreign key
    C. A foreign key may never be part of a primary key
    D. ON DELETE actions are allowed only on single-attribute keys

391. Foreign keys are enforced with SQL's default matching (MATCH SIMPLE). Each INSERT runs on its own, and a failing statement changes nothing. What does the final query return, and why? Analysis

```
CREATE TABLE section (course TEXT, sec_no INT,
  PRIMARY KEY (course, sec_no));
CREATE TABLE enrol (sid INT PRIMARY KEY, course TEXT, sec_no INT,
  FOREIGN KEY (course, sec_no) REFERENCES section(course, sec_no));
INSERT INTO section VALUES ('CS101', 1);
INSERT INTO enrol VALUES (1, 'CS101', 1);
INSERT INTO enrol VALUES (2, 'CS999', NULL);
INSERT INTO enrol VALUES (3, 'CS101', 3);
SELECT COUNT(*) FROM enrol;
```

    A. 2: the row with sec_no NULL is not checked, and (CS101, 3) is rejected
    B. 1: both the NULL row and (CS101, 3) are rejected
    C. 3: composite foreign keys are not checked at all
    D. 2: (CS999, NULL) is rejected, and (CS101, 3) is accepted because CS101 exists

392. dept and emp reference each other, and both foreign keys are DEFERRABLE INITIALLY DEFERRED. What happens? Analysis

```
CREATE TABLE dept (dno INT PRIMARY KEY,
  mgr INT NOT NULL REFERENCES emp(eid) DEFERRABLE INITIALLY DEFERRED);
CREATE TABLE emp (eid INT PRIMARY KEY,
  dno INT NOT NULL REFERENCES dept(dno) DEFERRABLE INITIALLY DEFERRED);
BEGIN;
INSERT INTO dept VALUES (10, 1);
INSERT INTO emp VALUES (1, 10);
COMMIT;
SELECT COUNT(*) FROM dept;
SELECT COUNT(*) FROM emp;
```

    A. The first INSERT fails, because employee 1 does not exist yet
    B. The second INSERT fails, because the cycle is never allowed
    C. COMMIT fails, so both tables end empty
    D. Every statement succeeds, and each table ends with 1 row

393. R(A, B, C, D) has the functional dependencies AB → CD and D → A. A student says {A, B} is the only candidate key because nothing determines B. Which assessment is correct? Analysis

    A. Wrong: {B, D} is also a candidate key, since D → A and then AB → CD
    B. Correct: {A, B} is the only candidate key
    C. Wrong: {B} alone is a candidate key
    D. Wrong: {D} is a candidate key

394. K = {A, B, C} is a superkey of R. Which conclusion must be true? Analysis

    A. {A, B} is also a superkey
    B. K is a candidate key
    C. At least one single attribute of K is a key
    D. Every superset of K within R is also a superkey

395. R(A, B, C) has the functional dependencies A → B, B → C and C → A. A student counts 3 superkeys, one for each candidate key. What is the correct count? Analysis

    A. 3
    B. 6
    C. 7
    D. 8

396. EMP(eid PRIMARY KEY, name, dno REFERENCES DEPT). Department 40 does not exist. Which constraints does INSERT INTO EMP VALUES (NULL, 'Ali', 40) violate? Analysis

    A. Only entity integrity
    B. Only referential integrity
    C. Entity integrity and referential integrity
    D. The key constraint and the domain constraint

397. PERSON(pid PRIMARY KEY, cnic UNIQUE, name) has cnic declared UNIQUE but nullable. In SQLite and PostgreSQL, several rows may have cnic NULL. Is cnic a candidate key in the formal sense? Analysis

    A. Yes, UNIQUE alone makes any column a full candidate key
    B. No, because a relation may have only one key
    C. Not strictly. Rows with NULL cnic are not identified by it, so declare it NOT NULL UNIQUE for it to act as an alternate key
    D. Yes, because NULLs count as equal, so at most one NULL is allowed

398. Foreign keys are enforced, and assign.eid has no ON DELETE action. What does the final query return? Analysis

```
CREATE TABLE dept (dno INT PRIMARY KEY);
CREATE TABLE emp (eid INT PRIMARY KEY,
  dno INT REFERENCES dept(dno) ON DELETE CASCADE);
CREATE TABLE assign (eid INT REFERENCES emp(eid), pno INT,
  PRIMARY KEY (eid, pno));
INSERT INTO dept VALUES (10), (20);
INSERT INTO emp VALUES (1, 10), (2, 10), (3, 20);
INSERT INTO assign VALUES (1, 500);
DELETE FROM dept WHERE dno = 10;
SELECT (SELECT COUNT(*) FROM dept), (SELECT COUNT(*) FROM emp),
       (SELECT COUNT(*) FROM assign);
```

    A. (1, 1, 0): the cascade removes everything linked to department 10
    B. (1, 1, 1): employees 1 and 2 are removed, and assign keeps its row
    C. (2, 1, 1): only the employees are removed
    D. (2, 3, 1): the DELETE fails, so nothing changes

399. R(A, B, C, D) has the functional dependencies A → B and B → C. Which attributes are non-prime? Analysis

    A. D only
    B. A and D
    C. B and C
    D. C only

400. EMP(eid PRIMARY KEY, name, dno) has dno as a foreign key to DEPT. A student says dno must be unique in EMP because it is a key. Which assessment is correct? Analysis

    A. Correct: every key attribute must be unique
    B. Correct: a foreign key must also be a candidate key of EMP
    C. Wrong: a foreign key need not be unique, and many employees can share the same dno
    D. Wrong only because dno must be NULL

Relational Algebra & Calculus MCQs for the NSCT

401. What does the selection operator σ do in relational algebra? Understanding
    A. Renames a relation or its attributes
    B. Returns chosen attributes of every tuple
    C. Returns the tuples that satisfy a condition, keeping all attributes
    D. Combines every tuple of one relation with every tuple of another

402. In relational algebra, what does π_dept(EMP) return? Understanding
    A. The distinct dept values of EMP, because the result is a set
    B. The tuples of EMP whose dept is not NULL
    C. The number of distinct departments
    D. Every dept value, including duplicates, in table order

403. Two relations must be union-compatible before R ∪ S, R ∩ S or R − S can be applied. What does that require? Understanding
    A. A common primary key attribute
    B. At least one tuple in common
    C. The same number of tuples
    D. The same degree, with compatible domains for corresponding attributes

404. Which relational algebra operator is the natural choice for queries that say 'for all' or 'every', such as students who took every core course? Understanding
    A. Union (∪)
    B. Division (÷)
    C. Cartesian product (×)
    D. Selection (σ)

405. How does the natural join R ⋈ S decide which tuples to combine? Understanding
    A. It keeps tuples of R that have no match in S
    B. It equates all attributes with the same name in R and S and keeps one copy of each
    C. It matches on the primary key of R only
    D. It pairs every tuple of R with every tuple of S

406. What does the rename operator ρ allow you to do? Understanding
    A. Replace NULL values with defaults
    B. Give a new name to a relation or its attributes, for example to join a relation with itself
    C. Remove duplicate tuples
    D. Sort the tuples of a relation

407. How is the theta join R ⋈_θ S defined in terms of basic operators? Understanding
    A. σ_θ(R × S)
    B. π_θ(R ∪ S)
    C. σ_θ(R) ∪ σ_θ(S)
    D. R × σ_θ(S)

408. What does the tuple relational calculus expression {t | EMP(t) ∧ t.salary > 50} return? Understanding
    A. All EMP tuples with salary above 50
    B. EMP tuples with salary 50 or less
    C. Only the salary attribute of EMP
    D. The number of employees earning above 50

409. Which set of operators is complete, meaning every other relational algebra operator can be expressed using only these? Understanding

A. {σ, π, ⋈}
B. {∪, ∩, −}
C. {σ, π, ∪, ρ, −, ×}
D. {σ, π, ∩, ÷}

410. Which expression always equals R ∩ S for union-compatible R and S? Understanding

A. R ∪ (R − S)
B. R − (R − S)
C. S − (R − S)
D. (R ∪ S) − R

411. Which statement about cascading selections is correct? Understanding

A. σ_c1(σ_c2(R)) always returns more tuples than σ_c1(R)
B. Selections can be swapped only when c1 and c2 use the same attribute
C. σ_c1(σ_c2(R)) = σ_c2(σ_c1(R)) = σ_c1 ∧ c2(R)
D. σ_c1(σ_c2(R)) = σ_c1 ∨ c2(R)

412. What does a left outer join R ⟕ S return? Understanding

A. Only the tuples of R that have a match in S
B. All matching pairs, plus every unmatched tuple of R padded with NULLs for S's attributes
C. Every tuple of S, with NULLs for unmatched R attributes
D. The Cartesian product of R and S

413. Why is {t | ¬STUDENT(t)} called an unsafe expression? Understanding

A. It uses a negation, and negation is never allowed in relational calculus
B. It describes all tuples that are not students, which is an infinite set
C. It needs a quantifier that tuple calculus does not have
D. It returns an empty result for every database

414. What is the main difference between tuple relational calculus (TRC) and domain relational calculus (DRC)? Understanding

A. TRC variables range over tuples of relations; DRC variables range over attribute values
B. TRC can express division, but DRC cannot
C. DRC is the basis of SQL, while TRC is the basis of QBE
D. TRC is procedural, while DRC is non-procedural

415. What does it mean for a query language to be relationally complete? Understanding

A. It can express any computable function over a database
B. It supports aggregates, sorting and recursion
C. It can express every query that relational algebra (or safe relational calculus) can express
D. It guarantees that every query runs in polynomial time

416. R(A, B) and S(C, D) share no attribute names. What does the natural join R ⋈ S produce? Understanding

A. An empty relation
B. R unchanged
C. An error, because there is nothing to join on
D. The Cartesian product R × S

417. R has 4 tuples and degree 3. S has 5 tuples and degree 2. What are the cardinality and degree of R × S? Application

A. 20 tuples, degree 5
B. 9 tuples, degree 6
C. 9 tuples, degree 5
D. 20 tuples, degree 6

418. How many tuples does σ_salary>50(EMP) return? Application

```
EMP
eid | name  | dept | salary
1   | Ali   | CS   | 60
2   | Sara  | EE   | 45
3   | Omar  | CS   | 50
4   | Hina  | ME   | 70
5   | Bilal | EE   | 55
6   | Zara  | CS   | 40
7   | Usman | ME   | 65
```

A. 4
B. 7
C. 3
D. 5

419. How many tuples does π_dept(EMP) return? Application

```
EMP
eid | name  | dept | salary
1   | Ali   | CS   | 60
2   | Sara  | EE   | 45
3   | Omar  | CS   | 50
4   | Hina  | ME   | 70
5   | Bilal | EE   | 55
6   | Zara  | CS   | 40
7   | Usman | ME   | 65
```

A. 1
B. 3
C. 4
D. 7

420. R(A) = {1, 2, 3, 4, 5} and S(A) = {4, 5, 6}. How many tuples are in R ∪ S? Application

A. 8
B. 5
C. 2
D. 6

421. Using the same R(A) = {1, 2, 3, 4, 5} and S(A) = {4, 5, 6}, what is R − S? Application

A. {6}
B. {1, 2, 3, 6}
C. {1, 2, 3}
D. {4, 5}

422. How many tuples does the natural join R ⋈ S contain? Application

```
R           S
A | B       B | C
1 | x       x | 10
2 | y       x | 20
3 | x       y | 30
4 | z       w | 40
```

A. 16
B. 4
C. 3
D. 5

423. Using the same R and S, how many tuples does the left outer join R ⟕ S contain? Application

```
R           S
A | B       B | C
1 | x       x | 10
2 | y       x | 20
3 | x       y | 30
4 | z       w | 40
```

A. 5
B. 7
C. 6
D. 4

424. R(a) = {1, 2, 3} and S(c) = {1, 2, 3, 4}. How many tuples does the theta join R ⋈_a<c S return? Application

A. 12
B. 3
C. 9
D. 6

425. Which expression returns the names of CS employees who earn more than 45? Application

```
EMP
eid | name  | dept | salary
1   | Ali   | CS   | 60
2   | Sara  | EE   | 45
3   | Omar  | CS   | 50
4   | Hina  | ME   | 70
5   | Bilal | EE   | 55
6   | Zara  | CS   | 40
7   | Usman | ME   | 65
```

A. π_name(σ_dept='CS' ∧ salary>45(EMP))
B. σ_dept='CS' ∧ salary>45(π_name(EMP))
C. π_name(σ_dept='CS' ∨ salary>45(EMP))
D. π_name,dept(σ_salary>45(EMP))

426. R and S are union-compatible, with |R| = 5 and |S| = 3. What are the minimum and maximum possible sizes of R ∪ S? Application

A. Minimum 3, maximum 8
B. Minimum 8, maximum 8
C. Minimum 0, maximum 15
D. Minimum 5, maximum 8

427. R and S are union-compatible, with |R| = 5 and |S| = 3. What are the smallest and largest possible sizes of the difference R − S? Application

A. Minimum 2, maximum 5
B. Minimum 2, maximum 2
C. Minimum 0, maximum 2
D. Minimum 0, maximum 5

428. What is ENROL ÷ CORE? Application

```
ENROL               CORE
sid | course        course
S1  | DB            DB
S1  | OS            OS
S1  | CN            CN
S2  | DB
S2  | OS
S3  | DB
S3  | OS
S3  | CN
S3  | AI
S4  | CN
```

A. {S3}
B. {S1, S2, S3}
C. {S1, S3}
D. {S1}

429. ORDERS(oid, cid, amount) has 100 tuples. cid is a NOT NULL foreign key to CUSTOMER(cid), which has 20 tuples, and referential integrity holds. How many tuples does ORDERS ⋈ CUSTOMER (natural join on cid) contain? Application

A. 2000
B. 20
C. Between 20 and 100
D. 100

430. Which relational algebra expression is equivalent to {t.name | EMP(t) ∧ ∃d (DEPT(d) ∧ d.dnum = t.dno ∧ d.dname = 'Research')}? Application

A. σ_dname='Research'(π_name(EMP ⋈_dno=dnum DEPT))
B. π_name(EMP) − π_name(σ_dname='Research'(DEPT))
C. π_name(σ_dname='Research'(EMP ⋈_dno=dnum DEPT))
D. π_name(σ_dname='Research'(EMP × DEPT))

431. R(A, B) and S(B). Which expression equals R ÷ S using only basic operators? Analysis

A. π_A(R) − π_A(R − (π_A(R) × S))
B. π_A(π_A(R) × S)
C. π_A((π_A(R) × S) − R)
D. π_A(R) − π_A((π_A(R) × S) − R)

432. ENROL is as shown, but CORE is now empty. Using the standard definition, what is ENROL ÷ CORE? Analysis

```
ENROL
sid | course
S1  | DB
S1  | OS
S1  | CN
S2  | DB
S2  | OS
S3  | DB
S3  | OS
S3  | CN
S3  | AI
S4  | CN

CORE(course) has no tuples.
```

A. {S1, S2, S3, S4}
B. The division is undefined
C. {S3}
D. The empty relation

433. Which expression is invalid? Analysis

A. π_name(π_name,salary(EMP))
B. σ_salary>50(σ_dept='CS'(EMP))
C. σ_salary>50(π_name(EMP))
D. π_name(σ_salary>50(EMP))

434. CUSTOMER(cid, name, city) holds the customer's home city; ORDERS(oid, cid, city, amount) holds the delivery city. How many tuples does CUSTOMER ⋈ ORDERS (natural join) return? Analysis

```
CUSTOMER                ORDERS
cid | name | city       oid | cid | city      | amount
1   | Ali  | Lahore     101 | 1   | Lahore    | 500
2   | Sara | Karachi    102 | 1   | Islamabad | 300
3   | Omar | Lahore     103 | 2   | Karachi   | 200
                         104 | 3   | Multan    | 700
```

A. 4
B. 2
C. 3
D. 12

435. For the EMP table shown, compare the relational algebra expression π_dept(EMP) with the SQL query SELECT dept FROM EMP. Analysis

```
EMP
eid | name  | dept | salary
1   | Ali   | CS   | 60
2   | Sara  | EE   | 45
3   | Omar  | CS   | 50
4   | Hina  | ME   | 70
5   | Bilal | EE   | 55
6   | Zara  | CS   | 40
7   | Usman | ME   | 65
```

A. π returns 3 tuples; SQL returns 7 rows
B. Both return 3
C. Both return 7
D. π returns 7 tuples; SQL returns 3 rows

436. A query optimiser may rewrite expressions only when they are always equivalent. Which rewrite is NOT always valid? Analysis

A. σ_c(R ∪ S) = σ_c(R) ∪ σ_c(S)
B. π_A(R ∪ S) = π_A(R) ∪ π_A(S)
C. σ_c(R − S) = σ_c(R) − S
D. π_A(R − S) = π_A(R) − π_A(S)

437. EMP(eid, name, salary, sup) stores each employee's supervisor id in sup. Which expression finds employees who earn more than their supervisor? Analysis

A. π_E.name(σ_E.salary>S.salary(ρ_E(EMP) ⋈_E.eid=S.sup ρ_S(EMP)))
B. π_name(σ_salary>salary(EMP ⋈ EMP))
C. π_name(σ_salary>sup(EMP))
D. π_E.name(σ_E.salary>S.salary(ρ_E(EMP) ⋈_E.sup=S.eid ρ_S(EMP)))

438. STUDENT(sid) = {S1, S2, S3, S4, S5}. ENROL(sid, course) = {(S1, CS101), (S1, MA101), (S2, MA101), (S3, CS101), (S4, PH101)}. For 'students who did not take CS101', E1 = π_sid(STUDENT) − π_sid(σ_course='CS101'(ENROL)) and E2 = π_sid(σ_course≠'CS101'(ENROL)). What do they return? Analysis

A. Both return {S2, S4, S5}
B. E1 = {S2, S4, S5}; E2 = {S1, S2, S4}
C. E1 = {S2, S4}; E2 = {S2, S4, S5}
D. E1 = {S1, S2, S4}; E2 = {S2, S4, S5}

439. How many tuples does the full outer join R ⟗ S return? Analysis

```
R           S
A | B       B | C
1 | x       x | 10
2 | y       x | 20
3 | x       y | 30
4 | z       w | 40
```

A. 6
B. 7
C. 5
D. 16

440. Which tuple relational calculus expression is safe? Analysis

A. {t | t.cgpa > 3.5}
B. {t | ¬ENROL(t) ∨ t.grade = 'A'}
C. {t | STUDENT(t) ∧ ¬∃e (ENROL(e) ∧ e.sid = t.sid)}
D. {t | ¬STUDENT(t)}

441. Why can relational algebra operations be nested, as in π_name(σ_salary>50(EMP))? Understanding

A. σ and π are commutative, so they can be applied in any order
B. Every operation takes relations as input and returns a relation, so its result can feed another operation
C. The DBMS always evaluates nested expressions from left to right
D. Nested expressions are converted into SQL subqueries before they run

442. Which group lists only unary relational algebra operators, each taking a single relation as input? Understanding

A. σ, ×, ρ
B. σ, π, ρ
C. π, ∪, ÷
D. ⋈, −, ρ

443. Which statement correctly contrasts relational algebra with relational calculus? Understanding

A. Calculus describes a sequence of operations that builds the answer; algebra describes the properties the answer must have
B. Algebra describes a sequence of operations that builds the answer; calculus describes the properties the answer must have
C. Safe relational calculus can express queries that relational algebra cannot
D. Algebra works only on single tables, while calculus can combine several tables

444. An equijoin EMP ⋈_EMP.dept=DEPT.dept DEPT and a natural join EMP ⋈ DEPT match exactly the same pairs of tuples. How do their results differ? Understanding

A. The natural join also keeps EMP tuples that have no matching DEPT tuple
B. The equijoin keeps both dept attributes; the natural join keeps only one copy
C. The equijoin removes duplicate tuples, but the natural join does not
D. The natural join compares dept values with ≥ instead of =

445. What does the semijoin R ⋉ S return? Understanding

A. The tuples of R that match no tuple of S
B. All combinations of R and S tuples that agree on their common attributes
C. The tuples of R that match at least one tuple of S, with only R's attributes
D. The tuples of S that match at least one tuple of R, with all attributes of both

446. Ignoring the order of attributes, which binary operator is NOT commutative? Understanding

A. Union (∪)
B. Intersection (∩)
C. Difference (−)
D. Natural join (⋈)

447. Which part of an SQL SELECT statement plays the role of the selection operator σ? Understanding

A. The SELECT list
B. The WHERE clause
C. The FROM clause
D. The ORDER BY clause

448. In Elmasri-Navathe notation, what does the statement IT_EMP ← σ_dept='IT'(EMP) do? Understanding

A. Stores the result as a named temporary relation IT_EMP for use in later steps
B. Deletes every non-IT tuple from EMP
C. Renames the dept attribute of EMP to IT_EMP
D. Creates a permanent table IT_EMP in the database schema

449. Compared with relation R, what are the degree and cardinality of σ_c(R)? Understanding

A. Fewer attributes, and the same number of tuples as R
B. The same degree, and at most as many tuples as R
C. The same degree, and at least as many tuples as R
D. Fewer attributes, and fewer tuples than R

450. A theta join R ⋈_θ S uses a condition θ that is true for every pair of tuples. What is the result? Understanding

A. R × S
B. R ∩ S
C. R ⋈ S (natural join)
D. R ∪ S

451. R and S have exactly the same attributes, A and B. What does the natural join R ⋈ S return? Understanding

A. R ∪ S
B. R × S
C. R − S
D. R ∩ S

452. The rule π_L1(π_L2(R)) = π_L1(R) lets an optimiser drop the inner projection. When is the left-hand expression valid in the first place? Understanding

A. When L2 is a subset of L1
B. When L1 and L2 share no attributes
C. Only when L1 and L2 are identical
D. When L1 is a subset of L2

453. Why does a heuristic query optimiser move selection operations as far down the query tree as possible? Understanding

A. Selections must run after projections, or the attributes they test are lost
B. Pushing selections down removes duplicate tuples from the final result
C. A selection at the bottom of the tree can use the rename operator automatically
D. Filtering early shrinks the intermediate relations that later joins and products must process

454. Which expression returns the tuples of R that have NO matching tuple in S (an anti-join)? Understanding

A. R ⋉ S
B. R − S
C. S − (R ⋉ S)
D. R − (R ⋉ S)

455. In the TRC expression {t | EMP(t) ∧ ∃d (DEPT(d) ∧ d.dept = t.dept)}, which tuple variables are free? Understanding

A. Only d
B. Both t and d
C. Only t
D. Neither, because both appear inside a condition

456. Which formula is equivalent to ∀x (P(x)) in relational calculus? Understanding

A. ∃x (¬P(x))
B. ¬∃x (P(x))
C. ¬∃x (¬P(x))
D. ¬∀x (¬P(x))

457. Which expression always equals the full outer join R ⟗ S? Understanding

A. (R ⟕ S) ∩ (R ⟖ S)
B. (R ⋈ S) ∪ (R × S)
C. R ⟕ (S ⟕ R)
D. (R ⟕ S) ∪ (R ⟖ S)

458. Query-By-Example (QBE), where the user fills example values into table skeletons, is based on which formalism? Understanding

A. Tuple relational calculus
B. Relational algebra
C. Entity-relationship algebra
D. Domain relational calculus

459. SQL has no division operator. Which construct is the standard way to write a 'for all' (division) query in SQL? Understanding

A. Two nested NOT EXISTS subqueries
B. A single IN subquery
C. A LEFT OUTER JOIN with IS NULL
D. UNION ALL of two SELECT statements

460. Which operation belongs to the extended relational algebra rather than to Codd's original set of operators? Understanding

A. Grouping with aggregate functions ($\mathscr{F}$ or γ)
B. Division (÷)
C. Natural join (⋈)
D. Intersection (∩)

461. PARENT(p, c) stores parent-child pairs. Which query cannot be written as a single expression of basic relational algebra, whatever the data? Understanding

A. All grandparents of a person
B. People who are parents but not children
C. All ancestors of a person, at any number of generations
D. People who have at least two children

462. For which operator can adding a tuple to one of its inputs make a tuple disappear from its output? Understanding

A. Selection σ_c(R)
B. Natural join R ⋈ S
C. Difference R − S, when the tuple is added to S
D. Union R ∪ S

463. Which law holds for relational algebra on sets but fails for SQL tables, which are multisets (bags)? Understanding

A. R ∪ S = S ∪ R
B. σ_c(σ_d(R)) = σ_d(σ_c(R))
C. R × S has |R| · |S| tuples
D. R ∪ R = R

464. Elmasri-Navathe's complete set is {σ, π, ∪, ρ, −, ×}. Why must − be in it while ∩ is left out? Understanding

A. ∩ is not a valid relational algebra operator
B. ∩ can be written as R − (R − S), but no expression built only from σ, π, ∪, ρ and × can express difference
C. − is unary while ∩ is binary
D. ∩ needs union-compatible relations but − does not

465. How many tuples does σ_dept='IT' ∨ salary<45(EMP) return? Application

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44
```

A. 5
B. 3
C. 2
D. 6

466. What does π_salary(σ_dept='IT'(EMP)) return? Application

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44
```

A. {75, 48}: 2 tuples
B. {75, 48, 75}: 3 tuples
C. {75}: 1 tuple
D. {48}: 1 tuple

467. What does π_dept(DEPT) − π_dept(EMP) return? Application

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44

DEPT
dept | floor
HR   | 1
IT   | 3
FIN  | 2
MKT  | 4
```

A. {OPS}
B. {MKT}
C. {MKT, OPS}
D. An empty relation

468. What does the domain relational calculus query {<n> | ∃e ∃d ∃s (EMP(e, n, d, s) ∧ s > 60)} return? Application

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44
```

A. Danish, Hamza, Kamran
B. Danish, Kamran
C. Danish, Hamza, Kamran, Laiba
D. Ayesha, Danish, Hamza, Kamran, Laiba

469. How many tuples does π_name(σ_dept='HR'(EMP)) ∪ π_name(σ_salary>70(EMP)) return? Application

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44
```

A. 2
B. 4
C. 3
D. 5

470. R and S are union-compatible, with |R| = 7 and |S| = 4. What are the smallest and largest possible sizes of R ∩ S? Application

A. Minimum 4, maximum 7
B. Minimum 0, maximum 7
C. Minimum 3, maximum 11
D. Minimum 0, maximum 4

471. Which expression returns the names of employees who work on floor 3? Application

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44

DEPT
dept | floor
HR   | 1
IT   | 3
FIN  | 2
MKT  | 4
```

A. π_name(σ_floor=3(EMP))
B. σ_floor=3(π_name(EMP ⋈ DEPT))
C. π_name(σ_floor=3(EMP ⋈ DEPT))
D. π_name(EMP) − π_name(σ_floor=3(EMP ⋈ DEPT))

472. How many tuples does π_dept,salary(EMP) return? Application

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44
```

A. 8
B. 4
C. 6
D. 7

473. How many tuples does the natural join EMP ⋈ DEPT return? Application

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44

DEPT
dept | floor
HR   | 1
IT   | 3
FIN  | 2
MKT  | 4
```

A. 7
B. 8
C. 4
D. 32

474. How many tuples does the right outer join EMP ⟖ DEPT return? Application

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44

DEPT
dept | floor
HR   | 1
IT   | 3
FIN  | 2
MKT  | 4
```

A. 7
B. 9
C. 4
D. 8

475. What does the anti-join EMP ▷ DEPT, written EMP − (EMP ⋉ DEPT), return? Application

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44

DEPT
dept | floor
HR   | 1
IT   | 3
FIN  | 2
MKT  | 4
```

A. One tuple: the MKT department
B. One tuple: Nabeel's
C. Seven tuples: every employee except Nabeel
D. An empty relation

476. How many tuples does σ_E1.dept=E2.dept ∧ E1.eid<E2.eid(ρ_E1(EMP) × ρ_E2(EMP)) return? Application

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44
```

A. 5
B. 10
C. 18
D. 6

477. R(a) = {2, 4, 6, 8} and S(b) = {3, 5}. How many tuples does R ⋈_a>b ∧ a<2b S return? Application

A. 3
B. 5
C. 2
D. 8

478. How many names does {t.name | EMP(t) ∧ ∃d (DEPT(d) ∧ d.dept = t.dept ∧ d.floor > 1)} return? Application

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44

DEPT
dept | floor
HR   | 1
IT   | 3
FIN  | 2
MKT  | 4
```

A. 5
B. 3
C. 7
D. 2

479. What does the grouping expression dept $\mathscr{F}$COUNT eid, AVERAGE salary (EMP) return (written dept γ COUNT(eid), AVG(salary) in some textbooks)? Application

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44
```

A. 4 tuples; the IT tuple is (IT, 3, 75)
B. 8 tuples, one per employee
C. 1 tuple: (8, 56.5)
D. 4 tuples; the IT tuple is (IT, 3, 66)

480. R(A, B) has 10 tuples and S(B, C) has 6 tuples. B is the primary key of S, but R.B is not a foreign key. What are the possible sizes of R ⋈ S? Application

A. Between 0 and 10
B. Between 6 and 10
C. Exactly 10
D. Between 0 and 60

481. What does π_name(EMP ⋈ (π_eid(EMP) − π_E1.eid(σ_E1.salary<E2.salary(ρ_E1(EMP) × ρ_E2(EMP))))) return? Application

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44
```

A. Danish
B. Kamran
C. An empty relation
D. Danish, Kamran

482. What does WORKS ÷ π_pno(σ_dept='IT'(PROJ)) return? Application

```
WORKS          PROJ
eid | pno      pno | dept
1   | P1       P1  | IT
1   | P2       P2  | HR
2   | P1       P3  | IT
2   | P2       P4  | FIN
2   | P3
3   | P3
4   | P1
4   | P3
6   | P2
```

A. {2}
B. {2, 3, 4}
C. {2, 4}
D. {1, 2, 4, 6}

483. R(A, B) has 12 tuples and S(B) has 3 tuples. What is the largest possible number of tuples in R ÷ S? Application

A. 3
B. 9
C. 12
D. 4

484. Which employees does π_eid(EMP) − π_E1.eid(σ_E1.salary≤E2.salary ∧ E2.dept='HR'(ρ_E1(EMP) × ρ_E2(EMP))) return? Application

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44
```

A. Danish, Kamran
B. Ayesha, Danish, Hamza, Kamran, Laiba
C. Hamza, Laiba
D. Danish, Hamza, Kamran, Laiba

485. TAKES(sid, course, sem) records each course a student took and the semester. What does TAKES ÷ CORE return? Application

```
TAKES                  CORE
sid | course | sem     course
S1  | DB     | 1       DB
S1  | OS     | 1       OS
S1  | DB     | 2
S2  | DB     | 1
S2  | OS     | 2
S3  | DB     | 2
S3  | OS     | 2
S3  | AI     | 1
S4  | OS     | 1
```

A. {S1, S2, S3}
B. {(S1, 1), (S1, 2), (S2, 1), (S3, 2)}
C. {(S1, 1), (S3, 2)}
D. {(S1, 1), (S2, 1), (S3, 2)}

486. Assume R and S are union-compatible. Which rewrite is always valid? Analysis

A. π_A(R ∩ S) = π_A(R) ∩ π_A(S)
B. σ_c(R − S) = σ_c(R) − S
C. (R − S) − T = R − (S − T)
D. R − S = S − R

487. To find the highest-paid employees, a student writes π_eid(EMP) − π_E1.eid(σ_E1.salary≤E2.salary(ρ_E1(EMP) × ρ_E2(EMP))). What does it return on this table? Analysis

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44
```

A. {2, 6}
B. {2}
C. An empty relation
D. {5}

488. Assume no NULLs. Which expression is NOT equivalent to σ_c1 ∨ c2(R)? Analysis

A. σ_c1(R) ∪ σ_c2(R)
B. R − σ_¬c1 ∧ ¬c2(R)
C. σ_c1(R) ∩ σ_c2(R)
D. σ_c2 ∨ c1(R)

489. For 'employees who work on both P1 and P2', a student writes π_eid(σ_pno='P1' ∧ pno='P2'(WORKS)). What does it return, and what is the correct result? Analysis

```
WORKS          PROJ
eid | pno      pno | dept
1   | P1       P1  | IT
1   | P2       P2  | HR
2   | P1       P3  | IT
2   | P2       P4  | FIN
2   | P3
3   | P3
4   | P1
4   | P3
6   | P2
```

A. It returns {1, 2}, which is correct
B. It returns {1, 2, 4, 6}; the correct result is {1, 2}
C. It returns an empty relation; the correct result is {1, 2}
D. It returns an empty relation; the correct result is {1, 2, 4, 6}

490. For 'employees who work on at most one project', a student writes π_eid(WORKS) − π_W1.eid(σ_W1.eid=W2.eid ∧ W1.pno≠W2.pno(ρ_W1(WORKS) × ρ_W2(WORKS))). Assess the result on this data. Analysis

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44

WORKS          PROJ
eid | pno      pno | dept
1   | P1       P1  | IT
1   | P2       P2  | HR
2   | P1       P3  | IT
2   | P2       P4  | FIN
2   | P3
3   | P3
4   | P1
4   | P3
6   | P2
```

A. It returns {3, 5, 6, 7, 8}, which is fully correct
B. It returns {1, 2, 4}, the employees with two or more projects
C. It returns {3, 6}, but misses employees 5, 7 and 8, who work on no project
D. It returns an empty relation, because every employee is paired with itself

491. Compare E1 = σ_floor=3(EMP ⋈ DEPT) with E2 = EMP ⋈ σ_floor=3(DEPT). How many tuples does each return? Analysis

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44

DEPT
dept | floor
HR   | 1
IT   | 3
FIN  | 2
MKT  | 4
```

A. Both have 3
B. E1 has 8; E2 has 3
C. E1 has 3; E2 has 7
D. E1 has 3; E2 has 8

492. R(a, b) and S(b) are relations. Which domain relational calculus expression is unsafe? Analysis

A. {<a, b> | R(a, b) ∧ ¬S(b)}
B. {<a, b> | R(a, b) ∨ S(b)}
C. {<a> | ∃b (R(a, b) ∧ S(b))}
D. {<a> | ∃b R(a, b) ∧ ∀c (S(c) → R(a, c))}

493. R(A) = {1, 2, 3} and S(A) = {2, 3, 4}. Compare the algebra expression R ∪ S with the SQL query SELECT A FROM R UNION ALL SELECT A FROM S. Analysis

A. Both return 4
B. R ∪ S has 4 tuples; UNION ALL returns 6 rows
C. Both return 6
D. R ∪ S has 6 tuples; UNION ALL returns 4 rows

494. ENROL(sid, course) holds many students, and CORE(course) lists three courses. One CORE course, ML, appears in no ENROL tuple. What is ENROL ÷ CORE? Analysis

A. An empty relation
B. Every sid in ENROL
C. The students who took the other two core courses
D. The students who took at least one core course

495. A student renames DEPT's dept attribute to d with DEPT2 ← ρ_(d, floor)(DEPT), then computes the natural join EMP ⋈ DEPT2. How many tuples result? Analysis

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44

DEPT
dept | floor
HR   | 1
IT   | 3
FIN  | 2
MKT  | 4
```

A. 32
B. 7
C. 8
D. 4

496. EMP has a dept attribute (the employee's department) and PROJ has a dept attribute (the controlling department). How many tuples does (EMP ⋈ WORKS) ⋈ PROJ return? Analysis

```
EMP
eid | name   | dept | salary
1   | Ayesha | HR   | 52
2   | Danish | IT   | 75
3   | Fatima | IT   | 48
4   | Hamza  | FIN  | 61
5   | Iqra   | HR   | 39
6   | Kamran | IT   | 75
7   | Laiba  | FIN  | 58
8   | Nabeel | OPS  | 44

WORKS          PROJ
eid | pno      pno | dept
1   | P1       P1  | IT
1   | P2       P2  | HR
2   | P1       P3  | IT
2   | P2       P4  | FIN
2   | P3
3   | P3
4   | P1
4   | P3
6   | P2

PROJ: P1 IT, P2 HR, P3 IT, P4 FIN
```

A. 9
B. 8
C. 4
D. 5

497. Relation S is not empty. Which expression is NOT always equal to the semijoin R ⋉ S? Analysis

A. π_attrs(R)(R ⋈ S)
B. R ⋈ π_common(S), where common is the attributes R and S share
C. π_attrs(R)(R × S)
D. R − (R ▷ S), where ▷ is the anti-join

498. What does {w.eid | WORKS(w) ∧ ∃p (PROJ(p) ∧ p.dept = 'IT' ∧ p.pno = w.pno)} return? Analysis

```
WORKS          PROJ
eid | pno      pno | dept
1   | P1       P1  | IT
1   | P2       P2  | HR
2   | P1       P3  | IT
2   | P2       P4  | FIN
2   | P3
3   | P3
4   | P1
4   | P3
6   | P2
```

A. {2, 4}
B. {1, 2, 3, 4}
C. {1, 2, 3, 4, 6}
D. {2, 3, 4}

499. R(A, B) has 5 tuples and S(B, C) is empty. What are the sizes of R ⋈ S, R ⟕ S and R × S? Analysis

A. 0, 5 and 0
B. 5, 5 and 5
C. 0, 0 and 0
D. 0, 5 and 5

500. EMP has 1,000 tuples and DEPT has 50, with 5 departments on floor 3. Plan A evaluates σ_floor=3 ∧ EMP.dept=DEPT.dept(EMP × DEPT). Plan B evaluates σ_EMP.dept=DEPT.dept(EMP × σ_floor=3(DEPT)). How many tuples does each plan's Cartesian product produce? Analysis

A. Plan A 5,000; plan B 50,000
B. Plan A 50,000; plan B 5,000
C. Both 50,000
D. Plan A 1,050; plan B 1,005

Structured Query Language (SQL) MCQs for the N...

501. Which of these SQL statements belongs to the Data Definition Language (DDL)? Understanding

A. UPDATE
B. GRANT
C. ALTER TABLE
D. SAVEPOINT

502. GRANT and REVOKE belong to which category of SQL commands? Understanding

A. Data Control Language (DCL)
B. Data Definition Language (DDL)
C. Data Manipulation Language (DML)
D. Transaction Control Language (TCL)

503. Which group consists entirely of Transaction Control Language (TCL) commands? Understanding

A. CREATE, ALTER, DROP
B. GRANT, REVOKE
C. INSERT, UPDATE, DELETE
D. COMMIT, ROLLBACK, SAVEPOINT

504. Which of the following is a Data Manipulation Language (DML) command? Understanding

A. TRUNCATE
B. UPDATE
C. CREATE INDEX
D. REVOKE

505. Why can a condition such as COUNT(*) > 3 not be written in a WHERE clause? Understanding

A. WHERE may only compare a column with a constant
B. WHERE filters individual rows before groups exist, so aggregates are not yet computed; such conditions go in HAVING
C. Aggregate functions are allowed only in ORDER BY
D. WHERE is evaluated after SELECT, so the aggregate has already been discarded

506. What is the difference between COUNT(*) and COUNT(phone)? Understanding

A. COUNT(phone) counts only distinct phone values
B. COUNT(*) counts all rows; COUNT(phone) counts only rows where phone is not NULL
C. COUNT(*) skips rows that contain a NULL in any column
D. There is no difference; both always return the same number

507. In a LIKE pattern, what does the underscore character (_) match? Understanding

A. Exactly one character
B. Zero or more characters
C. One or more characters
D. Only a literal underscore

508. The condition price BETWEEN 100 AND 200 is equivalent to which of the following? Understanding

A. price >= 100 AND price <= 200
B. price > 100 AND price < 200
C. price >= 100 AND price < 200
D. price > 100 AND price <= 200

509. Which statement about a PRIMARY KEY in standard SQL is correct? Understanding

A. It may contain a single NULL value as long as the rest are unique
B. A table may have several primary keys but only one UNIQUE constraint
C. Its values must be unique and non-NULL, and a table has at most one primary key
D. It automatically references the primary key of another table

510. How does a UNIQUE constraint differ from a PRIMARY KEY in standard SQL? Understanding

A. A UNIQUE column allows duplicate values as long as it is indexed
B. A UNIQUE column may contain NULLs, and a table can have several UNIQUE constraints
C. A UNIQUE constraint cannot cover more than one column
D. A UNIQUE column is implicitly NOT NULL, but only one is allowed per table

511. orders.customer_id is declared as REFERENCES customers(id) ON DELETE CASCADE. What happens when a customer row is deleted? Understanding

A. The delete is rejected while any order still references the customer
B. customer_id in the matching orders is set to NULL
C. All orders referencing that customer are deleted as well
D. The entire orders table is truncated

512. Which constraint ensures that every value in a column satisfies a condition such as age >= 18? Understanding

A. UNIQUE
B. CHECK
C. NOT NULL
D. FOREIGN KEY

513. Which command removes both a table's rows and the table's definition from the database? Understanding

A. DELETE FROM t
B. DROP TABLE t
C. TRUNCATE TABLE t
D. ALTER TABLE t DROP COLUMN c

514. Which statement about DELETE and TRUNCATE is correct? Understanding

A. DELETE can remove selected rows using WHERE; TRUNCATE removes all rows but keeps the table
B. TRUNCATE accepts a WHERE clause, while DELETE always removes every row
C. DELETE removes the table definition along with the rows
D. TRUNCATE drops the table and all of its constraints

515. In the logical processing order of a SELECT statement, which clause is evaluated last? Understanding

A. HAVING
B. SELECT
C. ORDER BY
D. GROUP BY

516. Why does WHERE manager_id = NULL return no rows, even when some manager_id values are NULL? Understanding

A. = NULL is a syntax error, so the query returns nothing
B. NULLs are stored as empty strings, so they never equal NULL
C. WHERE skips NULL values only when the column is indexed
D. Any comparison with NULL gives UNKNOWN, which WHERE treats as not true; IS NULL is needed

517. After these statements run, what do the three COUNT expressions return? Application

```
CREATE TABLE emp (id INT, name TEXT, phone TEXT);
INSERT INTO emp VALUES
  (1, 'Ali',  '0300'), (2, 'Sara', NULL),
  (3, 'Omar', '0301'), (4, 'Hina', NULL),
  (5, 'Zain', '0300');
SELECT COUNT(*), COUNT(phone), COUNT(DISTINCT phone) FROM emp;
```

A. 5, 3, 2
B. 5, 5, 3
C. 3, 3, 2
D. 5, 3, 3

518. What value does AVG(bonus) return for this table? Application

```
CREATE TABLE staff (name TEXT, bonus INT);
INSERT INTO staff VALUES ('A', 100), ('B', 200), ('C', NULL), ('D', 300);
SELECT AVG(bonus) FROM staff;
```

A. 200
B. 150
C. NULL
D. 600

519. Which region totals survive the HAVING filter? Application

```
CREATE TABLE sales (region TEXT, amount INT);
INSERT INTO sales VALUES
  ('North', 100), ('North', 250), ('South', 300),
  ('East', 50), ('East', 60), ('East', 70), ('South', 20);
SELECT region, SUM(amount)
FROM sales
GROUP BY region
HAVING SUM(amount) > 300
ORDER BY region;
```

A. North 350 only
B. North 350, South 320
C. East 180, North 350, South 320
D. North 2, South 2

520. How many student names match the LIKE pattern? Application

```
CREATE TABLE students (name TEXT);
INSERT INTO students VALUES ('Ali'), ('A'), ('Amna'), ('Bilal'), ('Asad'), ('Sana');
SELECT COUNT(*) FROM students WHERE name LIKE 'A_%';
```

A. 4
B. 2
C. 1
D. 3

521. How many rows does the query count? Application

```
CREATE TABLE result (roll INT, marks INT);
INSERT INTO result VALUES (1, 40), (2, 50), (3, 60), (4, 70), (5, 80), (6, 90);
SELECT COUNT(*) FROM result WHERE marks BETWEEN 50 AND 80;
```

A. 2
B. 3
C. 4
D. 5

522. Which name appears in the second row of the result? Application

```
CREATE TABLE emp (name TEXT, dept TEXT, salary INT);
INSERT INTO emp VALUES
  ('Sara', 'HR', 5000), ('Ali', 'IT', 7000), ('Hina', 'HR', 6000),
  ('Omar', 'IT', 9000), ('Zain', 'Admin', 4000);
SELECT name FROM emp ORDER BY dept ASC, salary DESC;
```

A. Sara
B. Omar
C. Ali
D. Hina

523. How many distinct (student, course) pairs does the query count? Application

```
CREATE TABLE enroll (student TEXT, course TEXT);
INSERT INTO enroll VALUES
  ('S1', 'DB'), ('S1', 'OS'), ('S2', 'DB'),
  ('S1', 'DB'), ('S2', 'DB'), ('S3', 'OS');
SELECT COUNT(*) FROM (SELECT DISTINCT student, course FROM enroll) AS t;
```

A. 3
B. 2
C. 4
D. 6

524. How many cities fall outside the two listed provinces? Application

```
CREATE TABLE city (name TEXT, province TEXT);
INSERT INTO city VALUES
  ('Lahore', 'Punjab'), ('Karachi', 'Sindh'), ('Quetta', 'Balochistan'),
  ('Multan', 'Punjab'), ('Peshawar', 'KP');
SELECT COUNT(*) FROM city WHERE province NOT IN ('Punjab', 'Sindh');
```

A. 3
B. 5
C. 1
D. 2

525. Table emp(name, dept) lists one row per employee. Which query returns the departments that have more than 2 employees? Application

A. SELECT dept FROM emp GROUP BY dept HAVING COUNT(*) > 2;
B. SELECT dept FROM emp WHERE COUNT(*) > 2 GROUP BY dept;
C. SELECT dept FROM emp GROUP BY dept HAVING COUNT(*) >= 2;
D. SELECT DISTINCT dept FROM emp HAVING COUNT(*) > 2;

526. What are the values of SUM(a + b) and SUM(a) + SUM(b)? Application

```
CREATE TABLE t (a INT, b INT);
INSERT INTO t VALUES (1, 2), (3, NULL), (NULL, 4), (5, 6);
SELECT SUM(a + b), SUM(a) + SUM(b) FROM t;
```

A. 21 and 21
B. NULL and 21
C. 14 and 14
D. 14 and 21

527. Given this table, which INSERT is rejected? Application

```
CREATE TABLE acct (
  id      INT PRIMARY KEY,
  balance INT CHECK (balance >= 0)
);
```

A. INSERT INTO acct VALUES (3, -1);
B. INSERT INTO acct VALUES (1, 0);
C. INSERT INTO acct VALUES (2, NULL);
D. INSERT INTO acct VALUES (4, 999);

528. Assume foreign keys are enforced. Which INSERT into emp is rejected? Application

```
CREATE TABLE dept (id INT PRIMARY KEY, name TEXT);
CREATE TABLE emp (
  id      INT PRIMARY KEY,
  dept_id INT REFERENCES dept(id)
);
INSERT INTO dept VALUES (10, 'HR'), (20, 'IT');
```

A. INSERT INTO emp VALUES (1, 10);
B. INSERT INTO emp VALUES (2, NULL);
C. INSERT INTO emp VALUES (3, 30);
D. INSERT INTO emp VALUES (4, 20);

529. After the DELETE, what does the final COUNT(*) return? Application

```
CREATE TABLE orders (id INT, status TEXT);
INSERT INTO orders VALUES
  (1, 'pending'), (2, 'shipped'), (3, 'pending'),
  (4, 'cancelled'), (5, 'shipped'), (6, 'pending');
DELETE FROM orders WHERE status = 'pending' OR status = 'cancelled';
SELECT COUNT(*) FROM orders;
```

A. 4
B. 3
C. 0
D. 2

530. What is the result row for the IT department? Application

```
CREATE TABLE emp (name TEXT, dept TEXT, salary INT);
INSERT INTO emp VALUES
  ('Ali', 'IT', 7000), ('Omar', 'IT', 9000), ('Zara', 'IT', 5500),
  ('Sara', 'HR', 6000), ('Hina', 'HR', 4500);
SELECT dept, MAX(salary), MIN(salary)
FROM emp
GROUP BY dept
ORDER BY dept;
```

A. IT, 9000, 4500
B. IT, 7000, 5500
C. IT, 9000, 5500
D. IT, 9000, 7000

531. The scholarship table contains a NULL. Which student names does the NOT IN query return? Analysis

```
CREATE TABLE students (id INT, name TEXT);
INSERT INTO students VALUES (1, 'Ali'), (2, 'Sara'), (3, 'Omar');
CREATE TABLE scholarship (student_id INT);
INSERT INTO scholarship VALUES (1), (NULL);
SELECT name FROM students
WHERE id NOT IN (SELECT student_id FROM scholarship);
```

A. Sara, Omar
B. Ali
C. Ali, Sara, Omar
D. No rows

532. Compare the results of Q1 and Q2 on this table. Analysis

```
CREATE TABLE emp (name TEXT, dept TEXT);
INSERT INTO emp VALUES ('A','IT'),('B','IT'),('C','IT'),('D','HR'),('E','HR'),('F','Admin');

-- Q1
SELECT dept, COUNT(*) FROM emp
WHERE dept <> 'HR'
GROUP BY dept;

-- Q2
SELECT dept, COUNT(*) FROM emp
GROUP BY dept
HAVING dept <> 'HR';
```

A. Both return the same rows; filtering in WHERE is usually better because rows are dropped before grouping
B. Q2 is invalid because HAVING may only contain aggregate functions
C. Q1 adds the HR employees' count into the remaining groups
D. They differ: Q2 still returns HR with a count of 0

533. What three values does the query return? Analysis

```
CREATE TABLE r (score INT);
INSERT INTO r VALUES (10), (20), (NULL), (30), (NULL);
SELECT AVG(score),
       SUM(score) * 1.0 / COUNT(*),
       SUM(score) * 1.0 / COUNT(score)
FROM r;
```

A. 12, 12, 20
B. 20, 20, 20
C. 20, 12, 20
D. NULL, 12, 20

534. What does the query return for the Admin department? Analysis

```
CREATE TABLE dept (id INT, name TEXT);
INSERT INTO dept VALUES (1, 'HR'), (2, 'IT'), (3, 'Admin');
CREATE TABLE emp (id INT, dept_id INT);
INSERT INTO emp VALUES (1, 1), (2, 1), (3, 2);
SELECT d.name, COUNT(*) AS c1, COUNT(e.id) AS c2
FROM dept d LEFT JOIN emp e ON e.dept_id = d.id
GROUP BY d.name
ORDER BY d.name;
```

A. Admin, 0, 0
B. Admin, 1, 0
C. Admin, 1, 1
D. No Admin row appears

535. After this transaction commits, which ids remain in the log table? Analysis

```
CREATE TABLE log (id INT);
BEGIN;
INSERT INTO log VALUES (1), (2);
SAVEPOINT sp1;
INSERT INTO log VALUES (3);
DELETE FROM log WHERE id = 1;
ROLLBACK TO SAVEPOINT sp1;
INSERT INTO log VALUES (4);
COMMIT;
SELECT id FROM log ORDER BY id;
```

A. 1, 2, 4
B. 2, 3, 4
C. 4
D. 1, 2, 3, 4

536. How many rows does the query return, and what count is shown for the NULL city? Analysis

```
CREATE TABLE p (city TEXT);
INSERT INTO p VALUES ('Lahore'), (NULL), ('Karachi'), (NULL), ('Lahore');
SELECT city, COUNT(*) FROM p GROUP BY city;
```

A. 2 rows; NULL cities are left out of grouping
B. 4 rows; each NULL forms its own group
C. 3 rows; the NULL group shows 0
D. 3 rows; the NULL group shows 2

537. Table emp(name, dept, salary). Which query is invalid in standard SQL? Analysis

A. SELECT dept, COUNT(*) FROM emp GROUP BY dept;
B. SELECT COUNT(*) FROM emp;
C. SELECT dept, MAX(salary) FROM emp GROUP BY dept HAVING COUNT(*) > 1;
D. SELECT dept, name, COUNT(*) FROM emp GROUP BY dept;

538. What do query 1 and query 2 return? Analysis

```
CREATE TABLE v (color TEXT);
INSERT INTO v VALUES ('red'), ('blue'), (NULL), ('red'), (NULL);
SELECT COUNT(DISTINCT color) FROM v;   -- query 1
SELECT DISTINCT color FROM v;          -- query 2
```

A. Query 1 returns 3; query 2 returns 3 rows
B. Query 1 returns 2; query 2 returns 3 rows
C. Query 1 returns 2; query 2 returns 2 rows
D. Query 1 returns 3; query 2 returns 2 rows

539. Which names match the pattern '%a_'? Analysis

```
CREATE TABLE people (name TEXT);
INSERT INTO people VALUES
  ('sana'), ('amna'), ('faraz'), ('zara'), ('bilal'), ('asad');
SELECT name FROM people WHERE name LIKE '%a_' ORDER BY name;
```

A. amna, sana, zara
B. asad, bilal, faraz
C. amna, asad, bilal, faraz, sana, zara
D. amna, asad

540. Given SQL operator precedence, which names does this WHERE clause return? Analysis

```
CREATE TABLE emp (name TEXT, dept TEXT, salary INT);
INSERT INTO emp VALUES
  ('Ali', 'IT', 40000), ('Sara', 'HR', 60000), ('Omar', 'IT', 70000),
  ('Hina', 'HR', 30000), ('Zain', 'Admin', 80000);
SELECT name FROM emp
WHERE dept = 'IT' OR dept = 'HR' AND salary > 50000
ORDER BY name;
```

A. Omar, Sara
B. Omar, Sara, Zain
C. Ali, Omar, Sara
D. Ali, Hina, Omar, Sara

541. A column is declared CHAR(10) and another VARCHAR(10). Both store the value 'Ali'. What is the difference? Understanding

A. VARCHAR(10) pads the value with spaces to 10 characters; CHAR(10) stores only the 3 characters used
B. CHAR(10) pads the value with spaces to 10 characters; VARCHAR(10) stores only the 3 characters used
C. CHAR(10) can hold at most 10 values, while VARCHAR(10) can hold any number
D. CHAR(10) accepts only letters, while VARCHAR(10) also accepts digits

542. Which statement adds a new column email to the existing table student? Understanding

A. ALTER TABLE student ADD email VARCHAR(50);
B. UPDATE student ADD email VARCHAR(50);
C. INSERT INTO student COLUMN email VARCHAR(50);
D. CREATE COLUMN email VARCHAR(50) ON student;

543. A column is declared status VARCHAR(10) DEFAULT 'active'. When is the value 'active' stored? Understanding

A. When an INSERT does not supply a value for status
B. Whenever an INSERT supplies NULL or an empty string for status
C. Only for the first row inserted into the table
D. Whenever an UPDATE changes another column of the row

544. A query ends with ORDER BY salary and gives no direction. How are the rows sorted? Understanding

A. Descending, from the highest salary to the lowest
B. In the order the rows were inserted
C. Ascending, from the lowest salary to the highest
D. In random order, because no direction was given

545. In a LIKE pattern, what does the % wildcard match? Understanding

A. Any sequence of zero or more characters
B. Exactly one character
C. One or more digits only
D. The percent sign itself

546. Which condition is equivalent to dept IN ('Sales', 'Tech')? Understanding

A. dept = 'Sales' AND dept = 'Tech'
B. dept BETWEEN 'Sales' AND 'Tech'
C. dept LIKE 'Sales%Tech'
D. dept = 'Sales' OR dept = 'Tech'

547. In SELECT salary * 12 AS annual FROM staff, what does AS annual do? Understanding

A. It creates a new column annual in the staff table
B. It gives the computed output column the name annual
C. It stores the result in a table called annual
D. It filters the rows to those where annual is true

548. Table enrol is declared with PRIMARY KEY (student_id, course_id). What does this require? Understanding

A. Each (student_id, course_id) combination must be unique, and neither column may be NULL
B. student_id and course_id must each be unique on their own
C. Only student_id must be unique; course_id is a foreign key
D. The two columns must always hold the same value

549. A transaction updates some rows and then runs COMMIT. A later ROLLBACK is issued in the same session. What happens to the committed update? Understanding

A. It is undone, because ROLLBACK reverses every change in the session
B. It is undone only if no other user has read the data
C. The table is restored to the state it had when it was created
D. It stays; ROLLBACK can only undo changes made since the last COMMIT

550. Which integrity rule says that no primary key value may be NULL? Understanding

A. Entity integrity
B. Referential integrity
C. Domain integrity
D. User-defined integrity

551. The owner runs GRANT SELECT ON staff TO ali WITH GRANT OPTION. What can ali now do? Understanding

A. Read and modify staff, but not grant anything
B. Read staff, but only after the owner grants permission again
C. Read staff and grant SELECT on staff to other users
D. Drop the staff table and recreate it

552. A query uses HAVING COUNT(*) > 5 but has no GROUP BY clause. How does standard SQL treat it? Understanding

A. The query is always rejected, because HAVING requires GROUP BY
B. The whole result of FROM and WHERE is one group, so the query returns one row or none
C. HAVING is treated as WHERE and applied to each row
D. Each row becomes its own group of size 1

553. Standard SQL rejects SELECT salary * 12 AS annual FROM staff WHERE annual > 6000. Why? Understanding

A. Aliases cannot be given to computed columns
B. WHERE can only compare columns with other columns, not with constants
C. The multiplication must be repeated in ORDER BY first
D. WHERE is evaluated before the SELECT list, so the alias annual does not exist yet

554. emp.dept_id is declared REFERENCES dept(id) ON DELETE SET NULL. What happens to an employee row when its department is deleted? Understanding

A. The employee row is deleted as well
B. The department delete is rejected
C. The row stays, and its dept_id becomes NULL
D. dept_id is reset to the column's DEFAULT value

555. A foreign key is declared with no ON DELETE clause, so the default NO ACTION applies. What happens when someone deletes a parent row that child rows still reference? Understanding

A. The child rows are deleted too
B. The child rows' foreign keys are set to NULL
C. The parent row is deleted and the child rows keep the old value
D. The delete is rejected with a constraint violation

556. In standard SQL, which columns of the parent table can a FOREIGN KEY reference? Understanding

A. Any column, as long as the data types match
B. Its primary key or a column set declared UNIQUE
C. Only an integer column with an index
D. Only a column that is also a foreign key in another table

557. Which aggregate functions can be applied to a VARCHAR column such as name in standard SQL? Understanding

A. SUM and AVG
B. COUNT, MIN and MAX
C. All five: COUNT, SUM, AVG, MIN and MAX
D. Only COUNT

558. Under SQL's three-valued logic, which expression evaluates to TRUE? Understanding

A. TRUE AND UNKNOWN
B. TRUE OR UNKNOWN
C. NOT UNKNOWN
D. UNKNOWN OR FALSE

559. What does INSERT INTO archive (id, name) SELECT id, name FROM staff WHERE dept = 'Sales' do? Understanding

A. Inserts one row, the first Sales employee found
B. Copies every Sales row's id and name into archive, possibly many rows at once
C. Moves the Sales rows, deleting them from staff
D. Creates the table archive with columns id and name

560. Which clause order is syntactically valid for a SELECT statement? Understanding

A. SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY
B. SELECT, FROM, GROUP BY, WHERE, HAVING, ORDER BY
C. SELECT, FROM, WHERE, HAVING, ORDER BY, GROUP BY
D. FROM, WHERE, SELECT, GROUP BY, ORDER BY, HAVING

561. Table t has a row with a = 1 and b = 2. After UPDATE t SET a = b, b = a, what does the row hold in standard SQL? Understanding

A. a = 2, b = 1
B. a = 2, b = 2
C. a = 1, b = 1
D. a = 1, b = 2

562. The owner grants SELECT to ali WITH GRANT OPTION, and ali grants SELECT to sara. The owner then runs REVOKE SELECT ON staff FROM ali CASCADE. What happens? Understanding

A. Only ali loses SELECT; sara keeps it
B. Only sara loses SELECT
C. Both ali and sara lose SELECT
D. The REVOKE fails because ali has granted the privilege to someone

563. What does SELECT DISTINCT dept, name FROM staff remove? Understanding

A. Rows with a repeated dept, keeping one name per department
B. Rows with a repeated name, whatever their department
C. Rows where both dept and name repeat together
D. Nothing, because DISTINCT can only follow a single column

564. A column is declared city VARCHAR(5). In standard SQL, what happens to INSERT ... VALUES ('Karachi')? Understanding

A. It stores 'Karac', silently cutting off the rest
B. It stores 'Karachi' and widens the column to 7
C. It is rejected, because 'Karachi' is longer than 5 characters
D. It stores NULL, because the value does not fit

565. What does the query return? Application

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
SELECT COUNT(*) FROM staff WHERE dept = 'Tech' AND salary >= 600;
```

A. 3
B. 1
C. 4
D. 2

566. Which name appears in the first row of the result? Application

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
SELECT name FROM staff ORDER BY dept, salary DESC;
```

A. Faisal
B. Mehwish
C. Bushra
D. Asim

567. What value does the query return for Bushra? Application

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
SELECT name, salary + COALESCE(bonus, 0) AS total FROM staff WHERE name = 'Bushra';
```

A. 550
B. NULL
C. 0
D. 600

568. How many rows does the query count? Application

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
SELECT COUNT(*) FROM staff WHERE salary NOT BETWEEN 450 AND 650;
```

A. 4
B. 5
C. 2
D. 3

569. A column is declared price DECIMAL(6, 2). What is the largest value it can store? Application

A. 999999.99
B. 9999.99
C. 999999
D. 99.999999

570. What does the final query return? Application

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
ALTER TABLE staff ADD COLUMN status TEXT DEFAULT 'active';
SELECT COUNT(*) FROM staff WHERE status = 'active';
```

A. 0
B. NULL
C. 1
D. 7

571. What does the final query return? Application

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
UPDATE staff SET salary = salary + 100 WHERE dept = 'Sales';
SELECT SUM(salary) FROM staff WHERE dept = 'Sales';
```

A. 1700
B. 1400
C. 1500
D. 3850

572. What row does the query return for Tech? Application

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
SELECT dept, COUNT(*), COUNT(bonus) FROM staff GROUP BY dept;
```

A. Tech, 3, 2
B. Tech, 3, 3
C. Tech, 2, 2
D. Tech, 3, 100

573. What does the query return? Application

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
SELECT COUNT(*) FROM staff HAVING AVG(salary) > 500;
```

A. No rows
B. One row with the value 7
C. One row with the value 3
D. An error, because there is no GROUP BY

574. How many employees are counted in band B? Application

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
SELECT CASE WHEN salary >= 600 THEN 'A'
            WHEN salary >= 450 THEN 'B'
            ELSE 'C' END AS band,
       COUNT(*)
FROM staff
GROUP BY band;
```

A. 2
B. 4
C. 5
D. 3

575. This query uses the LIMIT ... OFFSET syntax of SQLite, MySQL and PostgreSQL. Which name does it return? Application

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
SELECT name FROM staff ORDER BY salary DESC LIMIT 1 OFFSET 1;
```

A. Faisal
B. Bushra
C. Javed
D. Hira

576. How many rows does the query return? Application

```
CREATE TABLE enrol (student TEXT, course TEXT, sem INT);
INSERT INTO enrol VALUES
  ('S1', 'DB', 1), ('S1', 'OS', 1), ('S2', 'DB', 1),
  ('S2', 'DB', 2), ('S3', 'OS', 2), ('S3', 'DB', 1);
SELECT course, sem, COUNT(*) FROM enrol GROUP BY course, sem;
```

A. 3
B. 4
C. 2
D. 6

577. How many codes does the query count? Application

```
CREATE TABLE promo (code TEXT);
INSERT INTO promo VALUES ('50%OFF'), ('5OFF'), ('150%'), ('50OFF%');
SELECT COUNT(*) FROM promo WHERE code LIKE '%0!%%' ESCAPE '!';
```

A. 1
B. 3
C. 4
D. 2

578. Given this table, which INSERT is rejected? Application

```
CREATE TABLE enrol (
  sid INT NOT NULL,
  cid INT NOT NULL,
  grade TEXT,
  PRIMARY KEY (sid, cid)
);
INSERT INTO enrol VALUES (1, 101, 'A'), (1, 102, 'B'), (2, 101, 'B');
```

A. INSERT INTO enrol VALUES (2, 102, 'A');
B. INSERT INTO enrol VALUES (3, 101, 'A');
C. INSERT INTO enrol VALUES (1, 103, NULL);
D. INSERT INTO enrol VALUES (2, 101, 'C');

579. Assume foreign keys are enforced. What does the final query return? Application

```
CREATE TABLE dept (id INT PRIMARY KEY, name TEXT);
CREATE TABLE emp (
  id INT PRIMARY KEY,
  dept_id INT REFERENCES dept(id) ON DELETE SET NULL
);
INSERT INTO dept VALUES (1, 'HR'), (2, 'IT');
INSERT INTO emp VALUES (10, 1), (11, 2), (12, 2), (13, NULL);
DELETE FROM dept WHERE id = 2;
SELECT COUNT(*), COUNT(dept_id) FROM emp;
```

A. 2 and 1
B. 4 and 3
C. 4 and 2
D. 4 and 1

580. Which name appears in the first row of the result? Application

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
SELECT name, salary FROM staff ORDER BY 2 DESC;
```

A. Asim
B. Rizwan
C. Faisal
D. Mehwish

581. The average salary before the update is about 507. What does the final query return? Application

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
UPDATE staff SET salary = salary + 50
WHERE salary < (SELECT AVG(salary) FROM staff);
SELECT SUM(salary) FROM staff;
```

A. 3550
B. 3750
C. 3800
D. 3700

582. Run in SQLite, what are the two values returned? Application

```
CREATE TABLE m (x INT);
INSERT INTO m VALUES (1), (2), (4);
SELECT SUM(x) / COUNT(x), AVG(x) FROM m;
```

A. 2.333... and 2.333...
B. 2 and 2
C. 3 and 2.333...
D. 2 and 2.333...

583. What does the query return? Application

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
SELECT dept, SUM(salary) AS total
FROM staff
WHERE bonus IS NOT NULL
GROUP BY dept
HAVING COUNT(*) >= 2
ORDER BY total DESC;
```

A. Sales 1400, then Tech 1850
B. Tech 1850, then Sales 1400
C. Tech 1200, then Sales 850
D. Sales 850, then Tech 1200

584. What rows does top_pay contain at the end? Application

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
CREATE TABLE top_pay (dept TEXT, top INT);
INSERT INTO top_pay
  SELECT dept, MAX(salary) FROM staff
  WHERE bonus IS NOT NULL
  GROUP BY dept;
SELECT * FROM top_pay ORDER BY dept;
```

A. (Sales, 450) and (Tech, 700)
B. (Admin, 300), (Sales, 550) and (Tech, 700)
C. (Sales, 550) and (Tech, 700)
D. (Admin, NULL), (Sales, 450) and (Tech, 700)

585. What does the final query return? Application

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
DELETE FROM staff
WHERE salary < (SELECT AVG(salary) FROM staff WHERE dept = 'Tech');
SELECT COUNT(*) FROM staff;
```

A. 5
B. 2
C. 4
D. 3

586. A developer meant to remove one employee but ran DELETE FROM staff; with no WHERE clause, outside any transaction. What is the state afterwards? Analysis

A. The table and its definition are gone
B. Only the first row has been deleted
C. The table still exists, with 0 rows
D. Nothing changed, because DELETE needs a WHERE clause

587. Why does this query return no rows, although several salaries lie between 450 and 650? Analysis

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
SELECT name FROM staff WHERE salary BETWEEN 650 AND 450;
```

A. BETWEEN excludes both end values, and no salary lies strictly inside
B. BETWEEN only works on dates and strings, not on integers
C. BETWEEN a AND b means >= a AND <= b, so with the larger bound first no value can satisfy both
D. The query returns 650 and 450 only, because they match the bounds exactly

588. What do Q1 and Q2 return? Analysis

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
SELECT COUNT(*) FROM staff WHERE NOT (bonus > 25);       -- Q1
SELECT COUNT(*) FROM staff WHERE bonus <= 25 OR bonus IS NULL;  -- Q2
```

A. Q1 returns 1; Q2 returns 4
B. Both return 4
C. Both return 1
D. Q1 returns 4; Q2 returns 1

589. No employee is in Legal. What does the query return? Analysis

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
SELECT COUNT(*), SUM(salary), MAX(salary) FROM staff WHERE dept = 'Legal';
```

A. One row: 0, NULL, NULL
B. No rows at all
C. One row: 0, 0, 0
D. One row: NULL, NULL, NULL

590. A manager wants every department with its number of staff earning above 600, including departments with none. What does this query return? Analysis

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
SELECT dept, COUNT(*) FROM staff WHERE salary > 600 GROUP BY dept;
```

A. (Admin, 0), (Sales, 0) and (Tech, 2)
B. (Tech, 3), because the whole Tech group is counted
C. Only (Tech, 2); Sales and Admin do not appear
D. (Sales, 3) and (Tech, 3)

591. Assume a DBMS that accepts the DEFAULT keyword in VALUES, as standard SQL, PostgreSQL and MySQL do. What does the final query return? Analysis

```
CREATE TABLE ticket (id INT, status TEXT DEFAULT 'new');
INSERT INTO ticket (id) VALUES (1);
INSERT INTO ticket (id, status) VALUES (2, NULL);
INSERT INTO ticket VALUES (3, DEFAULT);
SELECT COUNT(*) FROM ticket WHERE status = 'new';
```

A. 2
B. 3
C. 1
D. 0

592. Table t has id INT PRIMARY KEY and already holds the row (1, 'a'). The statement INSERT INTO t VALUES (2, 'b'), (1, 'c'), (3, 'd') fails with a key violation. How many rows does t hold afterwards? Analysis

A. 2
B. 3
C. 1
D. 4

593. Three courses exist: DB, OS and AI. S2 repeated DB, so it appears twice. Which query correctly finds students who took all three courses, and what does each return? Analysis

```
CREATE TABLE enrol (sid TEXT, course TEXT);
INSERT INTO enrol VALUES
  ('S1', 'DB'), ('S1', 'OS'), ('S1', 'AI'),
  ('S2', 'DB'), ('S2', 'DB'), ('S2', 'OS'),
  ('S3', 'DB'), ('S3', 'OS');
SELECT sid FROM enrol GROUP BY sid HAVING COUNT(*) = 3;               -- Q1
SELECT sid FROM enrol GROUP BY sid HAVING COUNT(DISTINCT course) = 3; -- Q2
```

A. Q1 is correct and returns S1 and S2; Q2 returns S1
B. Both are correct and return S1
C. Q2 is correct and returns S1; Q1 returns S1 and S2
D. Q2 is correct and returns S1 and S3; Q1 returns S1

594. Assume foreign keys are enforced. What does the final query return? Analysis

```
CREATE TABLE dept (id INT PRIMARY KEY);
CREATE TABLE emp (id INT PRIMARY KEY,
  dept_id INT REFERENCES dept(id) ON DELETE CASCADE);
CREATE TABLE project (id INT PRIMARY KEY,
  dept_id INT REFERENCES dept(id) ON DELETE SET NULL);
INSERT INTO dept VALUES (1), (2);
INSERT INTO emp VALUES (10, 1), (11, 2), (12, 2);
INSERT INTO project VALUES (100, 2), (101, 2), (102, 1);
DELETE FROM dept WHERE id = 2;
SELECT (SELECT COUNT(*) FROM emp), (SELECT COUNT(*) FROM project),
       (SELECT COUNT(dept_id) FROM project);
```

A. 1, 1, 1
B. 3, 3, 1
C. 1, 3, 1
D. 1, 3, 3

595. Which query returns the departments in which every employee earns more than 400? Analysis

A. SELECT dept FROM staff GROUP BY dept HAVING MIN(salary) > 400;
B. SELECT dept FROM staff WHERE salary > 400 GROUP BY dept;
C. SELECT dept FROM staff GROUP BY dept HAVING MAX(salary) > 400;
D. SELECT dept FROM staff GROUP BY dept HAVING AVG(salary) > 400;

596. Mehwish is the only Admin employee, and her bonus is NULL. What row does the query return for Admin? Analysis

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
SELECT dept, COUNT(*), COUNT(bonus), AVG(bonus)
FROM staff GROUP BY dept;
```

A. Admin, 1, 1, NULL
B. Admin, 1, 0, NULL
C. Admin, 1, 0, 0
D. Admin, 0, 0, NULL

597. Which of these queries is invalid in standard SQL? Analysis

A. SELECT dept FROM staff ORDER BY salary;
B. SELECT name, salary * 12 AS annual FROM staff ORDER BY annual;
C. SELECT name, salary FROM staff ORDER BY 2;
D. SELECT DISTINCT dept FROM staff ORDER BY salary;

598. What does the final query return? Analysis

```
CREATE TABLE t (v INT);
INSERT INTO t VALUES (1), (2), (3);
BEGIN;
INSERT INTO t VALUES (4);
COMMIT;
BEGIN;
DELETE FROM t;
INSERT INTO t VALUES (9);
ROLLBACK;
SELECT COUNT(*) FROM t;
```

A. 1
B. 3
C. 0
D. 4

599. How do the results of Q1 and Q2 differ? Analysis

```
CREATE TABLE staff (id INT, name TEXT, dept TEXT, salary INT, bonus INT);
INSERT INTO staff VALUES
  (1, 'Asim', 'Sales', 400, 50), (2, 'Bushra', 'Sales', 550, NULL),
  (3, 'Faisal', 'Tech', 700, 80), (4, 'Hira', 'Tech', 650, NULL),
  (5, 'Javed', 'Tech', 500, 20), (6, 'Mehwish', 'Admin', 300, NULL),
  (7, 'Rizwan', 'Sales', 450, 30);
SELECT dept, COUNT(*) FROM staff WHERE bonus IS NOT NULL GROUP BY dept;  -- Q1
SELECT dept, COUNT(bonus) FROM staff GROUP BY dept;                      -- Q2
```

A. They are identical, including the Admin row
B. Q1 lists Admin with 0; Q2 omits Admin
C. Q2 also lists Admin with 0; otherwise the counts match (Sales 2, Tech 2)
D. Q2 counts every row, giving Sales 3 and Tech 3

600. The codes are stored as text. What does the query return? Analysis

```
CREATE TABLE item (code VARCHAR(5));
INSERT INTO item VALUES ('9'), ('10'), ('100'), ('25');
SELECT MAX(code) FROM item;
```

A. '100'
B. '9'
C. '25'
D. '10'

Advanced SQL MCQs for NSCT

601. What makes a subquery correlated? Understanding

A. It returns more than one column
B. It appears in the FROM clause as a derived table
C. It is joined to the outer query with an explicit JOIN keyword
D. It refers to a column of the outer query, so it is logically evaluated once per outer row

602. Which statement best describes an ordinary (non-materialised) view? Understanding

A. A physical copy of selected rows that is refreshed every night
B. An index that speeds up queries on the base table
C. A temporary table that is dropped at the end of the session
D. A named, stored query whose result is computed from the base tables whenever it is used

603. A trigger in SQL is best described as: Understanding

A. A procedure that a user must call explicitly with CALL or EXEC
B. Code the DBMS runs automatically when a specified event, such as INSERT, UPDATE or DELETE, occurs on a table
C. A constraint that can only check a single column value
D. A saved query that behaves like a virtual table

604. What is the main difference between a stored procedure and a trigger? Understanding

A. A stored procedure cannot take parameters; a trigger can
B. A stored procedure runs when it is called explicitly; a trigger runs automatically on a table event
C. A trigger is stored on the client; a stored procedure is stored on the server
D. A stored procedure can only read data; a trigger can only write data

605. How does UNION differ from UNION ALL? Understanding

A. UNION removes duplicate rows from the combined result; UNION ALL keeps them
B. UNION ALL removes duplicates; UNION keeps them
C. UNION needs both queries to have the same column names; UNION ALL does not
D. UNION sorts by the first column; UNION ALL returns rows in random order

606. A table employee(id, name, manager_id) is joined with itself to list each employee beside their manager's name. What is this called? Understanding

A. A natural join
B. A cross join
C. A semi join
D. A self join

607. Which join keeps every row from both tables, filling the missing side with NULLs when there is no match? Understanding

A. LEFT OUTER JOIN
B. INNER JOIN
C. CROSS JOIN
D. FULL OUTER JOIN

608. Why would a DBA create a materialised view instead of an ordinary view? Understanding

A. To make the view always reflect base-table changes instantly with no storage cost
B. To allow the view to be updated even when it contains GROUP BY
C. To hide columns from users, which an ordinary view cannot do
D. To store a costly query result physically so reads are fast, accepting that it must be refreshed

609. A view is defined as SELECT * FROM product WHERE price < 100 WITH CHECK OPTION. What does WITH CHECK OPTION do? Understanding

A. Re-checks every base-table constraint each time the view is queried
B. Rejects an INSERT or UPDATE through the view that would create a row with price of 100 or more
C. Makes the view read-only
D. Refreshes the view automatically when the base table changes

610. In a row-level AFTER UPDATE trigger, what do OLD and NEW refer to? Understanding

A. The previous and the next row in the table
B. The table's definition before and after an ALTER TABLE
C. The values of the affected row before and after the update
D. The first and the last row changed by the statement

611. When does WHERE EXISTS (subquery) evaluate to TRUE? Understanding

A. When the subquery returns a non-NULL value in its first column
B. When the subquery returns exactly one row
C. When the subquery returns at least one row, whatever values that row holds
D. When every row returned by the subquery is TRUE

612. Which kind of view is generally NOT updatable in standard SQL? Understanding

A. A view that selects some columns of one table, including its primary key
B. A view on one table with a simple WHERE condition
C. A view that renames the columns of one table
D. A view that uses GROUP BY with aggregate functions

613. How do RANK() and DENSE_RANK() differ when there are ties? Understanding

A. DENSE_RANK() skips numbers after a tie; RANK() does not
B. RANK() gives tied rows different numbers; DENSE_RANK() gives them the same number
C. They differ only when the ORDER BY column contains NULLs
D. RANK() skips numbers after a tie; DENSE_RANK() does not

614. A scalar subquery is used in a SELECT list. What must it return? Understanding

A. Exactly one column and any number of rows
B. Exactly one row and any number of columns
C. At most one row with exactly one column
D. A single table that can be joined

615. What does a NATURAL JOIN do? Understanding

A. Joins on the primary key and foreign key declared between the tables
B. Returns the Cartesian product of the two tables
C. Joins on the first column of each table
D. Joins on equality of all columns with the same name and shows each such column once

616. In standard SQL, what is the result of salary > ALL (subquery) when the subquery returns no rows? Understanding

A. FALSE for every row
B. TRUE for every row
C. UNKNOWN for every row
D. A run-time error

617. How many rows does this LEFT JOIN return? Application

```
CREATE TABLE student (id INT, name TEXT);
INSERT INTO student VALUES (1,'Ali'),(2,'Sara'),(3,'Omar'),(4,'Hina');
CREATE TABLE enrol (sid INT, course TEXT);
INSERT INTO enrol VALUES (1,'DB'),(1,'OS'),(2,'DB'),(9,'AI');
SELECT s.name, e.course
FROM student s LEFT JOIN enrol e ON s.id = e.sid;
```

A. 5
B. 3
C. 4
D. 6

618. What does the query return? Application

```
CREATE TABLE sizes (s TEXT);
INSERT INTO sizes VALUES ('S'),('M'),('L');
CREATE TABLE colours (c TEXT);
INSERT INTO colours VALUES ('red'),('blue'),('green'),('black');
SELECT COUNT(*) FROM sizes CROSS JOIN colours;
```

A. 7
B. 12
C. 4
D. 3

619. What do the two queries return, in order? Application

```
CREATE TABLE lahore (name TEXT);
INSERT INTO lahore VALUES ('Ali'),('Sara'),('Ali');
CREATE TABLE karachi (name TEXT);
INSERT INTO karachi VALUES ('Sara'),('Omar');
SELECT COUNT(*) FROM (SELECT name FROM lahore UNION SELECT name FROM karachi);
SELECT COUNT(*) FROM (SELECT name FROM lahore UNION ALL SELECT name FROM karachi);
```

A. 5 and 3
B. 3 and 3
C. 3 and 5
D. 4 and 5

620. Which names does the query return? Application

```
CREATE TABLE dept (did INT, dname TEXT, city TEXT);
INSERT INTO dept VALUES (10,'IT','Lahore'),(20,'HR','Karachi'),(30,'Sales','Lahore'),
(40,'Legal','Quetta');
CREATE TABLE emp (eid INT, name TEXT, did INT, salary INT);
INSERT INTO emp VALUES
  (1,'Ali',10,90),(2,'Sara',10,60),(3,'Omar',20,50),
  (4,'Hina',20,70),(5,'Zain',30,40);
SELECT name FROM emp
WHERE did IN (SELECT did FROM dept WHERE city = 'Lahore')
ORDER BY name;
```

A. Ali, Sara, Zain
B. Ali, Sara
C. Zain
D. Ali, Hina, Sara, Zain

621. Which names does the correlated query return? Application

```
CREATE TABLE dept (did INT, dname TEXT, city TEXT);
INSERT INTO dept VALUES (10,'IT','Lahore'),(20,'HR','Karachi'),(30,'Sales','Lahore'),
(40,'Legal','Quetta');
CREATE TABLE emp (eid INT, name TEXT, did INT, salary INT);
INSERT INTO emp VALUES
  (1,'Ali',10,90),(2,'Sara',10,80),(3,'Omar',20,50),
  (4,'Hina',20,70),(5,'Zain',30,40);
SELECT e.name FROM emp e
WHERE e.salary > (SELECT AVG(x.salary) FROM emp x
                  WHERE x.did = e.did)
ORDER BY e.name;
```

A. Ali, Hina
B. Ali, Hina, Sara
C. Ali
D. Ali, Hina, Omar

622. Which department names does the query return? Application

```
CREATE TABLE dept (did INT, dname TEXT, city TEXT);
INSERT INTO dept VALUES (10,'IT','Lahore'),(20,'HR','Karachi'),(30,'Sales','Lahore'),
(40,'Legal','Quetta');
CREATE TABLE emp (eid INT, name TEXT, did INT, salary INT);
INSERT INTO emp VALUES
  (1,'Ali',10,90),(2,'Sara',10,60),(3,'Omar',20,50),
  (4,'Hina',20,70),(5,'Zain',30,40);
SELECT d.dname FROM dept d
WHERE NOT EXISTS (SELECT 1 FROM emp e WHERE e.did = d.did);
```

A. Sales
B. IT, HR, Sales
C. Legal
D. No rows

623. After the INSERT and UPDATE on the base table, what does the final SELECT return? Application

```
CREATE TABLE product (id INT, name TEXT, price INT);
INSERT INTO product VALUES (1,'Pen',50),(2,'Bag',900);
CREATE VIEW cheap AS SELECT name FROM product WHERE price < 100;
INSERT INTO product VALUES (3,'Ruler',30);
UPDATE product SET price = 80 WHERE id = 2;
SELECT COUNT(*) FROM cheap;
```

A. 1
B. 2
C. An error, because the view was created before the changes
D. 3

624. The script uses SQLite's trigger syntax. What does the final SELECT return? Application

```
CREATE TABLE orders (id INT, qty INT);
CREATE TABLE stats (n INT, total INT);
INSERT INTO stats VALUES (0, 0);
CREATE TRIGGER trg AFTER INSERT ON orders
FOR EACH ROW
BEGIN
  UPDATE stats SET n = n + 1, total = total + NEW.qty;
END;
INSERT INTO orders VALUES (1, 5), (2, 3);
INSERT INTO orders VALUES (3, 2);
SELECT n, total FROM stats;
```

A. 2, 10
B. 2, 8
C. 3, 10
D. 1, 2

625. What RANK and DENSE_RANK values does student D get? Application

```
CREATE TABLE score (name TEXT, marks INT);
INSERT INTO score VALUES ('A',90),('B',85),('C',90),('D',80),('E',85);
SELECT name,
       RANK()       OVER (ORDER BY marks DESC) AS r,
       DENSE_RANK() OVER (ORDER BY marks DESC) AS dr
FROM score
ORDER BY name;
```

A. r = 3, dr = 3
B. r = 5, dr = 5
C. r = 5, dr = 3
D. r = 4, dr = 3

626. How many rows does this self join return, and who appears as a manager? Application

```
CREATE TABLE staff (id INT, name TEXT, mgr INT);
INSERT INTO staff VALUES (1,'Asma',NULL),(2,'Bilal',1),(3,'Danish',1),(4,'Fatima',2);
SELECT w.name, m.name
FROM staff w JOIN staff m ON w.mgr = m.id
ORDER BY w.name;
```

A. 4 rows; Asma, Bilal and NULL appear as managers
B. 3 rows; Asma and Bilal appear as managers
C. 3 rows; only Asma appears as a manager
D. 2 rows; Asma and Bilal appear as managers

627. What does the EXCEPT query return? Application

```
CREATE TABLE a (x INT);
INSERT INTO a VALUES (1),(2),(2),(3),(4);
CREATE TABLE b (x INT);
INSERT INTO b VALUES (2),(4),(5);
SELECT x FROM a EXCEPT SELECT x FROM b;
```

A. 1, 3 and 5
B. 1, 2 and 3
C. 1 and 3
D. 2 and 4

628. What does the query return? Application

```
CREATE TABLE emp (name TEXT, salary INT);
INSERT INTO emp VALUES ('Ali',70),('Sara',90),('Omar',90),('Hina',60);
SELECT MAX(salary) FROM emp
WHERE salary < (SELECT MAX(salary) FROM emp);
```

A. 90
B. 60
C. 70
D. NULL

629. What does the recursive common table expression return? Application

```
WITH RECURSIVE n(k) AS (
  SELECT 1
  UNION ALL
  SELECT k + 2 FROM n WHERE k < 9
)
SELECT COUNT(*), SUM(k) FROM n;
```

A. 4 and 16
B. 5 and 35
C. 6 and 36
D. 5 and 25

630. Using the dept and emp tables (emp.did references dept.did), which query lists departments that have no employees? Application

A. SELECT d.dname FROM dept d JOIN emp e ON d.did = e.did WHERE e.eid IS NULL
B. SELECT d.dname FROM dept d LEFT JOIN emp e ON d.did = e.did WHERE e.eid IS NULL
C. SELECT d.dname FROM dept d LEFT JOIN emp e ON d.did = e.did WHERE e.eid = NULL
D. SELECT d.dname FROM dept d LEFT JOIN emp e ON d.did = e.did AND e.eid IS NULL

631. The two queries differ only in where the salary filter is placed. What counts do they return, in order? Analysis

```
CREATE TABLE dept (did INT, dname TEXT, city TEXT);
INSERT INTO dept VALUES (10,'IT','Lahore'),(20,'HR','Karachi'),(30,'Sales','Lahore'),
(40,'Legal','Quetta');
CREATE TABLE emp (eid INT, name TEXT, did INT, salary INT);
INSERT INTO emp VALUES
  (1,'Ali',10,90),(2,'Sara',10,60),(3,'Omar',20,50),
  (4,'Hina',20,70),(5,'Zain',30,40);
SELECT COUNT(*) FROM dept d LEFT JOIN emp e
  ON d.did = e.did AND e.salary > 55;
SELECT COUNT(*) FROM dept d LEFT JOIN emp e
  ON d.did = e.did WHERE e.salary > 55;
```

A. 3 and 3
B. 5 and 3
C. 5 and 5
D. 3 and 5

632. The fine table contains a NULL. Which names does the NOT EXISTS query return? Analysis

```
CREATE TABLE student (id INT, name TEXT);
INSERT INTO student VALUES (1,'Ali'),(2,'Sara'),(3,'Omar');
CREATE TABLE fine (sid INT);
INSERT INTO fine VALUES (2),(NULL);
SELECT name FROM student s
WHERE NOT EXISTS (SELECT 1 FROM fine f WHERE f.sid = s.id)
ORDER BY name;
```

A. No rows
B. Ali, Sara, Omar
C. Ali, Omar
D. Sara

633. Ali placed orders worth 300 in total. Why does this query report something else, and what does it return? Analysis

```
CREATE TABLE orders (cust TEXT, amount INT);
INSERT INTO orders VALUES ('Ali',100),('Ali',200);
CREATE TABLE payments (cust TEXT, paid INT);
INSERT INTO payments VALUES ('Ali',50),('Ali',50),('Ali',100);
SELECT SUM(o.amount)
FROM orders o JOIN payments p ON o.cust = p.cust;
```

A. 300, because SUM ignores the payments table
B. 900, because each order row is repeated once for each of Ali's 3 payment rows
C. 600, because the join doubles each order
D. 1200, because amount and paid are both added

634. Two departments are in Lahore, so the subquery returns two rows. What happens in standard SQL (for example in PostgreSQL)? Analysis

```
CREATE TABLE dept (did INT, dname TEXT, city TEXT);
INSERT INTO dept VALUES (10,'IT','Lahore'),(20,'HR','Karachi'),(30,'Sales','Lahore'),
(40,'Legal','Quetta');
CREATE TABLE emp (eid INT, name TEXT, did INT, salary INT);
INSERT INTO emp VALUES
  (1,'Ali',10,90),(2,'Sara',10,60),(3,'Omar',20,50),
  (4,'Hina',20,70),(5,'Zain',30,40);
SELECT name FROM emp
WHERE did = (SELECT did FROM dept WHERE city = 'Lahore');
```

A. A run-time error: a subquery used with = returned more than one row
B. It returns Ali, Sara and Zain, as if = were IN
C. It returns no rows
D. It returns Ali and Sara, using the first value the subquery returns

635. Two sales share day 2. What running totals does the query show for the two day-2 rows? Analysis

```
CREATE TABLE sales (day INT, amt INT);
INSERT INTO sales VALUES (1,10),(2,20),(2,5),(3,15);
SELECT day, amt, SUM(amt) OVER (ORDER BY day) AS run
FROM sales
ORDER BY day, amt;
```

A. 35 for both rows
B. 30 for the first and 35 for the second
C. 15 for the first and 35 for the second
D. 25 for both rows

636. Legal has no employees. What values of a and b does the query show for Legal? Analysis

```
CREATE TABLE dept (did INT, dname TEXT, city TEXT);
INSERT INTO dept VALUES (10,'IT','Lahore'),(20,'HR','Karachi'),(30,'Sales','Lahore'),
(40,'Legal','Quetta');
CREATE TABLE emp (eid INT, name TEXT, did INT, salary INT);
INSERT INTO emp VALUES
  (1,'Ali',10,90),(2,'Sara',10,60),(3,'Omar',20,50),
  (4,'Hina',20,70),(5,'Zain',30,40);
SELECT d.dname, COUNT(*) AS a, COUNT(e.eid) AS b
FROM dept d LEFT JOIN emp e ON d.did = e.did
WHERE d.dname = 'Legal'
GROUP BY d.dname;
```

A. a = 0, b = 0
B. a = 1, b = 0
C. a = 1, b = 1
D. a = 0, b = 1

637. The query was meant to list employees who earn more than their own department's average (Sara earns 80). What does it actually return? Analysis

```
CREATE TABLE dept (did INT, dname TEXT, city TEXT);
INSERT INTO dept VALUES (10,'IT','Lahore'),(20,'HR','Karachi'),(30,'Sales','Lahore'),
(40,'Legal','Quetta');
CREATE TABLE emp (eid INT, name TEXT, did INT, salary INT);
INSERT INTO emp VALUES
  (1,'Ali',10,90),(2,'Sara',10,80),(3,'Omar',20,50),
  (4,'Hina',20,70),(5,'Zain',30,40);
SELECT e.name FROM emp e
WHERE e.salary > (SELECT AVG(x.salary) FROM emp x)
ORDER BY e.name;
```

A. Ali, Hina, Sara: everyone above the company-wide average of 66
B. Ali, Hina: everyone above their department average
C. No rows, because the subquery is not correlated
D. Ali only: the highest earner

638. Each INSERT runs as its own statement in autocommit mode. What does the final SELECT return? Analysis

```
CREATE TABLE pay (id INT, amount INT);
CREATE TRIGGER no_neg BEFORE INSERT ON pay
FOR EACH ROW WHEN NEW.amount < 0
BEGIN
  SELECT RAISE(ABORT, 'negative amount');
END;
INSERT INTO pay VALUES (1, 500);
INSERT INTO pay VALUES (2, -20);
INSERT INTO pay VALUES (3, 300);
SELECT COUNT(*), SUM(amount) FROM pay;
```

A. 2 and 800
B. 3 and 780
C. 1 and 500
D. 0 and NULL

639. Which of these views would standard SQL most likely allow you to UPDATE directly? Analysis

A. SELECT id, name, price FROM product WHERE price < 100
B. SELECT DISTINCT category FROM product
C. SELECT category, AVG(price) FROM product GROUP BY category
D. SELECT p.category, COUNT(*) FROM product p JOIN sale s ON p.id = s.pid GROUP BY p.category

640. How many rows does the FULL OUTER JOIN return? Analysis

```
CREATE TABLE t1 (k INT);
INSERT INTO t1 VALUES (1),(2),(3);
CREATE TABLE t2 (k INT);
INSERT INTO t2 VALUES (2),(3),(4),(5);
SELECT COUNT(*) FROM t1 FULL OUTER JOIN t2 ON t1.k = t2.k;
```

A. 5
B. 2
C. 7
D. 12

641. What does INTERSECT return when it combines two SELECT results? Understanding

A. The distinct rows that appear in both results
B. Every row from either result, with duplicates kept
C. The rows of the first result that are missing from the second
D. Every pairing of a row from the first result with a row from the second

642. Two SELECT statements are combined with UNION. Which condition must they meet? Understanding

A. Their columns must have identical names in the same order
B. They must return the same number of columns, with compatible data types in matching positions
C. They must read from the same base table
D. Each must contain its own ORDER BY clause

643. A clerk must see staff names and departments but never salaries. How can a view help? Understanding

A. Create a materialised view of the whole table, which hides columns automatically
B. Add a trigger that deletes salaries whenever the clerk reads the table
C. Create a view that selects only the name and department columns, and grant the clerk access to the view but not the base table
D. Create an index on the name column so the salary column cannot be scanned

644. What is an INSTEAD OF trigger mainly used for? Understanding

A. Running code after a statement has already changed a table
B. Replacing a stored procedure that has been dropped
C. Undoing the last transaction when a constraint fails
D. Making a view accept INSERT, UPDATE or DELETE by running custom code against the base tables in place of the original statement

645. Which is a key advantage of putting a sequence of SQL statements in a stored procedure? Understanding

A. The client sends one CALL instead of many statements, which cuts network round trips and lets the DBMS reuse the procedure's logic
B. Stored procedures run automatically whenever a table is modified
C. Stored procedures bypass all access-control checks
D. Stored procedures make the statements inside them run outside any transaction

646. In a stored procedure, what is an OUT parameter for? Understanding

A. Receiving a read-only value from the caller
B. Returning a value from the procedure to the caller
C. Naming the table the procedure writes its output to
D. Marking the parameter as optional

647. Which window function gives every row a distinct sequential number, even when the ORDER BY values tie? Understanding

A. RANK()
B. DENSE_RANK()
C. ROW_NUMBER()
D. NTILE(1)

648. What does PARTITION BY do inside an OVER clause? Understanding

A. It collapses each group into one output row, like GROUP BY
B. It physically splits the table across disks
C. It filters out rows that do not belong to the first partition
D. It splits the rows into groups so the window function restarts for each group, while every row stays in the result

649. What does LAG(amount) OVER (ORDER BY day) return for each row? Understanding

A. The amount from the previous row in day order, or NULL for the first row
B. The amount from the next row in day order
C. The running total of amount up to the current row
D. The smallest amount seen so far

650. What is a common table expression (CTE)? Understanding

A. A table stored permanently and shared by every session
B. A named temporary result defined with WITH that the following statement can refer to like a table
C. An index the optimiser builds for a single query
D. A trigger that runs once per statement

651. You need customers who have placed at least one order, each listed once. Which construct fits best? Understanding

A. An INNER JOIN of customer and orders with no DISTINCT
B. A CROSS JOIN of customer and orders
C. WHERE EXISTS (SELECT 1 FROM orders o WHERE o.cust_id = c.id)
D. WHERE NOT EXISTS (SELECT 1 FROM orders o WHERE o.cust_id = c.id)

652. Where may ORDER BY appear in a query that combines SELECTs with UNION? Understanding

A. In every SELECT, and each one sorts its own part of the output
B. Only in the first SELECT
C. Nowhere, because UNION results cannot be sorted
D. Once, after the last SELECT, and it sorts the whole combined result

653. An UPDATE statement changes 50 rows of a table that has one row-level and one statement-level AFTER UPDATE trigger. How many times does each fire? Understanding

A. Row-level 50 times, statement-level once
B. Both fire 50 times
C. Both fire once
D. Row-level once, statement-level 50 times

654. Why would a designer choose a BEFORE INSERT row trigger rather than an AFTER INSERT one? Understanding

A. Because only BEFORE triggers can write to another table
B. To check or adjust the NEW values before the row is written, for example to reject bad data or fill a default
C. Because AFTER triggers cannot see the inserted values
D. To make the insert run outside its transaction

655. How does a user-defined function usually differ from a stored procedure? Understanding

A. A function runs automatically on table events; a procedure does not
B. A procedure can take parameters; a function cannot
C. A function returns a value and can be called inside a SELECT expression; a procedure is invoked with CALL and returns results through OUT parameters or result sets
D. A function is stored on the client; a procedure is stored on the server

656. A recursive CTE has two parts joined by UNION ALL. When does the recursion stop? Understanding

A. After exactly ten iterations, the SQL standard limit
B. As soon as the anchor member returns its first row
C. When the outer query applies ORDER BY
D. When the recursive member produces no new rows in an iteration

657. View v2 is defined on view v1, and v2 has WITH CASCADED CHECK OPTION. What is checked when a row is inserted through v2? Understanding

A. The row must satisfy the WHERE conditions of v2 and of every view it is built on, including v1
B. Only v2's own WHERE condition is checked
C. Only v1's WHERE condition is checked
D. No condition is checked, because CASCADED only affects deletes

658. What is the main drawback of a materialised view compared with an ordinary view? Understanding

A. Every read recomputes the full defining query
B. Its stored result can become stale until it is refreshed, and it costs storage and refresh time
C. It can never be indexed
D. It cannot contain joins or aggregates

659. Why does x NOT IN (subquery) return no rows when the subquery result contains a NULL? Understanding

A. Because NULL values are converted to zero before comparison
B. Because NOT IN raises an error that the DBMS silently suppresses
C. NOT IN means x <> v for every v, and x <> NULL is UNKNOWN, so the whole predicate is never TRUE
D. Because the subquery is skipped when it contains NULL

660. Which predicate is equivalent to x IN (SELECT y FROM t)? Understanding

A. x = ALL (SELECT y FROM t)
B. x <> ANY (SELECT y FROM t)
C. EXISTS (SELECT y FROM t)
D. x = ANY (SELECT y FROM t)

661. Which predicate is equivalent to x NOT IN (SELECT y FROM t)? Understanding

A. x <> ALL (SELECT y FROM t)
B. x <> ANY (SELECT y FROM t)
C. NOT EXISTS (SELECT y FROM t)
D. x = ALL (SELECT y FROM t)

662. In SUM(amt) OVER (ORDER BY day ...), which frame clause gives each of several rows with the same day its own running total, instead of one shared total? Understanding

A. RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
B. ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
C. No frame clause, because the default already counts rows one by one
D. RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING

663. Trigger T1 on orders updates the stock table, and trigger T2 on stock inserts into reorder_log. What happens when one row is inserted into orders? Understanding

A. Only T1 fires, because a trigger can never cause another trigger to fire
B. T2 fires first, because it was created later
C. T1 fires, its UPDATE on stock fires T2, and T2 writes to reorder_log, all within the same transaction
D. The insert fails, because triggers on different tables cannot interact

664. A correlated subquery is logically re-run once per outer row. Why can a modern DBMS still execute it quickly on large tables? Understanding

A. The DBMS caches the subquery result once and ignores the outer row's values
B. Correlated subqueries are always rewritten as CROSS JOINs, which are the fastest join
C. The SQL standard limits correlated subqueries to ten outer rows
D. The optimiser can unnest (decorrelate) it into an equivalent join or semi-join and choose an efficient plan

665. What does the INTERSECT query return? Application

```
CREATE TABLE chess (name TEXT);
INSERT INTO chess VALUES ('Ayesha'),('Bilal'),('Hina');
CREATE TABLE debate (name TEXT);
INSERT INTO debate VALUES ('Bilal'),('Hina'),('Hina'),('Usman');
SELECT COUNT(*) FROM (
  SELECT name FROM chess INTERSECT SELECT name FROM debate
);
```

A. 2
B. 1
C. 3
D. 4

666. What value does ROW_NUMBER give Danish? Application

```
CREATE TABLE staff (name TEXT, dept TEXT, salary INT);
INSERT INTO staff VALUES ('Asad','Sales',90),('Sana','Sales',50),
  ('Kamran','HR',80),('Danish','Sales',70);
SELECT name, ROW_NUMBER() OVER (PARTITION BY dept ORDER BY salary DESC) AS rn
FROM staff ORDER BY name;
```

A. 1
B. 2
C. 3
D. 4

667. What does the query return for Usman? Application

```
CREATE TABLE cust (id INT, name TEXT);
INSERT INTO cust VALUES (1,'Ayesha'),(2,'Usman');
CREATE TABLE orders (cid INT, amount INT);
INSERT INTO orders VALUES (1,300);
SELECT c.name, COALESCE(o.amount, 0)
FROM cust c LEFT JOIN orders o ON o.cid = c.id
WHERE c.name = 'Usman';
```

A. NULL
B. Usman is not returned
C. 0
D. 1

668. What does the CASE query return? Application

```
CREATE TABLE result (marks INT);
INSERT INTO result VALUES (85),(45),(60),(30);
SELECT CASE WHEN marks >= 60 THEN 'High' ELSE 'Low' END AS band, COUNT(*)
FROM result GROUP BY band ORDER BY band;
```

A. High 3, Low 1
B. High 1, Low 3
C. High 2, Low 1
D. High 2, Low 2

669. What does the query using a CTE return? Application

```
CREATE TABLE sale (city TEXT, amt INT);
INSERT INTO sale VALUES ('Lahore',100),('Karachi',120),('Lahore',50),('Quetta',40);
WITH t AS (SELECT city, SUM(amt) AS total FROM sale GROUP BY city)
SELECT city FROM t WHERE total > 130;
```

A. Lahore
B. Karachi
C. Lahore and Karachi
D. Quetta

670. What values does the change column show, in month order? Application

```
CREATE TABLE monthly (m INT, rev INT);
INSERT INTO monthly VALUES (1,100),(2,120),(3,110);
SELECT m, rev - LAG(rev) OVER (ORDER BY m) AS change
FROM monthly ORDER BY m;
```

A. 0, 20, -10
B. NULL, 20, -10
C. 20, -10, NULL
D. NULL, 120, 110

671. Which names does the top-earner-per-department query return? Application

```
CREATE TABLE staff (name TEXT, dept TEXT, salary INT);
INSERT INTO staff VALUES ('Asad','Sales',90),('Sana','Sales',50),
  ('Kamran','HR',80),('Danish','Sales',70),('Iqra','HR',60);
WITH r AS (
  SELECT name, ROW_NUMBER() OVER (PARTITION BY dept ORDER BY salary DESC) AS rn
  FROM staff)
SELECT name FROM r WHERE rn = 1 ORDER BY name;
```

A. Asad only
B. Asad, Kamran and Danish
C. Asad and Kamran
D. Kamran and Sana

672. Which customers does the query return? Application

```
CREATE TABLE cust (id INT, name TEXT);
INSERT INTO cust VALUES (1,'Ayesha'),(2,'Bilal'),(3,'Zara');
CREATE TABLE orders (cid INT);
INSERT INTO orders VALUES (1),(1),(2),(3),(3),(3);
SELECT name FROM cust WHERE id IN (
  SELECT cid FROM orders GROUP BY cid HAVING COUNT(*) >= 2
) ORDER BY name;
```

A. Ayesha only
B. Ayesha, Bilal and Zara
C. Bilal only
D. Ayesha and Zara

673. What does the query with a derived table return? Application

```
CREATE TABLE emp (dept TEXT, salary INT);
INSERT INTO emp VALUES ('IT',100),('IT',50),('HR',40),('HR',50),('Ops',60);
SELECT AVG(total) FROM (
  SELECT dept, SUM(salary) AS total FROM emp GROUP BY dept
) AS t;
```

A. 100
B. 75
C. 60
D. 300

674. How many rows does the RIGHT JOIN return? Application

```
CREATE TABLE enrol (sid INT, code TEXT);
INSERT INTO enrol VALUES (1,'DB'),(2,'DB'),(1,'OS'),(3,'ML');
CREATE TABLE course (code TEXT);
INSERT INTO course VALUES ('DB'),('OS'),('AI');
SELECT COUNT(*) FROM enrol e RIGHT JOIN course c ON e.code = c.code;
```

A. 3
B. 4
C. 2
D. 5

675. The DELETE removes every row with qty = 0. What does the final SELECT return? Application

```
CREATE TABLE item (id INT, qty INT);
INSERT INTO item VALUES (1,0),(2,5),(3,0),(4,0);
CREATE TABLE del_log (id INT);
CREATE TRIGGER log_del AFTER DELETE ON item FOR EACH ROW
BEGIN
  INSERT INTO del_log VALUES (OLD.id);
END;
DELETE FROM item WHERE qty = 0;
SELECT COUNT(*) FROM del_log;
```

A. 1
B. 0
C. 3
D. 4

676. View v2 is built on view v1. After the UPDATE on the base table, what does the final SELECT return? Application

```
CREATE TABLE product (name TEXT, price INT, stock INT);
INSERT INTO product VALUES ('Pen',20,0),('Mouse',120,8),('Desk',900,3);
CREATE VIEW v1 AS SELECT * FROM product WHERE price < 200;
CREATE VIEW v2 AS SELECT name FROM v1 WHERE stock > 0;
UPDATE product SET price = 150 WHERE name = 'Pen';
SELECT name FROM v2;
```

A. Mouse and Pen
B. Pen
C. Nothing: the view is empty
D. Mouse

677. The recursive query walks the reporting chain below employee 1. What does it return? Application

```
CREATE TABLE emp (id INT, mgr INT);
INSERT INTO emp VALUES (1,NULL),(2,1),(3,1),(4,2),(5,4),(6,7),(7,NULL);
WITH RECURSIVE sub(id) AS (
  SELECT id FROM emp WHERE mgr = 1
  UNION ALL
  SELECT e.id FROM emp e JOIN sub s ON e.mgr = s.id
)
SELECT COUNT(*) FROM sub;
```

A. 4
B. 5
C. 3
D. 2

678. Five rows are split with NTILE(2). How many rows land in each bucket? Application

```
CREATE TABLE t (v INT);
INSERT INTO t VALUES (10),(20),(30),(40),(50);
SELECT b, COUNT(*) FROM (
  SELECT NTILE(2) OVER (ORDER BY v) AS b FROM t
) GROUP BY b ORDER BY b;
```

A. Bucket 1: 2 rows, bucket 2: 3 rows
B. Bucket 1: 3 rows, bucket 2: 2 rows
C. Both buckets get 2 rows and one row gets NULL
D. Bucket 1: 5 rows, bucket 2: 0 rows

679. What does the correlated subquery in the SELECT list show for Bilal? Application

```
CREATE TABLE cust (id INT, name TEXT);
INSERT INTO cust VALUES (1,'Ayesha'),(2,'Bilal');
CREATE TABLE orders (cid INT);
INSERT INTO orders VALUES (1),(1);
SELECT name, (SELECT COUNT(*) FROM orders o WHERE o.cid = c.id) AS n
FROM cust c ORDER BY id;
```

A. NULL
B. 1
C. 0
D. Bilal is not returned

680. What does the next_score column show, in order of roll number? Application

```
CREATE TABLE exam (roll INT, score INT);
INSERT INTO exam VALUES (1,50),(2,70),(3,90);
SELECT roll, LEAD(score) OVER (ORDER BY roll) AS next_score
FROM exam ORDER BY roll;
```

A. NULL, 50, 70
B. 50, 70, 90
C. 70, 90, 0
D. 70, 90, NULL

681. What does the recursive CTE return? Application

```
WITH RECURSIVE fact(n, f) AS (
  SELECT 1, 1
  UNION ALL
  SELECT n + 1, f * (n + 1) FROM fact WHERE n < 5
)
SELECT MAX(f) FROM fact;
```

A. 120
B. 24
C. 720
D. 15

682. What moving-average values does the query show, in day order? Application

```
CREATE TABLE d (day INT, v INT);
INSERT INTO d VALUES (1,10),(2,30),(3,50),(4,50);
SELECT day, AVG(v) OVER (ORDER BY day
  ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) AS ma
FROM d ORDER BY day;
```

A. 10, 15, 26.67, 35
B. 10, 20, 40, 50
C. 10, 20, 30, 40
D. 15, 30, 45, 50

683. The view joins two tables, so it is not directly updatable. The script uses SQLite's trigger syntax. What does the final SELECT return? Application

```
CREATE TABLE dept (did INT, dname TEXT);
CREATE TABLE emp (name TEXT, did INT);
INSERT INTO dept VALUES (1,'IT');
INSERT INTO emp VALUES ('Ali',1);
CREATE VIEW staff AS
  SELECT e.name, d.did, d.dname FROM emp e JOIN dept d ON e.did = d.did;
CREATE TRIGGER staff_ins INSTEAD OF INSERT ON staff
BEGIN
  INSERT INTO dept VALUES (NEW.did, NEW.dname);
  INSERT INTO emp VALUES (NEW.name, NEW.did);
END;
INSERT INTO staff VALUES ('Sana', 2, 'HR');
SELECT (SELECT COUNT(*) FROM dept), (SELECT COUNT(*) FROM emp);
```

A. 1 and 1
B. 2 and 1
C. 2 and 2
D. An error: inserts into a join view are never allowed

684. The query uses a double NOT EXISTS. Which students does it return? Application

```
CREATE TABLE course (code TEXT);
INSERT INTO course VALUES ('DB'),('OS');
CREATE TABLE took (name TEXT, code TEXT);
INSERT INTO took VALUES ('Ayesha','DB'),('Ayesha','OS'),('Bilal','DB'),
  ('Hina','OS'),('Hina','AI');
SELECT DISTINCT t.name FROM took t
WHERE NOT EXISTS (
  SELECT * FROM course c WHERE NOT EXISTS (
    SELECT * FROM took t2 WHERE t2.name = t.name AND t2.code = c.code));
```

A. Ayesha and Bilal
B. Bilal and Hina
C. Hina
D. Ayesha

685. Two triggers cascade. The script uses SQLite's trigger syntax. What does the final SELECT return? Application

```
CREATE TABLE stock (item INT, qty INT);
INSERT INTO stock VALUES (1, 10);
CREATE TABLE orders (item INT, qty INT);
CREATE TABLE reorder (item INT);
CREATE TRIGGER t1 AFTER INSERT ON orders FOR EACH ROW BEGIN
  UPDATE stock SET qty = qty - NEW.qty WHERE item = NEW.item;
END;
CREATE TRIGGER t2 AFTER UPDATE ON stock FOR EACH ROW WHEN NEW.qty < 5 BEGIN
  INSERT INTO reorder VALUES (NEW.item);
END;
INSERT INTO orders VALUES (1, 7);
SELECT (SELECT qty FROM stock), (SELECT COUNT(*) FROM reorder);
```

A. 3 and 1
B. 3 and 0
C. 10 and 1
D. 3 and 2

686. A student writes this query to list employees who earn more than the average. Why does it fail, and what is the fix? Analysis

```
SELECT name FROM emp WHERE salary > AVG(salary);
```

A. AVG needs a GROUP BY name clause to work in WHERE
B. Aggregates are not allowed in WHERE; compare with a subquery: WHERE salary > (SELECT AVG(salary) FROM emp)
C. WHERE must be replaced by HAVING with no other change
D. The query is valid and returns every employee

687. The query should list departments that have no employees. What does it return, and why? Analysis

```
CREATE TABLE dept (did INT, dname TEXT);
INSERT INTO dept VALUES (10,'IT'),(20,'HR'),(30,'Legal');
CREATE TABLE emp (name TEXT, did INT);
INSERT INTO emp VALUES ('Ali',10),('Sana',NULL);
SELECT dname FROM dept WHERE did NOT IN (SELECT did FROM emp);
```

A. HR and Legal, the two departments without employees
B. Legal only, because NULL matches HR
C. No rows, because emp.did contains a NULL and NOT IN then can never be TRUE
D. An error, because the subquery returns more than one row

688. A developer filters on a window function in the WHERE clause. What happens, and what is the fix? Analysis

```
SELECT name, RANK() OVER (ORDER BY marks DESC) AS r
FROM score
WHERE r = 1;
```

A. It works and returns the top-ranked row
B. It fails, and the fix is to move the condition into GROUP BY
C. It works, but returns every row because WHERE ignores aliases
D. It fails, because window functions are computed after WHERE; compute the rank in a CTE or derived table and filter outside it

689. Both tables contain a NULL key. How many rows does the inner join return? Analysis

```
CREATE TABLE a (k INT);
INSERT INTO a VALUES (1),(2),(NULL);
CREATE TABLE b (k INT);
INSERT INTO b VALUES (2),(3),(NULL);
SELECT COUNT(*) FROM a JOIN b ON a.k = b.k;
```

A. 1
B. 2
C. 3
D. 4

690. The analyst wanted every department, with its employees where they exist. How many rows does the query return, and why? Analysis

```
CREATE TABLE dept (did INT, dname TEXT);
INSERT INTO dept VALUES (1,'IT'),(2,'Legal');
CREATE TABLE emp (eid INT, name TEXT, did INT);
INSERT INTO emp VALUES (1,'Ali',1),(2,'Sana',1);
CREATE TABLE project (eid INT, pname TEXT);
INSERT INTO project VALUES (1,'ERP'),(2,'CRM');
SELECT COUNT(*) FROM dept d
  LEFT JOIN emp e ON e.did = d.did
  JOIN project p ON p.eid = e.eid;
```

A. 3, because the LEFT JOIN keeps Legal
B. 2, because the second inner join drops the NULL-padded row for Legal
C. 1, because only IT has a project
D. 4, because the joins multiply rows

691. The EXISTS subquery was meant to find customers with an order over 500. What does it return, and why? Analysis

```
CREATE TABLE cust (id INT, name TEXT);
INSERT INTO cust VALUES (1,'Ayesha'),(2,'Bilal'),(3,'Zara');
CREATE TABLE orders (cid INT, amount INT);
INSERT INTO orders VALUES (1,800),(2,200);
SELECT name FROM cust c
WHERE EXISTS (SELECT 1 FROM orders o WHERE o.amount > 500);
```

A. Ayesha only
B. No customers, because EXISTS needs a column list
C. All three customers, because the subquery is not correlated and is TRUE for every row
D. Ayesha and Bilal

692. The developer expected counter = 1 after one UPDATE statement. The script uses SQLite, where triggers are row-level. What is the value, and why? Analysis

```
CREATE TABLE p (id INT, price INT);
INSERT INTO p VALUES (1,10),(2,20),(3,30),(4,500);
CREATE TABLE audit (counter INT);
INSERT INTO audit VALUES (0);
CREATE TRIGGER t AFTER UPDATE ON p FOR EACH ROW BEGIN
  UPDATE audit SET counter = counter + 1;
END;
UPDATE p SET price = price + 1 WHERE price < 100;
SELECT counter FROM audit;
```

A. 1, because one statement fires the trigger once
B. 0, because AFTER triggers do not see UPDATE statements
C. 4, because the trigger also fires for the unchanged row
D. 3, because the trigger fires once for each of the three updated rows

693. No employee is named 'Nobody'. What does the query return, and why? Analysis

```
CREATE TABLE emp (name TEXT, salary INT);
INSERT INTO emp VALUES ('Ali',50),('Sana',70);
SELECT name FROM emp
WHERE salary > (SELECT salary FROM emp WHERE name = 'Nobody');
```

A. No rows, because the scalar subquery yields NULL and salary > NULL is UNKNOWN
B. Every employee, because a missing comparison value counts as zero
C. An error, because the subquery returns no rows
D. Only the highest-paid employee

694. The query was meant to show, on every row, the largest marks value in the table. What does the last column show for the row with marks 70? Analysis

```
CREATE TABLE s (name TEXT, marks INT);
INSERT INTO s VALUES ('A',50),('B',70),('C',90);
SELECT name, marks, LAST_VALUE(marks) OVER (ORDER BY marks) AS top
FROM s ORDER BY marks;
```

A. 90, the largest value in the table
B. 70, because with ORDER BY the default frame ends at the current row and its peers
C. NULL, because LAST_VALUE needs a PARTITION BY
D. 50, the first value in the window

695. Two Sales staff tie on the top salary. How many rows does each query return, in order? Analysis

```
CREATE TABLE staff (name TEXT, dept TEXT, salary INT);
INSERT INTO staff VALUES ('Asad','Sales',90),('Bina','Sales',90),
  ('Sana','Sales',50),('Kamran','HR',80);
SELECT COUNT(*) FROM (SELECT ROW_NUMBER() OVER
  (PARTITION BY dept ORDER BY salary DESC) AS r FROM staff) WHERE r = 1;
SELECT COUNT(*) FROM (SELECT RANK() OVER
  (PARTITION BY dept ORDER BY salary DESC) AS r FROM staff) WHERE r = 1;
```

A. 2 and 2
B. 3 and 3
C. 2 and 3
D. 3 and 2

696. Table b stores its columns in the order (city, name). Which row does the query produce for Sana? Analysis

```
CREATE TABLE a (name TEXT, city TEXT);
INSERT INTO a VALUES ('Ali','Lahore');
CREATE TABLE b (city TEXT, name TEXT);
INSERT INTO b VALUES ('Karachi','Sana');
SELECT name, city FROM a
UNION ALL
SELECT * FROM b;
```

A. name = 'Sana', city = 'Karachi'
B. No row: the query fails because the column names of the two SELECTs differ
C. No row: UNION ALL skips rows whose columns are in a different order
D. name = 'Karachi', city = 'Sana'

697. The query should list the top earner of each department. What does it return, and why? Analysis

```
CREATE TABLE staff (name TEXT, dept TEXT, salary INT);
INSERT INTO staff VALUES ('Asad','Sales',90),('Bina','Sales',90),
  ('Sana','Sales',50),('Kamran','HR',80);
SELECT name FROM staff s
WHERE salary = (SELECT MAX(salary) FROM staff t WHERE t.dept = s.dept)
ORDER BY name;
```

A. Asad, Bina and Kamran, because the correlated MAX returns ties as well
B. Asad and Kamran, one per department
C. Kamran only, because the subquery uses the company-wide maximum
D. An error, because the subquery returns more than one row

698. There are two courses, and Bilal's DB enrolment was recorded twice. Which names does the 'took every course' query return, and how should it be fixed? Analysis

```
CREATE TABLE course (code TEXT);
INSERT INTO course VALUES ('DB'),('OS');
CREATE TABLE took (name TEXT, code TEXT);
INSERT INTO took VALUES ('Ayesha','DB'),('Ayesha','OS'),
  ('Bilal','DB'),('Bilal','DB'),('Hina','OS');
SELECT name FROM took GROUP BY name
HAVING COUNT(*) = (SELECT COUNT(*) FROM course) ORDER BY name;
```

A. Ayesha only; the query is already correct
B. Ayesha and Bilal; the fix is HAVING COUNT(DISTINCT code) = (SELECT COUNT(*) FROM course)
C. Bilal only; the fix is to add DISTINCT to the outer SELECT
D. Nobody; the fix is to use COUNT(code) instead of COUNT(*)

699. Zara has no orders. What do the two queries return for her, in order, and what is the lesson? Analysis

```
CREATE TABLE orders (cust TEXT, amount INT);
INSERT INTO orders VALUES ('Ayesha',300);
SELECT SUM(amount) FROM orders WHERE cust = 'Zara';
SELECT COALESCE(SUM(amount), 0) FROM orders WHERE cust = 'Zara';
```

A. 0 and 0: SUM over no rows is 0
B. NULL and NULL: COALESCE cannot change an aggregate result
C. NULL and 0: SUM over no rows is NULL, so wrap it in COALESCE when 0 is wanted
D. 0 and NULL: COALESCE turns 0 into NULL

700. The flight graph contains a cycle (Lahore, Karachi, Quetta, Lahore). What does the query return, and why does it finish? Analysis

```
CREATE TABLE flight (src TEXT, dst TEXT);
INSERT INTO flight VALUES ('Lahore','Karachi'),('Karachi','Quetta'),
  ('Quetta','Lahore');
WITH RECURSIVE reach(city) AS (
  SELECT 'Lahore'
  UNION
  SELECT f.dst FROM flight f JOIN reach r ON f.src = r.city
)
SELECT COUNT(*) FROM reach;
```

A. It never finishes, because the cycle repeats forever
B. 4, because Lahore is counted twice
C. 2, because the anchor city is not included
D. 3, because UNION discards rows already produced, so the recursion stops once no new city appears

Database Design & Normalization MCQs for NSCT

701. What does the functional dependency X → Y mean for a relation R? Understanding
A. Any two tuples that agree on Y must also agree on X
B. X and Y must both be candidate keys
C. Any two tuples of R that agree on X must also agree on Y
D. Every value of X appears exactly once in R

702. A relation is in First Normal Form (1NF) when: Understanding
A. It has no partial dependencies on the primary key
B. Every determinant is a candidate key
C. Every attribute holds a single atomic value, with no repeating groups or multi-valued cells
D. It has no transitive dependencies

703. Which problem does Second Normal Form (2NF) remove? Understanding
A. Non-prime attributes that depend on only part of a composite candidate key
B. Cells that hold more than one value
C. Non-prime attributes that depend on another non-prime attribute
D. Multivalued dependencies between independent attributes

704. Third Normal Form (3NF) mainly removes which kind of dependency? Understanding
A. Partial dependencies on a composite key
B. Transitive dependencies of non-prime attributes on a key
C. Multivalued dependencies
D. Join dependencies

705. A relation R is in Boyce-Codd Normal Form (BCNF) if, for every non-trivial functional dependency X → Y that holds in R: Understanding
A. X is a superkey of R
B. X is a superkey of R or Y is a prime attribute
C. Y is part of a candidate key
D. X is a single attribute

706. An attribute is called prime when it: Understanding

A. Belongs to the primary key chosen by the designer, and to no other key
B. Is functionally determined by every other attribute
C. Has no NULL values
D. Belongs to at least one candidate key

707. A table repeats each instructor's phone number in every course row. A clerk changes the number in one row and forgets the others. Which anomaly is this? Understanding

A. Insertion anomaly
B. Deletion anomaly
C. Update anomaly
D. Phantom anomaly

708. Which of these is NOT one of Armstrong's three basic axioms? Understanding

A. Reflexivity: if $Y \subseteq X$, then $X \rightarrow Y$
B. Augmentation: if $X \rightarrow Y$, then $XZ \rightarrow YZ$
C. Transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
D. Union: if $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

709. R is decomposed into R1 and R2. Under which condition is the decomposition guaranteed lossless? Understanding

A. The common attributes $R1 \cap R2$ functionally determine all of R1 or all of R2
B. R1 and R2 have the same number of attributes
C. $R1 \cup R2 = R$, whatever the common attributes are
D. Each FD of R holds entirely inside R1 or inside R2

710. R is in 1NF and every candidate key of R has exactly one attribute. What can you conclude? Understanding

A. R is in 3NF
B. R is in BCNF
C. Nothing beyond 1NF
D. R is in 2NF

711. Which statement about decomposing a relation is correct? Understanding

A. A lossless, dependency-preserving BCNF decomposition always exists
B. Any 3NF decomposition is automatically in BCNF
C. A lossless, dependency-preserving 3NF decomposition always exists; for BCNF that is not always possible
D. Decomposing into BCNF always loses information

712. Which functional dependency is trivial? Understanding

A. $A \rightarrow B$
B. $AB \rightarrow C$
C. $A \rightarrow AB$
D. $AB \rightarrow A$

713. Which property is NOT required of a minimal (canonical) cover? Understanding

A. Every left-hand side is a candidate key
B. It is equivalent to the original set F
C. No left-hand side has an extraneous attribute
D. No dependency can be removed without changing the closure

714. Why might a designer deliberately denormalise a table? Understanding

A. To remove update anomalies
B. To make the schema satisfy BCNF
C. To speed up frequent read queries by avoiding joins, at the cost of redundancy
D. To reduce the storage the database needs

715. Teacher(Name, Subject, Hobby) records every subject and every hobby of a teacher, and subjects and hobbies are independent. Which normal form deals with this redundancy? Understanding

A. Third Normal Form (3NF), through transitive dependencies
B. BCNF, through non-key determinants
C. Second Normal Form (2NF), through partial dependencies
D. Fourth Normal Form (4NF), through multivalued dependencies

716. Which statement is always true of a relation with exactly two attributes? Understanding

A. It is in BCNF
B. It is in 3NF but may not be in BCNF
C. It is in 2NF but may not be in 3NF
D. Its only candidate key is both attributes together

717. R(A, B, C) has F = {A → B, B → C}. What is the closure A^+? Application

A. {A, B}
B. {A}
C. {B, C}
D. {A, B, C}

718. R(A, B, C, D) has F = {A → B, B → C, C → D}. What is the candidate key? Application

A. AB
B. D
C. A
D. ABCD

719. R(A, B, C, D) has F = {AB → C, C → D, D → A}. What are all the candidate keys? Application

A. AB only
B. AB, BC and BD
C. AB and CD
D. AB and BC

720. R(A, B, C) has F = {AB → C, C → B}. What is the highest normal form of R? Application

A. BCNF
B. 3NF
C. 2NF
D. 1NF

721. R(A, B, C, D) has F = {A → C, AB → D}. What is the highest normal form of R? Application

A. 2NF
B. 3NF
C. BCNF
D. 1NF

722. Student(StudentID, Name, DeptID, DeptName) has StudentID → Name, DeptID and DeptID → DeptName. What is its highest normal form? Application

A. 2NF
B. 1NF
C. 3NF
D. BCNF

723. Given F = {A → B, B → C}, which dependency is implied by F? Application

A. C → A
B. A → C
C. B → A
D. C → B

724. R(A, B, C, D, E) has F = {A → B, BC → D, D → E, E → A}. What is $(BC)^+$? Application

A. {A, B, C, D, E}
B. {B, C, D}
C. {B, C, D, E}
D. {B, C}

725. R(A, B, C, D) has A as its only candidate key. How many superkeys does R have? Application

A. 1
B. 4
C. 8
D. 16

726. R(A, B, C) has the single FD A → B. Which decomposition is lossless? Application

A. (A, B) and (B, C)
B. (A, B) and (A, C)
C. (A, C) and (B, C)
D. (A) and (B, C)

727. R(A, B, C, D, E) has F = {A → B, BC → E, ED → A}. What are the candidate keys? Application

A. ACD only
B. CD
C. ACD and BCD
D. ACD, BCD and CDE

728. R(A, B, C, D) has F = {AB → CD}, and AB is its only candidate key. Which attributes are non-prime? Application

A. C and D
B. A and B
C. A, B, C and D
D. D only

729. Which set is a minimal cover of F = {A → BC, B → C, A → B, AB → C}? Application

A. {A → B, A → C, B → C}
B. {A → B, B → C}
C. {A → BC}
D. {A → B, AB → C}

730. Student(ID, Name, Phones) stores values such as '0300-1111, 0321-2222' in Phones. What is the standard 1NF fix? Application

A. Add columns Phone1, Phone2 and Phone3 to Student
B. Move phones into StudentPhone(ID, Phone) with key (ID, Phone)
C. Store the phone list as a single longer text value
D. Make Phones the primary key of Student

731. R(A, B, C) has F = {AB → C, C → B}. A designer decomposes it into R1(C, B) and R2(A, C) to reach BCNF. Which statement is correct? Analysis

A. The decomposition preserves every FD but the join is lossy
B. The decomposition is lossless and preserves every FD
C. R2(A, C) is not in BCNF, so the decomposition fails
D. Both parts are in BCNF and the join is lossless, but AB → C is no longer preserved

732. R(A, B, C, D, E) has F = {A → B, B → C, C → D, D → E}. What is the highest normal form of R? Analysis

A. 1NF
B. 2NF
C. 3NF
D. BCNF

733. Course(CourseID, Title, Instructor, InstructorPhone) stores each instructor's phone in their course rows. Dr Saad teaches only CS-401. If CS-401 is deleted, his phone number disappears from the database. What is the cause and the fix? Analysis

A. An insertion anomaly caused by the composite key; add a surrogate key
B. An update anomaly caused by CourseID → Title; split Title into its own table
C. A deletion anomaly caused by Instructor → InstructorPhone; move instructors to their own table
D. A lost update caused by concurrent transactions; use locking

734. R(A, B, C, D) has F = {A → B, C → D}. Which decomposition is lossless with every relation in BCNF? Analysis

A. (A, B), (C, D)
B. (A, B), (C, D), (A, C)
C. (A, B, C), (C, D)
D. (A, B), (B, C, D)

735. R(A, B, C, D) has F = {AB → C, C → D, D → A}. What is the highest normal form of R? Analysis

A. 3NF
B. BCNF
C. 2NF
D. 1NF

736. Rows of Timetable(sid, course, teacher, day): (S1, DB, Saad, Mon), (S2, DB, Saad, Mon), (S1, OS, Amna, Tue), (S3, AI, Saad, Wed). Which FD is violated by this data? Analysis

A. course → teacher
B. course → day
C. sid, course → teacher
D. teacher → course

737. Which set of functional dependencies is equivalent to F = {A → B, B → C}? Analysis

A. {A → B, A → C}
B. {A → BC}
C. {A → B, B → C, A → C}
D. {B → C, A → C}

738. R(A, B, C, D) has F = {AB → C, C → D}. A student says R is in 1NF only, 'because D does not depend on the whole key'. What is the correct highest normal form? Analysis

A. 2NF
B. 1NF
C. 3NF
D. BCNF

739. R(A, B, C, D) has F = {A → B, B → A, C → D, D → C}. How many candidate keys does R have? Analysis

A. 1
B. 4
C. 2
D. 6

740. R(A, B, C, D) has F = {A → B, B → C}. Which result does the 3NF synthesis algorithm produce? Analysis

A. (A, B), (B, C)
B. (A, B, C), (A, D)
C. (A, B), (B, C), (A, D)
D. (A, B, C, D), left unchanged

741. Employee(EmpID, Name, DeptID, DeptName) stores department details in employee rows. A new department with no staff yet cannot be recorded. Which anomaly is this? Understanding

A. Insertion anomaly
B. Update anomaly
C. Deletion anomaly
D. Referential anomaly

742. What is a candidate key? Understanding

A. Any set of attributes that determines every attribute of the relation
B. A minimal superkey: a set of attributes that determines every attribute, with no proper subset that also does
C. The single key the designer picks to identify rows
D. An attribute that refers to the primary key of another table

743. Which inference rule does 'if X → Y then XZ → YZ' state? Understanding

A. Reflexivity
B. Transitivity
C. Augmentation
D. Decomposition

744. Given X → YZ, which rule lets you conclude X → Y and X → Z? Understanding

A. Union (additivity)
B. Augmentation
C. Pseudotransitivity
D. Decomposition (projectivity)

745. What is the main goal of normalising a relational schema? Understanding

A. To store each fact once, reducing redundancy and the update, insertion and deletion anomalies it causes
B. To make every query run faster by reducing the number of joins
C. To ensure every table has a single-attribute primary key
D. To remove all NULL values from the database

746. Y is fully functionally dependent on X when: Understanding

A. X → Y holds and X is a single attribute
B. X → Y holds and no proper subset of X also determines Y
C. Y → X also holds
D. X → Y holds only through some intermediate attribute

747. In Enrol(StudentID, CourseID, StudentName, Grade) with key (StudentID, CourseID), which dependency is partial? Understanding

A. (StudentID, CourseID) → Grade
B. (StudentID, CourseID) → StudentName, Grade
C. StudentID → StudentName
D. CourseID → CourseID

748. Which describes a transitive dependency? Understanding

A. A non-prime attribute depends on part of a composite key
B. An attribute depends on itself
C. Two attributes determine each other
D. A non-prime attribute depends on the key through another non-key attribute, as in EmpID → DeptID → DeptName

749. Under the Zaniolo definition, R is in 3NF when every non-trivial FD X → A satisfies which condition? Understanding

A. X is a superkey of R, or A is a prime attribute
B. X is a superkey of R
C. A is a non-prime attribute
D. X is a single attribute

750. What does it mean for a decomposition of R into R1 and R2 to be lossless? Understanding

A. No attribute of R is missing from both R1 and R2
B. The natural join of R1 and R2 always gives back exactly the original rows of R, with no spurious tuples
C. Every FD of R can be checked inside R1 or R2
D. R1 and R2 together use less storage than R

751. A decomposition is dependency-preserving when: Understanding

A. The fragments can be joined back without spurious rows
B. Each fragment keeps the original relation's primary key
C. Every FD of the original relation can be inferred from the FDs that hold inside the individual fragments
D. No FD has a composite left-hand side

752. What does the multivalued dependency Course ↠ Teacher state? Understanding

A. Each course has exactly one teacher
B. Each teacher teaches exactly one course
C. Course and Teacher together form the primary key
D. Each course has a set of teachers that is independent of the other attributes in the relation

753. Which rule is pseudotransitivity? Understanding

A. If X → Y and WY → Z, then WX → Z
B. If X → Y and Y → Z, then X → Z
C. If X → Y and X → Z, then X → YZ
D. If XY → Z, then X → Z

754. Why can a relation be in 3NF but not in BCNF? Understanding

A. 3NF forbids partial dependencies, which BCNF allows
B. 3NF allows a non-superkey X → A when A is prime; BCNF allows no such FD
C. BCNF only applies to relations with a single candidate key
D. 3NF permits multivalued dependencies, while BCNF forbids them

755. How is the attribute-closure algorithm used to test whether a set X is a superkey of R? Understanding

A. Check whether X appears on the left of some FD in F
B. Check whether X^+ equals X
C. Compute X^+ from F; X is a superkey exactly when X^+ contains every attribute of R
D. Count the FDs whose right side lies inside X

756. Which normal form deals with join dependencies that are not implied by candidate keys? Understanding

A. Fourth Normal Form (4NF)
B. Boyce-Codd Normal Form
C. Third Normal Form
D. Fifth Normal Form (5NF, project-join normal form)

757. In AB → C, attribute B is extraneous when: Understanding

A. A → C can be derived from F, so B is not needed on the left
B. B → C holds on its own
C. B appears on the right of some other FD
D. B is not part of any candidate key

758. When does splitting R into R1 and R2 produce spurious tuples on rejoin? Understanding

A. When R1 and R2 share at least one attribute
B. When the common attributes R1 ∩ R2 are a superkey of neither R1 nor R2
C. When either fragment is in BCNF
D. When the FDs are not preserved

759. Mapping an ER design, Student and Course have an M:N relationship Enrols with attribute Grade. What is the standard relational result? Understanding

A. A CourseID foreign key and Grade column added to Student
B. A StudentID foreign key and Grade column added to Course
C. A separate table Enrols(StudentID, CourseID, Grade) with the pair (StudentID, CourseID) as its key
D. Grade stored as a multivalued column in Student

760. In a 1:N relationship where one Department has many Employees, where does the foreign key go? Understanding

A. In Department, as a list of employee IDs
B. In a new bridge table, as for every relationship
C. In both tables, referencing each other
D. In Employee, referencing Department, because each employee row names exactly one department

761. How is a multivalued attribute such as a student's phone numbers mapped from an ER diagram? Understanding

A. Into a separate relation holding the owner's key and the value, with both together as the key
B. Into several columns Phone1, Phone2 and Phone3 in Student
C. Into one comma-separated column in Student
D. Into a foreign key from Student to a Phone lookup table

762. Under which condition is the MVD X ↠ Y on R trivial? Understanding

A. Y ⊆ X only
B. Y ⊆ X, or X ∪ Y contains every attribute of R
C. X is a superkey of R
D. X and Y have no attributes in common

763. Why is every relation in 4NF also in BCNF? Understanding

A. 4NF and BCNF have the same definition
B. 4NF relations have only one candidate key
C. Every FD X → Y is also an MVD X ↠ Y, so 4NF's superkey rule for non-trivial MVDs covers every non-trivial FD
D. BCNF forbids multivalued dependencies, and 4NF is defined on top of that

764. A weak entity Dependant (partial key Name) belongs to the owner entity Employee (key EmpID). What is the primary key of the Dependant relation? Understanding

A. Name alone
B. EmpID alone
C. A new surrogate key, because weak entities cannot have keys
D. (EmpID, Name): the owner's key plus the partial key

765. R(A, B, C, D, E) has F = {A → C, C → D, B → E}. What is A^+? Application

A. {A, C, D}
B. {A, C}
C. {A, C, D, E}
D. {A, B, C, D, E}

766. Given F = {A → B, C → D}, which dependency is implied by F? Application

A. A → D
B. AC → BD
C. B → A
D. D → C

767. R(A, B, C, D, E) has F = {A → B, C → D}. What is the candidate key? Application

A. AC
B. ABCDE
C. ACE
D. ACD

768. R(A, B, C) has F = {A → B, A → C}. What is the highest normal form of R? Application

A. 3NF
B. 2NF
C. 1NF
D. BCNF

769. R(A, B, C) has F = {A → BC, B → A}. Which attributes are prime? Application

A. A and B
B. A only
C. C only
D. A, B and C

770. R(A, B, C, D, E) has F = {AB → C, C → D, D → E, E → A}. What are all the candidate keys? Application

A. AB only
B. AB, BC, BD and BE
C. AB and BE
D. B, AB and BC

771. R(A, B, C, D) has F = {A → B, B → C, C → A}. What is the highest normal form of R? Application

A. BCNF
B. 2NF
C. 3NF
D. 1NF

772. R(A, B, C, D, E) has F = {A → B, C → D}. Which decomposition is lossless? Application

A. (A, B, C) and (C, D, E)
B. (A, C) and (B, D, E)
C. (A, B, D) and (C, D, E)
D. (A, B) and (A, C, D, E)

773. What is a minimal cover of F = {A → BC, B → C, AB → D}? Application

A. {A → B, A → D, B → C}
B. {A → B, B → C, AB → D}
C. {A → BC, A → D}
D. {A → B, A → C, B → C, A → D}

774. R(A, B, C, D) has F = {A → B, B → C, C → D} and is split into (A, B), (A, C) and (C, D). Which FD is not preserved? Application

A. None: every FD is preserved
B. B → C only
C. B → C and C → D
D. A → B only

775. R(A, B, C, D, E) has F = {A → BC, CD → E, B → D, E → A}. How many candidate keys does R have? Application

A. 1
B. 2
C. 4
D. 3

776. R(A, B, C, D) has F = {A → B, B → C}, so its key is AD. Applying the BCNF decomposition algorithm and splitting first on B → C, what is the result? Application

A. (B, C), (A, B, D)
B. (A, B), (B, C), (C, D)
C. (A, B, C), (A, D)
D. (B, C), (A, B), (A, D)

777. Enrol(StudentID, CourseID, StudentName, CourseTitle, Grade) has StudentID → StudentName, CourseID → CourseTitle and (StudentID, CourseID) → Grade. What is the 2NF decomposition? Application

A. (StudentID, StudentName), (CourseID, CourseTitle), (StudentID, CourseID, Grade)
B. (StudentID, CourseID, StudentName), (CourseID, CourseTitle, Grade)
C. (StudentID, StudentName, Grade), (CourseID, CourseTitle)
D. (StudentID, CourseID), (StudentName, CourseTitle, Grade)

778. Are F = {A → B, B → C, C → A} and G = {A → C, C → B, B → A} equivalent? Application

A. No: G reverses every dependency, so it means the opposite
B. No: F contains A → B, which G cannot derive
C. Yes: under both sets, each of A, B and C determines the other two
D. Only if both sets also include A → BC

779. R(A, B, C, D, E, F) has F = {A → B, C → D, BD → E, E → F}. What is $(AC)^+$? Application

A. {A, B, C, D}
B. {A, B, C, D, E}
C. {A, C}
D. {A, B, C, D, E, F}

780. R(A, B, C, D, E, F) has F = {A → B, B → C, CD → E, E → F, F → D}. What are all the candidate keys? Application

A. AD, AE and AF
B. AD only
C. AC, AD and AE
D. ADE

781. What is a minimal cover of F = {A → BC, CD → E, B → D, E → A, AB → E}? Application

A. {A → B, B → D, CD → E, E → A}
B. {A → B, A → C, B → D, CD → E, E → A}
C. {A → B, A → C, B → D, CD → E, E → A, A → E}
D. {A → BC, B → D, CD → E, E → A, AB → E}

782. R(A, B, C, D, E) has the minimal cover {A → B, A → C, C → D, D → C}. What does 3NF synthesis produce? Application

A. (A, B), (A, C), (C, D), (D, C)
B. (A, B, C), (C, D)
C. (A, B, C), (C, D), (A, E)
D. (A, B, C, D), (A, E)

783. R(A, B, C, D) has F = {A → B, B → C, C → D}. Which three-way decomposition is lossless? Application

A. (A, B), (C, D), (A, D)
B. (A, C), (B, D), (C, D)
C. (A, D), (B, C), (C, D)
D. (A, B), (B, C), (C, D)

784. R(A, B, C, D) has exactly two candidate keys, AB and CD. How many superkeys does R have? Application

A. 7
B. 8
C. 6
D. 4

785. Today's rows of Staff(ID, City) are (1, Lahore), (2, Lahore), (3, Quetta). A student concludes that ID → City is a functional dependency of the schema. What is correct? Analysis

A. The FD is proven, because no two rows with the same ID disagree on City
B. The data is only consistent with ID → City; an FD comes from the meaning of the attributes, not from one instance
C. City → ID also holds, because the data shows it
D. No FD can hold, because two rows share Lahore

786. R(A, B, C, D, E) has F = {AB → CDE, C → A, D → B}. Its keys are AB, AD, BC and CD. A student says R is in BCNF 'because almost every attribute is prime'. What is the correct verdict? Analysis

A. BCNF, since every left side is a candidate key
B. 2NF only, because E depends on AB
C. 3NF but not BCNF: C → A and D → B have non-superkey left sides, but A and B are prime
D. 1NF only, because C → A is a partial dependency

787. Assign(Emp, Project, City) has rows (Ali, ERP, Lahore) and (Sana, CRM, Lahore). It is split into (Emp, City) and (Project, City), then rejoined on City. How many rows does the join produce, and what does it show? Analysis

```
CREATE TABLE ec (emp TEXT, city TEXT);
INSERT INTO ec VALUES ('Ali','Lahore'),('Sana','Lahore');
CREATE TABLE pc (project TEXT, city TEXT);
INSERT INTO pc VALUES ('ERP','Lahore'),('CRM','Lahore');
SELECT COUNT(*) FROM ec NATURAL JOIN pc;
```

A. 2 rows, because the split is lossless
B. 3 rows, because one combination is duplicated
C. 4 rows, all of them correct, because the join keeps every combination
D. 4 rows, 2 of them spurious, because City is not a key of either fragment

788. Orders stores order_total alongside OrderLine rows that already hold each line's amount, so reports avoid a SUM. What is the main risk, and how is it usually managed? Analysis

A. The stored total can disagree with its lines after an edit, so it must be maintained by a trigger or in the same transaction
B. None, because a derived value cannot become inconsistent
C. It breaks 1NF, so the table cannot be created
D. It creates a deletion anomaly that removes the order when its total changes

789. OrderLine(OrderID, ProductID, ProductName, Qty) has key (OrderID, ProductID) and ProductID → ProductName. Renaming a product requires editing hundreds of rows. What is the highest normal form, and the fix? Analysis

A. 2NF; move Qty to a separate table
B. 1NF; move (ProductID, ProductName) to its own table, leaving OrderLine(OrderID, ProductID, Qty)
C. 3NF; the repetition is acceptable
D. BCNF; add a surrogate key to OrderLine

790. R(A, B, C, D) has F = {A → B, BC → D}. A student claims A → D follows from F. Is the claim correct? Analysis

A. Yes, by transitivity through B
B. Yes, by pseudotransitivity with W = C
C. No: A^+ = {A, B}, so A → D does not follow; AC → D does
D. No, because no FD can have D on the right

791. Student(RollNo, Name, DeptID, DeptName) has RollNo → Name, DeptID and DeptID → DeptName. A designer adds an auto-number SID as primary key and says the table is now in 3NF. What is correct? Analysis

A. It is in 3NF, because every attribute now depends on SID
B. It is in BCNF, because SID is a single-attribute key
C. It is now in 1NF only, because there are two keys
D. It is still not in 3NF: DeptID → DeptName has a non-superkey left side and a non-prime right side

792. Course DB has teachers {Saad, Amna} and books {B1, B2, B3}, independent of each other, stored in CTB(Course, Teacher, Book). After the 4NF decomposition into (Course, Teacher) and (Course, Book), how many rows does DB need in total? Analysis

A. 5 instead of 6
B. 6 instead of 6
C. 5 instead of 5
D. 3 instead of 6

793. R(A, B, C, D, E) has F = {A → BC, CD → E, B → D, E → A}, with keys A, E, BC and CD. What is the highest normal form of R, and why? Analysis

A. BCNF: every attribute is part of some key
B. 3NF: B → D has a non-superkey left side, but D is prime
C. 2NF: B → D is a transitive dependency
D. 1NF: B → D is a partial dependency on the key BC

794. R(A, B, C, D, E) has F = {A → B, B → C, D → E}. Which decomposition is lossless? Analysis

A. (A, B), (B, C), (C, D), (D, E)
B. (A, B, C), (C, D, E)
C. (A, B), (B, C), (A, D), (D, E)
D. (A, B), (C, D), (A, D, E)

795. F = {A → BC, B → AC, C → AB}. Which of these is NOT a minimal cover of F? Analysis

A. {A → B, B → C, C → A}
B. {A → C, C → B, B → A}
C. {A → B, B → A, A → C, C → A}
D. {A → B, B → C, C → A, A → C}

796. R(A, B, C, D) has F = {A → B, C → D} and is split into (A, B) and (C, D). Which statement is correct? Analysis

A. It preserves both FDs but is lossy, because the fragments share no attribute
B. It is lossless and preserves both FDs
C. It is lossless but loses C → D
D. It is lossy and loses both FDs

797. R(A, B, C, D, E, F) has F = {A → B, B → C, CD → E, E → F, F → D}. A student reasons 'A is on no right side, so the only key is AD'. How many candidate keys does R actually have? Analysis

A. 1
B. 3
C. 2
D. 4

798. R(A, B, C, D) has F = {AB → CD, C → A}, with keys AB and BC. A student says R is only in 1NF, because C → A is a partial dependency on the key BC. What is the correct highest normal form? Analysis

A. 1NF
B. 2NF
C. 3NF
D. BCNF

799. R(A, B, C, D) has F = {A → BCD, BC → A, D → B}, with keys A, BC and CD. A designer splits on D → B into (B, D) and (A, C, D). Which statement is correct? Analysis

A. The split is lossy, because D is not a key of (A, C, D)
B. (A, C, D) is not in BCNF, so another split is needed
C. Every FD is preserved, because BC → A follows from A → BCD
D. The split is lossless and both parts are in BCNF, but BC → A is no longer preserved

Transaction Management MCQs for NSCT

800. A transfer debits account A and then the system crashes before crediting account B. On restart, recovery undoes the debit. Which ACID property does this protect? Understanding
    A. Consistency
    B. Isolation
    C. Durability
    D. Atomicity

801. Which DBMS component is mainly responsible for durability? Understanding
    A. The query optimiser
    B. The concurrency-control manager
    C. The recovery manager, using the log
    D. The application programmer

802. Which ACID property is the concurrency-control component (for example, a lock manager) responsible for? Understanding
    A. Atomicity
    B. Durability
    C. Consistency
    D. Isolation

803. According to Silberschatz, who is mainly responsible for the consistency property of a single transaction? Understanding
    A. The recovery manager
    B. The concurrency-control manager
    C. The application programmer who writes the transaction, helped by integrity constraints
    D. The operating system's file system

804. A transaction has executed its final statement but has not yet committed. Which state is it in? Understanding
    A. Active
    B. Partially committed
    C. Committed
    D. Failed

805. Two operations in a schedule conflict when they: Understanding
    A. Belong to the same transaction and access the same item
    B. Belong to different transactions and are both reads of the same item
    C. Belong to different transactions, access the same item, and at least one is a write
    D. Belong to different transactions and access different items

806. What is a serial schedule? Understanding
    A. A schedule in which operations from transactions alternate one by one
    B. Any schedule that produces a consistent final state
    C. A schedule in which each transaction runs all its operations before the next one starts
    D. A schedule in which transactions run in order of their ID numbers

807. A schedule is recoverable if, whenever Tj reads a data item written by Ti: Understanding
    A. Ti commits before Tj reads the item
    B. Tj never writes the same item
    C. Ti and Tj commit at the same moment
    D. Ti commits before Tj commits

808. What does a cascadeless (ACA) schedule guarantee? Understanding

A. No transaction ever aborts
B. The schedule is conflict serializable
C. Transactions read only values written by committed transactions, so one abort cannot force others to roll back
D. No two transactions write the same data item

809. Which situation is a dirty read? Understanding

A. T2 reads the same item twice and gets different committed values
B. T2 re-runs a query and finds new rows inserted by T1
C. T1 and T2 both read a value, and T1's write is overwritten by T2
D. T2 reads a value written by T1 before T1 has committed

810. Which SQL isolation level prevents dirty reads but still allows non-repeatable reads? Understanding

A. READ COMMITTED
B. READ UNCOMMITTED
C. REPEATABLE READ
D. SERIALIZABLE

811. Which statement about conflict serializability (CSR) and view serializability (VSR) is correct? Understanding

A. Every VSR schedule is CSR
B. Every CSR schedule is VSR; a VSR schedule that is not CSR must contain a blind write
C. CSR and VSR are the same set of schedules
D. A schedule with a blind write can never be VSR

812. When is a schedule strict? Understanding

A. No transaction reads or writes an item until the last transaction that wrote it has committed or aborted
B. Every transaction holds all its locks until it commits
C. Transactions run strictly one after another
D. No transaction ever writes an item that another transaction has read

813. After a transaction enters the aborted state, what can the system do next? Understanding

A. Restart it as a new transaction, or kill it
B. Resume it from the statement where it failed
C. Commit the changes it made before the failure
D. Move it back to the partially committed state

814. T1 runs SELECT COUNT(*) FROM orders WHERE city = 'Lahore' twice. Between the two runs, T2 inserts a Lahore order and commits. T1 gets 10, then 11. What is this called? Understanding

A. A dirty read
B. A phantom read
C. A lost update
D. A cascading rollback

815. How are the sets of recoverable, cascadeless and strict schedules related? Understanding

A. Recoverable $\subset$ cascadeless $\subset$ strict
B. Strict $\subset$ cascadeless $\subset$ recoverable
C. Cascadeless $\subset$ strict $\subset$ recoverable
D. They are independent; none contains another

816. How many different serial schedules are possible for 3 transactions? Application

A. 3
B. 9
C. 27
D. 6

817. T1 has 3 operations and T2 has 2 operations. How many schedules (interleavings) keep each transaction's own operation order? Application

A. 5
B. 120
C. 10
D. 6

818. Which edges does the precedence graph of this schedule contain? Application

```
R1(A) R2(A) W1(A) W2(B) R1(B)
```

A. Only T1 → T2
B. T1 → T2 and T2 → T1
C. No edges
D. Only T2 → T1

819. Is this schedule conflict serializable, and if so, to which serial order? Application

```
R1(A) W1(A) R2(A) W2(A) R1(B) W1(B) R2(B) W2(B)
```

A. Yes, it is equivalent to T1 then T2
B. Yes, it is equivalent to T2 then T1
C. No, the precedence graph has a cycle
D. Yes, it is equivalent to both serial orders

820. Is this schedule conflict serializable? Application

```
R1(A) R2(A) W1(A) W2(A)
```

A. No: R2(A) before W1(A) gives T2 → T1, and W1(A) before W2(A) gives T1 → T2
B. Yes, equivalent to T1 then T2
C. Yes, equivalent to T2 then T1
D. Yes, because both transactions only touch one item

821. To which serial order is this three-transaction schedule conflict equivalent? Application

```
R3(A) W1(A) R1(B) W2(B) R2(C) W3(D)
```

A. T1, T2, T3
B. T2, T1, T3
C. T3, T1, T2
D. T3, T2, T1

822. A = 100. T1 does A = A − 10 and T2 does A = A + 20. They run as R1(A) R2(A) W1(A) W2(A), each writing the value it computed from its own read. What is the final value of A? Application

A. 110
B. 90
C. 120
D. 100

823. What balances does the final SELECT return for accounts 1 and 2? Application

```
CREATE TABLE acc (id INT, bal INT);
INSERT INTO acc VALUES (1, 500), (2, 300);
BEGIN;
UPDATE acc SET bal = bal - 100 WHERE id = 1;
UPDATE acc SET bal = bal + 100 WHERE id = 2;
COMMIT;
BEGIN;
UPDATE acc SET bal = bal - 50 WHERE id = 1;
UPDATE acc SET bal = bal + 50 WHERE id = 2;
ROLLBACK;
SELECT id, bal FROM acc ORDER BY id;
```

A. 350 and 450
B. 500 and 300
C. 450 and 350
D. 400 and 400

824. Classify this schedule. Application

```
W1(A) R2(A) C2 C1
```

A. Recoverable but not cascadeless
B. Cascadeless, because T2 commits first
C. Not recoverable: T2 reads A from T1 but commits before T1
D. Strict, because each transaction commits

825. Classify this schedule. Application

```
W1(A) R2(A) C1 C2
```

A. Recoverable but not cascadeless
B. Not recoverable
C. Cascadeless but not strict
D. Strict

826. Which anomaly does this schedule show? Application

```
R1(A) W2(A) C2 R1(A) C1
```

A. Dirty read
B. Non-repeatable (unrepeatable) read
C. Phantom read
D. Lost update

827. How many serial orders is this schedule conflict equivalent to? Application

```
W1(A) R2(A) W1(B) R3(B) W2(C) R3(D)
```

A. 1
B. 2
C. 3
D. 6

828. How many pairs of conflicting operations does this schedule contain? Application

```
R1(A) W2(A) W1(A) R2(B) W1(B)
```

A. 2
B. 3
C. 4
D. 5

829. T1 = R1(A) W1(A) and T2 = R2(A) W2(A). Of all the interleavings that keep each transaction's order, how many are conflict serializable? Application

A. 4 of 6
B. 1 of 6
C. 6 of 6
D. 2 of 6

830. Classify this schedule. Analysis

```
R1(A) W2(A) W1(A) W3(A)
```

A. View serializable but not conflict serializable
B. Conflict serializable
C. Neither conflict nor view serializable
D. Conflict serializable but not view serializable

831. Which statement about this schedule is correct? Analysis

```
W1(A) R2(A) C2 W1(B) C1
```

A. It is not conflict serializable but is recoverable
B. It is conflict serializable and cascadeless
C. It is neither conflict serializable nor recoverable
D. It is conflict serializable (T1 then T2) but not recoverable

832. T1 aborts after this prefix of the schedule. Which other transactions must also be rolled back? Analysis

```
W1(A) R2(A) W2(B) R3(B) R4(C)
```

A. T2 and T3
B. T2 only
C. T2, T3 and T4
D. None

833. Which classification of this schedule is correct? Analysis

```
R1(A) W1(A) R2(A) C1 W2(A) C2
```

A. Conflict serializable and recoverable, but not cascadeless
B. Conflict serializable and cascadeless
C. Recoverable but not conflict serializable
D. Not recoverable

834. A report reads each customer's balance twice and must see the same value both times. New customers added meanwhile do not matter. Which is the weakest SQL-standard isolation level that is enough? Analysis

    A. READ COMMITTED
    B. READ UNCOMMITTED
    C. SERIALIZABLE
    D. REPEATABLE READ

835. Classify this schedule. Analysis

```
W1(A) W2(A) C1 C2
```

    A. Cascadeless but not strict
    B. Strict
    C. Recoverable but not cascadeless
    D. Not recoverable

836. A = 100. T1 does A = A − 10 and T2 does A = A + 20. Which interleaving leaves the correct final value of 110? Analysis

    A. R1(A) R2(A) W1(A) W2(A)
    B. R1(A) W1(A) R2(A) W2(A)
    C. R1(A) R2(A) W2(A) W1(A)
    D. R2(A) R1(A) W1(A) W2(A)

837. Which serial orders is this schedule conflict equivalent to? Analysis

```
R1(A) R2(B) W3(A) W3(B)
```

    A. Only T1, T2, T3
    B. T1, T2, T3 and T2, T1, T3
    C. Only T3, T1, T2
    D. None, because the graph has a cycle

838. A bank app shows 'Transfer successful' after COMMIT returns. A power cut follows, and after restart the transfer is missing. Which property was violated, and what probably went wrong? Analysis

    A. Atomicity: the transfer was only half applied
    B. Durability: the commit log record was not forced to stable storage before success was reported
    C. Isolation: another transaction read the transfer too early
    D. Consistency: the transfer broke an integrity constraint

839. S1 = R1(A) W2(B) W1(A) R2(A). Which schedule is conflict equivalent to S1? Analysis

    A. R1(A) R2(A) W1(A) W2(B)
    B. R2(A) W2(B) R1(A) W1(A)
    C. W2(B) R1(A) W1(A) R2(A)
    D. R2(A) R1(A) W1(A) W2(B)

840. An active transaction hits a divide-by-zero error and cannot continue normally. Using Silberschatz's transaction state diagram, which state does it enter next? Understanding

    A. Aborted
    B. Failed
    C. Partially committed
    D. Committed

841. Which state transition is NOT possible in Silberschatz's transaction state diagram? Understanding

A. Active to failed
B. Committed to aborted
C. Partially committed to failed
D. Partially committed to committed

842. Why does a DBMS let transactions run concurrently instead of strictly one after another? Understanding

A. To improve throughput and resource use and to cut average response time
B. Because concurrent schedules are automatically serializable
C. Because concurrency removes the need for a recovery manager
D. So that each transaction can skip its consistency checks

843. In schedule analysis, what is a blind write? Understanding

A. A write that is made by an uncommitted transaction
B. A write of an item by a transaction that has not read that item first
C. A write that is later undone by an abort
D. A write that no other transaction ever reads

844. A transfer must leave the total A + B unchanged when it runs alone on a database that starts out correct. Which ACID property states this requirement? Understanding

A. Atomicity
B. Isolation
C. Durability
D. Consistency

845. A database client runs in autocommit mode. What does that mean? Understanding

A. Each SQL statement is its own transaction and commits automatically if it succeeds
B. All statements in a session form one transaction that commits at logout
C. The DBMS commits a transaction only after every concurrent transaction finishes
D. Statements are never rolled back, even if they fail

846. What is a cascading rollback? Understanding

A. A rollback that undoes every transaction since the last checkpoint
B. Rolling a transaction back to a savepoint instead of to its start
C. Restarting a transaction several times until it commits
D. The abort of one transaction forces other transactions that read its uncommitted data to abort too

847. In the SQL standard, which is the only isolation level that also prevents phantom reads? Understanding

A. SERIALIZABLE
B. REPEATABLE READ
C. READ COMMITTED
D. READ UNCOMMITTED

848. Durability needs data that survives any failure. What kind of storage does Silberschatz say the log must ultimately be written to? Understanding

A. Stable storage
B. Volatile storage such as main memory
C. The CPU cache
D. The buffer pool

849. T1 sums all account balances while T2 moves money between two accounts. T1 reads one account before T2 changes it and the other after. What do Elmasri and Navathe call this problem? Understanding

A. The lost update problem
B. The temporary update (dirty read) problem
C. The incorrect summary problem
D. The phantom problem

850. In Silberschatz's shadow-copy scheme, at what moment is a transaction considered committed? Understanding

A. When the transaction issues its last write in memory
B. When the old copy of the database is deleted
C. When the new copy has been created in main memory
D. When the updated db-pointer, pointing to the new copy, has been written to disk

851. According to the SQL standard (not product defaults), what is the default isolation level? Understanding

A. READ COMMITTED
B. SERIALIZABLE
C. REPEATABLE READ
D. READ UNCOMMITTED

852. When are two schedules conflict equivalent? Understanding

A. They leave the same final values for every initial database
B. They contain the same operations and order every pair of conflicting operations the same way
C. They have the same reads-from relationships and the same final writes
D. They contain the same operations in exactly the same order

853. Which condition is NOT part of the definition of view equivalence between two schedules? Understanding

A. A transaction that reads the initial value of an item does so in both schedules
B. Each read reads the value written by the same transaction in both schedules
C. The same transaction performs the final write of each item in both schedules
D. Every pair of conflicting operations appears in the same order

854. Why do DBMSs test for conflict serializability rather than view serializability? Understanding

A. Conflict serializability accepts more schedules than view serializability
B. View serializability ignores write operations
C. View serializable schedules are never recoverable
D. Conflict serializability is checked in polynomial time by cycle detection, while testing view serializability is NP-complete

855. Consider any serial schedule in which each transaction commits before the next one starts. Which statement is true? Understanding

A. It is recoverable but may cause cascading rollbacks
B. It is conflict serializable but may be unrecoverable
C. It is cascadeless but not strict
D. It is strict, so it is also cascadeless and recoverable

856. Why must a DBMS forbid non-recoverable schedules? Understanding

A. They always contain a cycle in the precedence graph
B. They force every transaction to run serially
C. A committed transaction may have read data from a transaction that then aborts, and undoing a committed transaction would break durability
D. They can never commit any transaction

857. Inside a transaction, a program issues ROLLBACK TO SAVEPOINT s1. What happens? Understanding

A. The whole transaction is undone and ends
B. Changes made before s1 are undone and later ones are kept
C. Changes made after s1 are undone and the transaction stays active
D. The transaction commits everything up to s1

858. Why is 'the two schedules produce the same final values' not used as the definition of schedule equivalence? Understanding

A. Final values cannot be computed without running the DBMS
B. Equal final values imply that the schedules are serial
C. Two schedules can produce the same values by coincidence for some initial values but not for others
D. Final values ignore the writes in the schedule

859. According to Elmasri and Navathe, when does a transaction reach its commit point? Understanding

A. When all its database operations have executed successfully and their effects have been recorded in the log
B. When it issues its first write
C. When every page it changed has been written to the database on disk
D. When all concurrently running transactions have finished

860. Silberschatz describes snapshot isolation, in which each transaction reads from a snapshot taken when it started. Which anomaly can snapshot isolation still allow? Understanding

A. Write skew
B. Dirty read
C. Lost update between two writers of the same item
D. Non-repeatable read

861. Elmasri and Navathe note that strict schedules simplify recovery. Why? Understanding

A. Strict schedules never need a log
B. Strict schedules never contain aborts
C. Strict schedules allow every write to be done in place without logging the old value
D. To undo an aborted transaction's write, the system can simply restore the item's before-image

862. Silberschatz notes that all four SQL isolation levels share one extra guarantee. What is it? Understanding

A. None of them allows dirty writes to an item written by an uncommitted transaction
B. None of them allows dirty reads
C. All of them prevent phantoms
D. All of them guarantee conflict serializability

863. T1 does A := A - 50, B := B + 50 and T2 does B := B - 10, A := A + 10. Some interleavings that fail the conflict-serializability test still give the same result as a serial run. Why does the test reject them? Understanding

A. It considers only the final values of the database
B. Those interleavings always contain a dirty read
C. Conflict serializability requires that transactions never touch the same item
D. It looks only at reads and writes, not at the fact that additions and subtractions commute

864. T1 has 4 operations and T2 has 2 operations. How many schedules keep each transaction's own operation order? Application

A. 720
B. 30
C. 8
D. 15

865. A = 50. T1 does A := A x 2 and T2 does A := A + 10. They run as R1(A) R2(A) W2(A) W1(A), each writing the value computed from its own read. What is the final value of A? Application

A. 110
B. 120
C. 100
D. 60

866. A = 100. T1 sets A to 200 but has not committed. T2 reads A, sets B := A + 50 and commits. Then T1 aborts. What value does B hold, and what problem is this? Application

A. 150, because T1's abort also fixes B
B. 250, caused by a lost update
C. 250, caused by a dirty read
D. 150, caused by a non-repeatable read

867. Which anomaly does this schedule show? Application

```
R1(A) R2(A) W2(A) C2 W1(A) C1
```

A. Dirty read
B. Non-repeatable read
C. Lost update
D. Phantom read

868. Which pair of adjacent operations can be swapped without changing the result of any schedule that contains it? Application

A. W1(A) R2(A)
B. R1(A) W2(A)
C. W1(B) W2(B)
D. W1(A) R2(B)

869. This script runs in SQLite, which is in autocommit mode outside BEGIN ... ROLLBACK. What does the final SELECT return? Application

```
CREATE TABLE marks (roll INT);
INSERT INTO marks VALUES (1);
BEGIN;
INSERT INTO marks VALUES (2);
DELETE FROM marks WHERE roll = 1;
ROLLBACK;
INSERT INTO marks VALUES (3);
SELECT COUNT(*) FROM marks;
```

A. 2
B. 1
C. 3
D. 0

870. Three transactions have 2 operations each. How many schedules keep each transaction's own operation order? Application

A. 720
B. 6
C. 90
D. 48

871. To which serial order is this schedule conflict equivalent? Application

```
R3(Y) R1(X) W2(Y) W1(Y) R3(X)
```

A. T1, T2, T3
B. T3, T1, T2
C. It is not conflict serializable
D. T3, T2, T1

872. How many edges does the precedence graph of this schedule have? Application

```
R1(A) R2(B) W2(A) W3(B) R1(C) W3(C)
```

A. 3
B. 4
C. 2
D. 5

873. How many pairs of conflicting operations does this schedule contain? Application

```
W1(A) R2(B) R2(A) W2(B) R1(B) W3(A)
```

A. 4
B. 3
C. 5
D. 6

874. The schedule W1(A) R2(A) W2(B) R3(B) is completed with a commit for each transaction. Which commit order makes it recoverable? Application

A. C3 C2 C1
B. C2 C1 C3
C. C1 C2 C3
D. C1 C3 C2

875. This script runs in SQLite. What does the final SELECT return? Application

```
CREATE TABLE stock (item TEXT, qty INT);
INSERT INTO stock VALUES ('pen', 10);
BEGIN;
UPDATE stock SET qty = qty - 3 WHERE item = 'pen';
SAVEPOINT s1;
UPDATE stock SET qty = qty - 5 WHERE item = 'pen';
ROLLBACK TO s1;
UPDATE stock SET qty = qty + 1 WHERE item = 'pen';
COMMIT;
SELECT qty FROM stock;
```

A. 8
B. 3
C. 11
D. 7

876. T1 runs at READ COMMITTED and reads seats = 5. T2 then sets seats = 4 and commits. T1 reads seats again in the same transaction. What does T1 see? Application

A. 5, because READ COMMITTED repeats earlier reads
B. 4, a non-repeatable read that READ COMMITTED allows
C. An error, because T2's update conflicts with T1's read
D. 5, because T2 had to wait for T1 to finish

877. A = B = 100. T1 moves 20 from A to B (A := A - 20, B := B + 20). T2 prints A + B. They run as R1(A) W1(A) R2(A) R2(B) R1(B) W1(B). What total does T2 print? Application

A. 180
B. 200
C. 220
D. 160

878. T1, T2 and T3 have 1, 2 and 3 operations respectively. How many schedules keep each transaction's own operation order? Application

A. 720
B. 60
C. 36
D. 6

879. Is this schedule conflict serializable? Application

```
R1(A) W2(A) R3(B) W1(B) W3(A)
```

A. Yes, equivalent to T1, T2, T3
B. No: R1(A) before W3(A) gives T1 -> T3, and R3(B) before W1(B) gives T3 -> T1
C. Yes, equivalent to T3, T1, T2
D. No: W2(A) before W3(A) gives T2 -> T3, and that alone forms a cycle

880. How many serial orders is this four-transaction schedule conflict equivalent to? Application

```
R1(A) W2(A) R3(B) W4(B)
```

A. 2
B. 24
C. 6
D. 4

881. This schedule is not conflict serializable. To which serial schedule is it view equivalent? Application

```
R2(A) W1(A) W2(A) W3(A)
```

A. T1, T2, T3
B. T2, T3, T1
C. T2, T1, T3
D. None: a cycle in the precedence graph rules out view serializability too

882. T1 = R1(A) W1(A) and T2 = W2(A) R2(B). Of all the interleavings that keep each transaction's order, how many are conflict serializable? Application

A. 2 of 6
B. 6 of 6
C. 3 of 6
D. 4 of 6

883. A schedule's precedence graph has the edges T1 -> T2, T2 -> T3, T1 -> T3 and T4 -> T2. How many conflict-equivalent serial orders exist? Application

A. 1
B. 2
C. 4
D. 24

884. Which of these schedules is conflict serializable? Application

A. R1(A) W2(A) R2(B) W1(B)
B. R1(A) W2(A) R2(B) W1(C) R3(C) W3(B)
C. W1(A) R2(A) W2(B) R1(B)
D. R1(A) R2(B) W2(A) W1(B)

885. A dashboard showed salary totals from a payroll update that was later rolled back, so the figures never existed. Which isolation level was the dashboard probably using, and what is the minimum fix? Analysis

A. READ COMMITTED; raise it to REPEATABLE READ
B. REPEATABLE READ; raise it to SERIALIZABLE
C. READ UNCOMMITTED; raise it to READ COMMITTED
D. SERIALIZABLE; lower it to READ COMMITTED

886. A developer says: 'Our schedules are always conflict serializable, so an abort can never cause trouble.' What is wrong with this claim? Analysis

A. Nothing; conflict serializable schedules are always strict
B. Conflict serializable schedules cannot contain aborts at all
C. Aborts only matter for view serializable schedules
D. Serializability ignores aborts; a serializable schedule can still be unrecoverable or cause cascading rollbacks

887. Classify this schedule. Analysis

```
R1(A) W2(A) C2 W1(A) C1
```

A. Conflict serializable, but not recoverable
B. Conflict serializable and strict
C. Strict, but not conflict serializable
D. Recoverable, but not cascadeless

888. An exam-booking transaction counts the bookings for Hall 3 and inserts a new one only if the count is below 50. Two such transactions must never both see 49. Which is the weakest SQL-standard isolation level that guarantees this? Analysis

A. REPEATABLE READ
B. SERIALIZABLE
C. READ COMMITTED
D. READ UNCOMMITTED

889. Which completed schedule built from W1(A) R2(A) W2(B) is cascadeless? Analysis

A. W1(A) R2(A) C1 W2(B) C2
B. W1(A) C1 R2(A) W2(B) C2
C. W1(A) R2(A) W2(B) C1 C2
D. W1(A) R2(A) W2(B) C2 C1

890. An auditor sums all balances while transfers run. The total is short by exactly the amount of one transfer that was in progress. What is the most likely explanation? Analysis

A. The transfer's update was lost when another transfer overwrote it
B. The audit read the source account after its debit and the destination before its credit
C. The transfer committed but was lost in a crash
D. A new account was inserted during the audit

891. S1 = W1(A) R2(B) R2(A) W1(B). Which schedule is conflict equivalent to S1? Analysis

A. W1(A) R2(A) W1(B) R2(B)
B. R2(B) R2(A) W1(A) W1(B)
C. W1(A) W1(B) R2(B) R2(A)
D. R2(B) W1(A) R2(A) W1(B)

892. Which schedule has exactly the precedence edges T1 -> T2 and T3 -> T2, and no others? Analysis

A. R1(A) W2(A) R2(B) W3(B)
B. W2(A) R1(A) R3(B) W2(B)
C. R1(A) R3(B) W2(A) W2(B)
D. R1(A) W3(A) W2(A) R3(B)

893. A = B = 100. T1 adds 10 to A and to B; T2 doubles A and B. Each item is read and then written. Which interleaving leaves A different from B? Analysis

A. R1(A) W1(A) R2(A) W2(A) R2(B) W2(B) R1(B) W1(B)
B. R1(A) W1(A) R1(B) W1(B) R2(A) W2(A) R2(B) W2(B)
C. R1(A) W1(A) R2(A) W2(A) R1(B) W1(B) R2(B) W2(B)
D. R2(A) W2(A) R1(A) W1(A) R2(B) W2(B) R1(B) W1(B)

894. After the prefix below, T1 aborts. What is the situation? Analysis

```
W1(A) R2(A) C2 R3(A) W3(B)
```

A. T3 must roll back, but T2 has already committed and cannot, so the schedule is not recoverable
B. T2 and T3 both roll back normally
C. Only T1 rolls back, because T2 committed before the abort
D. Nothing else is affected, because T3 read A after T2 committed

895. A transaction has executed its last statement and is partially committed when the system crashes, before its commit record reaches the log on disk. What happens on restart? Analysis

A. Its changes are redone, because it had finished all its statements
B. It is left in the partially committed state until an administrator decides
C. Only the changes already written to disk are kept
D. Its changes are undone, because it never committed

896. Removing all operations of which single transaction leaves a conflict-serializable schedule? Analysis

```
R1(A) R2(B) R2(C) R3(D) W2(A) W1(B) W3(C) W2(D)
```

A. T1
B. T2
C. T3
D. Removing any one of them is enough

897. Compare the number of serial orders this schedule is conflict equivalent to with the number it is view equivalent to. Analysis

```
W1(A) W2(A) W3(A) R4(A)
```

A. 1 conflict-equivalent order, 2 view-equivalent orders
B. 1 and 1
C. 2 and 2
D. 0 conflict-equivalent orders, 1 view-equivalent order

898. Which schedule is view serializable but NOT conflict serializable? Analysis

A. R1(A) W2(A) W1(A) R2(A)
B. W1(A) R2(B) W2(A) W1(A) W3(A)
C. W1(A) R2(A) W2(B) R1(B)
D. R1(A) W2(A) W1(B) W3(B)

899. Which schedule is conflict serializable and cascadeless but NOT strict? Analysis

A. R2(B) W1(A) W2(B) C1 C2
B. W1(A) R2(A) W2(B) C1 C2
C. W1(A) R2(B) W2(A) C2 C1
D. R1(A) W2(A) W1(A) C1 C2

Concurrency Control MCQs for NSCT

900. Under the standard shared/exclusive (S/X) locking protocol, which lock request can be granted at once while another transaction already holds a lock on the same item? Understanding

A. An exclusive lock requested while the other transaction holds a shared lock
B. A shared lock requested while the other transaction holds a shared lock
C. A shared lock requested while the other transaction holds an exclusive lock
D. An exclusive lock requested while the other transaction holds an exclusive lock

901. Which ACID property is concurrency control mainly responsible for enforcing? Understanding

A. Isolation
B. Atomicity
C. Durability
D. Consistency of each individual transaction's logic

902. In the two-phase locking (2PL) protocol, what may a transaction do during its growing phase? Understanding

A. Release locks but not acquire any
B. Acquire locks but not release any
C. Acquire and release locks freely
D. Read items but not write them

903. What is the lost update problem in concurrent transactions? Understanding

A. A transaction reads a value written by another transaction that later aborts
B. A transaction reads the same item twice and gets two different values
C. Two transactions read the same item and both write it back, so one transaction's write overwrites the other's
D. An aggregate is computed while another transaction is updating some of the rows

904. T2 reads a value that T1 wrote, and then T1 aborts. What do Elmasri-Navathe call this problem? Understanding

A. The lost update problem
B. The temporary update (dirty read) problem
C. The incorrect summary problem
D. The unrepeatable read problem

905. In deadlock detection, what does a cycle in the wait-for graph indicate? Understanding

A. The schedule is not conflict serialisable
B. The transactions on the cycle are deadlocked
C. One transaction is starving but there is no deadlock
D. The lock manager must upgrade a shared lock

906. What does basic two-phase locking guarantee? Understanding

A. Every schedule it allows is conflict serialisable
B. Deadlocks can never occur
C. Cascading rollbacks can never occur
D. No transaction ever has to wait for a lock

907. What extra rule does strict two-phase locking add to basic 2PL, and what does it buy? Understanding

A. Exclusive locks are held until commit or abort, so no one reads uncommitted data and cascading rollbacks are avoided
B. All locks are acquired before the transaction starts, so deadlocks cannot occur
C. Shared locks are released immediately after each read, so concurrency improves
D. Locks are granted in timestamp order, so starvation cannot occur

908. Which variant of two-phase locking is free of deadlock? Understanding

A. Strict 2PL
B. Rigorous 2PL
C. Conservative (static) 2PL
D. Basic 2PL

909. Under rigorous two-phase locking (all locks held until commit), in what order can the transactions be serialised? Understanding

A. The order in which they commit
B. The order in which they started
C. The order of their first lock requests
D. The reverse of their commit order

910. What is the lock point of a transaction under two-phase locking? Understanding

A. The moment it acquires its final lock
B. The moment it acquires its first lock
C. The moment it commits
D. The moment it performs its first write

911. In the basic timestamp-ordering protocol, which serial order must the produced schedule be equivalent to? Understanding

A. The order in which transactions commit
B. The order of decreasing timestamps (newest transactions first)
C. Any order, as long as the precedence graph is acyclic
D. The order of increasing transaction timestamps (older transactions first)

912. What does Thomas' write rule change in the basic timestamp-ordering protocol? Understanding

A. A read with TS(T) < W-timestamp(X) is ignored instead of rolling T back
B. A write with TS(T) < R-timestamp(X) is ignored instead of rolling T back
C. Conflicting transactions wait instead of being rolled back
D. A write with TS(T) < W-timestamp(X) is ignored instead of rolling T back

913. Which statement about the wait-die and wound-wait schemes is correct? Understanding

A. Wait-die is preemptive: an older requester forces a younger lock holder to roll back
B. In wait-die, a younger requester waits for an older holder
C. Wound-wait is preemptive: an older requester forces a younger lock holder to roll back
D. In wound-wait, an older requester always waits for a younger holder

914. Why do the wait-die and wound-wait schemes avoid starvation? Understanding

A. A rolled-back transaction gets a new, larger timestamp each time it restarts
B. Transactions are never rolled back in either scheme
C. Locks are always granted in first-come, first-served order
D. A rolled-back transaction restarts with its original timestamp, so it eventually becomes the oldest

915. In multiple-granularity locking (Silberschatz's IS, IX, S, SIX, X modes), which mode can another transaction be granted on a node on which an IX lock is already held? Understanding

A. IS
B. S
C. SIX
D. X

916. T1 holds a shared lock on A. T2 then requests a shared lock on A, and after that T3 requests an exclusive lock on A. Using S/X locking, what happens? Application

A. Both T2 and T3 are granted their locks
B. T2 must wait; T3 is granted the lock
C. Both T2 and T3 must wait
D. T2 is granted the lock; T3 must wait

917. Under wait-die (timestamps: smaller = older), T1 with TS 5 requests a lock on an item held by T2 with TS 12. What happens? Application

A. T1 is rolled back (dies)
B. T2 is rolled back (wounded)
C. Both are rolled back
D. T1 waits

918. Under wound-wait (smaller timestamp = older), T1 with TS 5 requests a lock on an item held by T2 with TS 12. What happens? Application

A. T1 waits for T2
B. T1 is rolled back (dies)
C. Both transactions wait
D. T2 is rolled back (wounded) and T1 gets the lock

919. Under wait-die, T3 with TS 20 requests an item held by T1 with TS 8. What happens to T3? Application

A. T3 waits for T1 to release the item
B. T3 is rolled back and later restarts with timestamp 20
C. T1 is rolled back and T3 gets the item
D. T3 is rolled back and restarts with a new, larger timestamp

920. S(x) = shared lock, X(x) = exclusive lock, U(x) = unlock, r/w = read/write. Which transaction follows two-phase locking? Application

A. S(A) r(A) U(A) X(B) w(B) U(B)
B. X(A) w(A) U(A) S(B) r(B) U(B)
C. S(A) r(A) X(B) U(A) w(B) U(B)
D. S(A) r(A) U(A) S(B) r(B) U(B)

921. S(x) = shared lock, X(x) = exclusive lock, U(x) = unlock, C = commit. Which transaction follows strict two-phase locking? Application

A. X(A) w(A) S(B) r(B) U(B) C U(A)
B. X(A) w(A) S(B) r(B) U(A) U(B) C
C. X(A) w(A) U(A) S(B) r(B) C U(B)
D. S(B) r(B) U(B) X(A) w(A) C U(A)

922. Basic timestamp ordering: TS(T1) = 5, TS(T2) = 10, and R-timestamp(X) = W-timestamp(X) = 0 at the start. The schedule is r2(X), w1(X). What happens? Application

A. Both operations succeed and W-timestamp(X) becomes 5
B. r2(X) is rejected because TS(T2) > W-timestamp(X)
C. w1(X) is rejected and T1 is rolled back, because TS(T1) < R-timestamp(X) = 10
D. w1(X) is ignored and T1 continues

923. Basic timestamp ordering with TS(T1) = 3, TS(T2) = 7, TS(T3) = 5, and both timestamps of X starting at 0. After r1(X), r3(X), w2(X), what are R-timestamp(X) and W-timestamp(X)? Application

A. 3 and 7
B. 7 and 7
C. 5 and 5
D. 5 and 7

924. TS(T1) = 4 and TS(T2) = 9, and no transaction reads X. The schedule is w2(X), w1(X). Under timestamp ordering with Thomas' write rule, what happens to w1(X)? Application

A. It succeeds and overwrites T2's value
B. T1 is rolled back
C. It is ignored and T1 continues
D. T2 is rolled back so that T1's write can proceed

925. Is the schedule r1(A), r2(A), w1(A), w2(A) conflict serialisable? Application

A. Yes: it is equivalent to T1 then T2
B. Yes: it is equivalent to T2 then T1
C. No: the precedence graph has the cycle T1 -> T2 -> T1
D. Yes: two reads never conflict, so there is no edge

926. Which serial order is conflict equivalent to the schedule r2(B), w1(A), r3(A), w2(A), w3(C)? Application

A. T2, T1, T3
B. T1, T3, T2
C. T1, T2, T3
D. T3, T1, T2

927. T1 holds A and waits for B. T2 holds B and waits for C. T3 holds C and waits for A. T4 holds D and waits for A. Which transactions are deadlocked? Application

A. T1, T2 and T3 only
B. T1, T2, T3 and T4
C. T1 and T3 only
D. None; T4 breaks the cycle

928. With multiple-granularity locking on a hierarchy database > file F > record r, which locks must a transaction hold to update record r? Application

A. X on the database, X on F, X on r
B. IS on the database, IS on F, X on r
C. X on r only
D. IX on the database, IX on F, X on r

929. A = 100. T1 does A := A - 30 and T2 does A := A + 50. They run as r1(A), r2(A), w1(A), w2(A), and each writes the value it computed from its own read. What is the final value of A? Application

A. 120
B. 150
C. 70
D. 100

930. Consider the schedule r1(X), w2(X), w3(Y), w1(Y). Which statement is true? Analysis

A. It is not conflict serialisable, because the precedence graph has a cycle
B. It is conflict serialisable, and 2PL could produce it if T1 locked Y before releasing X
C. It is conflict serialisable only as T1, T2, T3
D. It is conflict serialisable (as T3, T1, T2), but basic 2PL could not have produced it

931. Basic timestamp ordering (no Thomas' rule), TS(T1) = 10, TS(T2) = 20, TS(T3) = 30, all item timestamps 0 at the start. Schedule: r1(A), w2(A), r3(A), w1(B), r2(B), w3(B), w1(A). Which is the first operation rejected? Analysis

A. r3(A)
B. r2(B)
C. w1(A)
D. w3(B)

932. TS(T1) = 1, TS(T2) = 2, TS(T3) = 3. T2 holds a lock on A. T1 then requests A, and after that T3 requests A. Which transaction is rolled back under wait-die, and which under wound-wait? Analysis

A. Wait-die: T1; wound-wait: T3
B. Wait-die: T3; wound-wait: T2
C. Wait-die: T2; wound-wait: T3
D. Wait-die: T3; wound-wait: T3

933. Classify the schedule w1(A), r2(A), c2, c1. Analysis

A. Conflict serialisable but not recoverable
B. Recoverable but not cascadeless
C. Strict, and therefore recoverable
D. Not conflict serialisable

934. Classify the schedule w1(A), r2(A), c1, c2. Analysis

A. Cascadeless but not recoverable
B. Not recoverable, because T2 reads uncommitted data
C. Strict
D. Recoverable but not cascadeless

935. How many serial schedules are conflict equivalent to r1(A), w2(A), r3(B), w3(B)? Analysis

A. 1
B. 3
C. 2
D. 6

936. Under strict 2PL, requests arrive in this order: T1 X(A); T2 S(B); T1 S(B); T2 X(A); T1 upgrades to X(B). What is the state after the last request? Analysis

A. T1 waits for T2, but T2 can finish
B. All requests are granted
C. Deadlock: T1 waits for T2 on B and T2 waits for T1 on A
D. T2 waits for T1, but T1 can finish

937. TS(T1) < TS(T2). Which schedule is accepted by timestamp ordering with Thomas' write rule but rejected by basic timestamp ordering? Analysis

A. w2(X) r1(X)
B. r2(X) w1(X)
C. r1(X) w2(X) w1(X)
D. w1(X) w2(X)

938. A wait-for graph has edges T1 -> T2, T2 -> T3, T3 -> T1, T3 -> T4 and T4 -> T3. Rolling back which single transaction removes every deadlock? Analysis

A. T3
B. T1
C. T2
D. T4

939. T1 and T2 both run 'read A, then write A' under 2PL, taking S(A) to read and upgrading to X(A) to write. They interleave as r1(A), r2(A), w1(A), w2(A). What happens? Analysis

A. The lost update occurs, because 2PL allows this interleaving
B. Deadlock: each holds S(A) and waits for the other to release it before upgrading
C. T1 upgrades and finishes, then T2 upgrades
D. T2 is automatically rolled back by 2PL

940. In the validation-based (optimistic) concurrency-control protocol, in what order does a transaction go through its phases? Understanding

A. Read, validation, write
B. Validation, read, write
C. Write, validation, read
D. Read, write, validation

941. For which workload is optimistic (validation-based) concurrency control the best fit? Understanding

A. Many transactions that update the same few hot items
B. Mostly read-only transactions where conflicts are rare
C. Long transactions that must never be restarted
D. Workloads where deadlocks must be detected quickly

942. What is the main trade-off when choosing lock granularity, for example record locks versus whole-table locks? Understanding

A. Finer granularity allows more concurrency but costs more locking overhead
B. Finer granularity reduces both concurrency and overhead
C. Coarser granularity allows more concurrency and more overhead
D. Granularity only affects deadlock detection, not concurrency

943. How does requiring every transaction to lock data items in one fixed global order (for example, alphabetical) deal with deadlock? Understanding

A. It detects deadlock after it happens
B. It prevents deadlock, because a cycle of waits cannot form
C. It removes the need for exclusive locks
D. It makes every schedule serial

944. After detecting a deadlock, how does the DBMS usually choose which transaction to roll back? Understanding

A. It always rolls back the oldest transaction
B. It picks a transaction on the cycle whose rollback costs the least
C. It rolls back every transaction on the cycle
D. It rolls back a transaction that is not on the cycle

945. What is the main practical drawback of conservative (static) two-phase locking? Understanding

A. It allows deadlocks
B. It does not guarantee conflict serialisability
C. It releases locks before commit, so it causes dirty reads
D. Each transaction must know and lock its whole read and write set before it starts

946. What does the lock manager do with a lock request that conflicts with a lock already held? Understanding

A. Grants it anyway and logs a warning
B. Rolls back the transaction that holds the lock
C. Ignores the request and lets the transaction proceed without a lock
D. Adds it to that item's queue in the lock table and makes the transaction wait

947. What is lock escalation? Understanding

A. Upgrading a shared lock to an exclusive lock on the same item
B. Giving an older transaction priority in the lock queue
C. Replacing many fine-grained locks held by a transaction with one coarser lock
D. Rolling back a transaction that holds too many locks

948. Silberschatz allows lock conversions under two-phase locking. Where may upgrades (S to X) and downgrades (X to S) take place? Understanding

A. Upgrades only in the growing phase, downgrades only in the shrinking phase
B. Upgrades only in the shrinking phase, downgrades only in the growing phase
C. Both in either phase
D. Neither; conversions are not allowed under 2PL

949. In multiple-granularity locking, what does a SIX lock on a node give the holder? Understanding

A. An exclusive lock on the node and shared locks on its children
B. A shared lock that will be upgraded automatically at commit
C. Intention to take shared locks only, on some descendants
D. A shared lock on the whole subtree plus the right to take exclusive locks lower down

950. In multiple-granularity locking, in what order must a transaction acquire and release locks on the hierarchy? Understanding

A. Acquire from the leaves up to the root; release from the root down
B. Acquire and release in any order, as long as 2PL holds
C. Acquire from the root down to the leaves; release from the leaves up to the root
D. Acquire from the root down; release from the root down

951. In multiple-granularity locking, which modes requested by another transaction are compatible with an S lock already held on a node? Understanding

A. IS and IX
B. S only
C. IS and S
D. IS, IX and S

952. Record-level locking cannot stop a phantom, because the new row did not exist when the locks were taken. What technique does Silberschatz describe to prevent phantoms? Understanding

A. Wound-wait deadlock prevention
B. Thomas' write rule
C. Lock escalation from rows to pages
D. Index locking (or predicate locking), so inserts into the locked range must wait

953. A stream of transactions keeps getting S locks on A, so T1's X request on A waits forever. How does Silberschatz's lock manager prevent this starvation? Understanding

A. A lock is granted only if it is compatible with the held locks and no earlier-waiting request conflicts with it
B. It always grants X requests before S requests
C. It rolls back every transaction that holds an S lock
D. It assigns T1 a new, smaller timestamp

954. Some systems handle deadlock with lock timeouts: a transaction that waits longer than a limit is rolled back. What is the main weakness? Understanding

A. It cannot be implemented without timestamps
B. It makes every schedule non-serialisable
C. It may roll back transactions that were not deadlocked, and a real deadlock lasts until the timeout
D. It only works with conservative 2PL

955. What is the difference between total and partial rollback when breaking a deadlock? Understanding

A. Partial rollback undoes only the victim's reads; total rollback undoes its writes
B. Partial rollback aborts half of the transactions on the cycle
C. Partial rollback undoes the victim only as far as needed to release the lock in question; total rollback restarts it
D. There is no difference; both restart the victim from the beginning

956. Why can basic (non-strict) two-phase locking still produce cascading rollbacks? Understanding

A. A transaction may release an exclusive lock in its shrinking phase before it commits, so others can read its uncommitted write
B. Basic 2PL allows two transactions to hold X locks on the same item
C. Basic 2PL never releases any lock
D. Basic 2PL schedules are not conflict serialisable

957. Under multiversion timestamp ordering, what happens when a transaction reads an item? Understanding

A. The read waits until all older writers commit
B. The read always succeeds, returning the right version without waiting
C. The read is rejected if a younger transaction has written the item
D. The read always returns the newest version

958. Under snapshot isolation, two concurrent transactions both update the same row. What does the first-committer-wins rule do? Understanding

A. Both commit and the later value overwrites the earlier one
B. The older transaction always wins, whichever commits first
C. The second transaction waits and then re-reads its snapshot
D. The first to commit succeeds and the other is aborted when it tries to commit

959. In multiple-granularity locking, which mode requested by another transaction is compatible with a SIX lock already held on a node? Understanding

A. IS and IX
B. IS and S
C. IS only
D. None of the modes

960. Silberschatz's validation test for Ti checks each Tk with TS(Tk) < TS(Ti). Which pair of conditions lets Ti pass against Tk? Understanding

A. Tk finished before Ti started, or Tk finished before Ti's validation and Tk's write set does not meet Ti's read set
B. Tk started after Ti finished, or their read sets do not overlap
C. Tk's write set does not meet Ti's write set, whatever the timing
D. Ti started before Tk, or Ti has no writes

961. Under multiversion timestamp ordering, when is a writing transaction rolled back? Understanding
    A. Whenever an older transaction has written the item
    B. Whenever two transactions write the same item
    C. Never; writes always create a new version
    D. When a younger transaction has already read the version that the write should come after

962. Which statement about the tree protocol (a graph-based protocol for items arranged in a tree, using exclusive locks) is correct? Understanding
    A. It is a special case of two-phase locking
    B. It ensures conflict serialisability and freedom from deadlock, but not recoverability by itself
    C. It prevents cascading rollbacks without any extra rule
    D. It requires all locks to be taken before the transaction starts

963. Can the basic timestamp-ordering protocol produce a schedule that is not recoverable? Understanding
    A. No; timestamp ordering always yields strict schedules
    B. Yes; it checks only timestamps, so a transaction can read an uncommitted write and commit before the writer
    C. No; any conflict-serialisable schedule is recoverable
    D. Only if Thomas' write rule is used

964. Using multiple-granularity locking on database > file F > record r, which locks must a transaction hold to read record r? Application
    A. IS on the database, IS on F, S on r
    B. S on the database, S on F, S on r
    C. IX on the database, IX on F, S on r
    D. S on r only

965. With multiple-granularity locking, T1 holds an IS lock on file F. T2 now requests an X lock on F. What happens? Application
    A. T2 is granted the lock at once
    B. T1 is rolled back
    C. T2 waits, because X is incompatible with IS
    D. T2 is granted an IX lock instead

966. Under wound-wait (smaller timestamp = older), T3 with TS 30 requests a lock held by T1 with TS 10. What happens? Application
    A. T1 is rolled back
    B. T3 waits
    C. T3 is rolled back
    D. T3 is granted the lock

967. T1, T2 and T3 are deadlocked in one cycle. The cost of rolling back each is T1 = 40, T2 = 15 and T3 = 25. Using minimum-cost victim selection, which transaction is rolled back? Application
    A. T1
    B. T3
    C. T2
    D. All three

968. Basic timestamp ordering. Item X has W-timestamp 8 and R-timestamp 15. A transaction with TS 12 reads X. What happens? Application
    A. The read succeeds and R-timestamp(X) becomes 12
    B. The transaction is rolled back, because 12 < 15
    C. The read succeeds and R-timestamp(X) stays 15
    D. The read is ignored

969. T1 holds an exclusive lock on A, and T2 is waiting for that lock. Which edge goes into the wait-for graph? Application

A. T1 -> T2
B. A -> T1
C. T2 -> T1
D. T1 -> A

970. Validation-based protocol: T1 finished its write phase at time 10, and T2 started at time 12. T1 validated first. Does T2 pass validation against T1? Application

A. Yes, because T1 finished before T2 started, whatever their read and write sets
B. Only if T1's write set does not meet T2's read set
C. No, because T1 has the smaller timestamp
D. No, because the two transactions accessed the same items

971. With multiple-granularity locking, a transaction reads every record of file F and updates a few of them. Which lock on F is the best choice? Application

A. SIX
B. IX
C. S
D. IS

972. S(x) = shared lock, X(x) = exclusive lock, U(x) = unlock, C = commit. Which transaction follows basic two-phase locking but NOT strict two-phase locking? Application

A. X(A) w(A) S(B) U(A) r(B) U(B) C
B. S(B) r(B) X(A) w(A) C U(A) U(B)
C. X(A) w(A) U(A) S(B) r(B) U(B) C
D. S(A) r(A) U(A) X(A) w(A) C U(A)

973. Basic timestamp ordering with TS(T1) = 10, TS(T2) = 20, TS(T3) = 30; all item timestamps start at 0. The schedule is r2(X), r1(X), w3(X), r2(Y), w1(Y). What happens to w1(Y)? Application

A. It succeeds and W-timestamp(Y) becomes 10
B. It is rejected and T1 is rolled back, because TS(T1) = 10 < R-timestamp(Y) = 20
C. It is ignored under Thomas' write rule
D. It succeeds, but T2 is rolled back

974. Timestamp ordering with Thomas' write rule, TS(T1) = 10, TS(T2) = 20, TS(T3) = 30, all item timestamps 0 at the start. Schedule: w3(A), r2(B), w2(A), w1(B). What happens to w2(A) and w1(B)? Application

A. Both writes are ignored
B. w2(A) is rejected; w1(B) is ignored
C. Both writes succeed
D. w2(A) is ignored; w1(B) is rejected and T1 is rolled back

975. Under strict 2PL, requests arrive in this order: T1 S(A); T2 S(A); T3 X(A); T2 S(B). Which requests are granted? Application

A. All four requests
B. T1 S(A) and T3 X(A); T2 waits
C. T1 S(A), T2 S(A) and T2 S(B); T3 waits for T1 and T2
D. T1 S(A) only; the rest wait

976. Under two-phase locking, T1 reaches its lock point at step 6, T2 at step 3 and T3 at step 9. To which serial order is the schedule conflict equivalent? Application

A. T1, T2, T3
B. T2, T1, T3
C. T3, T1, T2
D. It depends on the commit order only

977. Multiversion timestamp ordering. Item Q has versions with write timestamps 5, 10 and 20. A transaction with TS 15 reads Q. Which version does it get? Application

A. The version with write timestamp 20
B. The version with write timestamp 10
C. The version with write timestamp 5
D. None; the transaction is rolled back

978. T1 and T2 hold S(A). T3's request for X(A) is waiting. Now T4 requests S(A). Under Silberschatz's starvation-free granting rule, what happens to T4? Application

A. T4 is granted S(A) at once
B. T3 is rolled back and T4 is granted S(A)
C. T1 and T2 are rolled back
D. T4 waits behind T3

979. Can basic two-phase locking produce the schedule r1(A), w2(A), r2(B), w1(B)? Application

A. No: it is not conflict serialisable (T1 -> T2 on A, T2 -> T1 on B), and every 2PL schedule is
B. Yes, if T1 releases A before T2 writes it
C. Yes, but only under strict 2PL
D. No, but only because T2 reads B before T1 writes it

980. Basic timestamp ordering (no Thomas' rule), TS(T1) = 10, TS(T2) = 20, TS(T3) = 30, all item timestamps 0 at the start. Schedule: r3(B), w1(A), r2(A), w2(B), w3(C), r1(C). Which is the first operation rejected? Application

A. r1(C)
B. w2(B)
C. r2(A)
D. w3(C)

981. Validation-based protocol: T1 starts at 2, validates at 5 and finishes its writes at 10, with write set {X}. T2 starts at 3, validates at 9 and reads {Y}. Does T2 pass validation against T1? Application

A. Yes: T1's write set and T2's read set are disjoint
B. No: T1 had not finished writing before T2 validated, so neither condition holds
C. Yes: T1 validated before T2
D. No: T2 started after T1 started

982. Multiversion timestamp ordering. Item Q has version Q5 (write timestamp 5, read timestamp 18) and version Q20 (write timestamp 20). A transaction with TS 12 writes Q. What happens? Application

A. It is rolled back, because a transaction with TS 18 has already read Q5
B. A new version Q12 is created
C. It overwrites version Q5
D. The write is ignored, as in Thomas' write rule

983. Under strict 2PL, requests arrive in this order: T1 S(A); T2 S(A); T3 X(B); T3 X(A); T1 X(B). Which transactions are deadlocked? Application

A. T1, T2 and T3
B. T1 and T3 only
C. T2 and T3 only
D. No transactions

984. Wound-wait with TS(T1) = 5, TS(T2) = 10, TS(T3) = 15. T3 holds A and T2 holds B. Then T2 requests A, and after that T1 requests B. Which transactions are rolled back? Application

A. T3 only
B. T2 only
C. None; T1 and T2 wait
D. T3 and T2

985. Long read-only reports keep blocking short update transactions under strict 2PL. Which change helps most without weakening isolation for the reports? Analysis

A. Switch to conservative 2PL
B. Let the reports run at READ UNCOMMITTED
C. Use wait-die deadlock prevention
D. Use multiversion concurrency control so reports read a snapshot and never block writers

986. Procedure P1 locks Orders then Payments, and procedure P2 locks Payments then Orders. They often deadlock. What is the simplest prevention fix? Analysis

A. Make both procedures lock Orders before Payments
B. Switch both to strict 2PL
C. Increase the size of the lock table
D. Run both at READ COMMITTED

987. A student draws a cycle in a schedule's precedence graph and concludes, 'these transactions are deadlocked'. What is wrong? Analysis

A. A precedence-graph cycle means the schedule is not conflict serialisable; deadlock is shown by a cycle in the wait-for graph
B. Nothing; the two graphs are the same
C. A precedence-graph cycle means the schedule is not recoverable
D. Deadlock is shown by the absence of a cycle

988. A nightly job reads 95% of the records in one large file and updates none of them. Other daytime transactions are not running. Which locking choice is most sensible? Analysis

A. An S lock on each record it reads
B. An X lock on the file
C. IS on the file and no record locks
D. One S lock on the file

989. TS(T1) = 1 and TS(T2) = 2. The schedule r1(B), r2(A), w1(A) is conflict serialisable, yet basic timestamp ordering rejects w1(A). Why? Analysis

A. TO rejects every schedule in which two transactions touch the same item
B. r1(B) and r2(A) conflict
C. TS(T1) is smaller than W-timestamp(A)
D. Its only equivalent serial order is T2, T1, but TO accepts only schedules equivalent to timestamp order T1, T2

990. A wait-for graph has edges T1 -> T2, T2 -> T1, T2 -> T3, T3 -> T4, T4 -> T5 and T5 -> T3. What is the minimum number of transactions that must be rolled back to remove every deadlock? Analysis

A. 1
B. 3
C. 2
D. 5

991. Two doctors are on call; the rule is that at least one must stay on call. Under snapshot isolation, each doctor's transaction checks that 2 are on call and then takes only itself off call. Both commit. What went wrong? Analysis

    A. Lost update on the same row
    B. A dirty read of an uncommitted row
    C. Write skew: each read the same snapshot and updated a different row, so first-committer-wins saw no conflict
    D. A deadlock that was resolved incorrectly

992. Two transactions deadlock under strict 2PL. A teammate proposes switching to rigorous 2PL to stop it. What is the best assessment? Analysis

    A. It will not help: rigorous 2PL holds locks even longer and can still deadlock; use lock ordering, conservative 2PL or wait-die/wound-wait
    B. It will fix it, because rigorous 2PL is deadlock-free
    C. It will fix it, because rigorous 2PL serialises transactions by commit order
    D. It will help only if timeouts are disabled

993. Under timestamp ordering, a long report transaction is rolled back again and again because younger transactions keep touching its items first, and each restart gets a new timestamp. What is happening, and what does Silberschatz suggest? Analysis

    A. Deadlock; build a wait-for graph and pick a victim
    B. Starvation; temporarily block conflicting transactions so the repeatedly restarted one can finish
    C. Cascading rollback; switch to strict 2PL
    D. A phantom; use index locking

994. Under 2PL with S/X locks, requests arrive in this order: T1 X(A); T2 S(B); T3 S(B); T2 S(A); T3 X(A); T1 S(B). What is the state? Analysis

    A. Deadlock between T1 and T2
    B. Deadlock between T1, T2 and T3
    C. No deadlock: T2 and T3 wait for T1, and T1 has all its locks
    D. T1 waits for T2 and T3 on B

995. Which of these schedules could basic two-phase locking (with S/X locks) produce? Analysis

    A. r1(A) w2(A) r3(B) w1(B)
    B. r1(A) w2(A) r2(B) w1(B)
    C. w1(A) w2(A) w3(B) w1(B)
    D. r1(A) w1(A) r2(A) w2(B) r1(C)

996. TS(T1) = 5, TS(T2) = 10, TS(T3) = 15. T1 holds A, T2 holds B and T3 holds C. Requests then arrive in this order: T3 requests B, T2 requests A, T1 requests C. Which transactions are rolled back under each scheme? Analysis

    A. Wait-die: T3 only; wound-wait: T3 and T2
    B. Wait-die: none; wound-wait: T2 and T3
    C. Wait-die: T1; wound-wait: T3
    D. Wait-die: T3 and T2; wound-wait: T3 only

997. TS(T1) = 1, TS(T2) = 2, TS(T3) = 3, all item timestamps 0 at the start. Schedule: r1(B), r2(A), w3(A), w2(A), w3(B). Which statement is correct? Analysis

    A. Both versions accept every operation unchanged
    B. Thomas' rule rolls back T3, because its write came before T2's
    C. Both versions reject w2(A), because R-timestamp(A) = 2
    D. Basic TO rolls back T2 at w2(A); with Thomas' rule w2(A) is ignored and the result is equivalent to T1, T2, T3

998. Validation-based protocol. T1: start 1, validate 4, finish 6, writes {A}. T2: start 2, validate 7, finish 9, reads {A, C}. T3: start 5, validate 8, finish 10, reads {B}. They validate in the order T1, T2, T3. Which transactions pass? Analysis
    A. All three
    B. T1 only
    C. T1 and T3; T2 fails
    D. T1 and T2; T3 fails

999. With multiple-granularity locking, T1 holds SIX on file F. T2 wants to read one record of F, and T3 wants to update a different record of F. What happens? Analysis
    A. Both proceed, because they use different records from T1's
    B. T2 proceeds (IS on F is compatible with SIX); T3 waits (IX on F is not)
    C. Both wait, because SIX is compatible with nothing
    D. T3 proceeds and T2 waits

Recovery Management MCQs for NSCT

1000. In Silberschatz's notation, what does the log record <Ti, X, V1, V2> store? Understanding
    A. Transaction Ti read X twice and got V1 and then V2
    B. Transaction Ti wrote data item X; V1 is the old value and V2 the new value
    C. Ti holds locks on items V1 and V2 of table X
    D. Ti wrote X; V1 is the new value and V2 the old value

1001. What does the write-ahead logging (WAL) rule require? Understanding
    A. A data page must be written to disk before its log records
    B. The log records for a data page must reach stable storage before that page is written to disk
    C. Every page a transaction changed must be written to disk before it commits
    D. The log must be written only at checkpoints

1002. What is the main purpose of taking a checkpoint? Understanding
    A. To make a backup copy of the whole database for media failure
    B. To release all locks held by active transactions
    C. To commit every active transaction
    D. To limit how much of the log must be scanned and redone after a crash

1003. Which kind of failure needs an archival copy (database dump) of the database to recover from? Understanding
    A. A disk (media) failure that destroys the stored data
    B. A transaction that divides by zero
    C. A system crash that wipes main memory
    D. A deadlock detected by the lock manager

1004. The deferred update technique (no data written to the database until commit) is known as which recovery algorithm? Understanding
    A. UNDO/NO-REDO
    B. NO-UNDO/REDO
    C. UNDO/REDO
    D. NO-UNDO/NO-REDO

1005. With immediate update, where pages may reach disk before commit and commit does not force pages, which operations can recovery need? Understanding

A. Undo only
B. Both undo and redo
C. Redo only
D. Neither undo nor redo

1006. If the buffer manager uses a steal policy, which recovery operation becomes necessary? Understanding

A. Redo
B. Undo
C. Neither undo nor redo
D. Only archival restore

1007. If the buffer manager uses a no-force policy, which recovery operation becomes necessary? Understanding

A. Undo
B. Redo
C. Neither undo nor redo
D. Shadow-page swapping

1008. Why must redo operations be idempotent? Understanding

A. The system may crash during recovery, so the same redo can be applied more than once
B. Redo is applied only to transactions that never committed
C. Idempotent operations use less log space
D. The undo phase runs before the redo phase

1009. At what point is a transaction considered committed in log-based recovery? Understanding

A. When its commit log record has been written to stable storage
B. When all its data pages have been written to disk
C. When it releases its first lock
D. When the next checkpoint is taken

1010. In UNDO/REDO recovery, in what direction is the log processed for each operation? Understanding

A. Undo goes forward; redo goes backward
B. Undo goes backward (newest record first); redo goes forward (oldest first)
C. Both go forward
D. Both go backward

1011. Which recovery technique, in a single-user setting, needs neither undo nor redo of log records? Understanding

A. Deferred update
B. Immediate update with a no-force policy
C. ARIES
D. Shadow paging

1012. What makes a fuzzy checkpoint different from a simple checkpoint? Understanding

A. No checkpoint record is written to the log
B. All active transactions are aborted at the checkpoint
C. It writes only the log, never any data pages
D. Transactions may resume before all modified buffers have been written to disk

1013. What are the three phases of ARIES recovery, in order? Understanding

A. Redo, analysis, undo
B. Undo, redo, analysis
C. Analysis, redo, undo
D. Analysis, undo, redo

1014. Why does ARIES write a compensation log record (CLR) each time it undoes an update? Understanding

A. So the original update can be redone later
B. So the transaction can be committed after the undo
C. So the undo is logged and never has to be repeated if the system crashes again
D. So the checkpoint can be skipped

1015. How is stable storage approximated in practice? Understanding

A. By keeping the log in main memory
B. By writing the log to a single hard disk
C. By using an uninterruptible power supply for RAM
D. By keeping copies on several non-volatile devices with independent failure modes

1016. The system uses immediate update with UNDO/REDO recovery and crashes after the log below. Which actions does recovery take? Application

```
<T1 start>
<T1, A, 100, 150>
<T1 commit>
<T2 start>
<T2, B, 200, 250>
--- crash ---
```

A. Undo T1 and redo T2
B. Redo T1 and undo T2
C. Redo both T1 and T2
D. Undo both T1 and T2

1017. The system uses deferred update (NO-UNDO/REDO), so log records hold new values only. After the crash below, what does recovery do? Application

```
<T1 start>
<T1, A, 150>
<T1 commit>
<T2 start>
<T2, B, 250>
--- crash ---
```

A. Redo T1 and undo T2
B. Redo both T1 and T2
C. Undo T2 only
D. Redo T1; ignore T2, since nothing it wrote reached the database

1018. Immediate update, UNDO/REDO, with a checkpoint that flushes all buffers (Elmasri-Navathe convention). After this crash, which transactions are redone and which are undone? Application

```
<T1 start>
<T1, A, 10, 20>
<T1 commit>
<T2 start>
<T2, B, 30, 40>
<T3 start>
<T3, C, 50, 60>
<checkpoint {T2, T3}>
<T2, A, 20, 25>
<T2 commit>
<T4 start>
<T4, D, 70, 80>
<T4 commit>
<T3, B, 40, 45>
--- crash ---
```

A. Redo T1, T2 and T4; undo T3
B. Redo T4; undo T2 and T3
C. Redo T2 and T4; undo T3; T1 needs nothing
D. Redo T2 and T4; undo T1 and T3

1019. For this log, what is the value of B after recovery finishes? Application

```
<T1 start>
<T1, A, 10, 20>
<T1 commit>
<T2 start>
<T2, B, 30, 40>
<T3 start>
<T3, C, 50, 60>
<checkpoint {T2, T3}>
<T2, A, 20, 25>
<T2 commit>
<T4 start>
<T4, D, 70, 80>
<T4 commit>
<T3, B, 40, 45>
--- crash ---
```

A. 45
B. 30
C. 40
D. 25

1020. The last record in the log before a crash is <T2, B, 40, 70>, and T2 never committed. Under UNDO/REDO recovery, what value is written to B? Application

A. 70
B. 0
C. 40
D. B is left unchanged

1021. Silberschatz's recovery algorithm starts the undo-list from the checkpoint's list and updates it while scanning forward. After the redo phase on this log, what is the undo-list? Application

```
<T1 start>
<T1, A, 0, 10>
<T2 start>
<T2, B, 0, 20>
<checkpoint {T1, T2}>
<T3 start>
<T1, C, 0, 30>
<T1 commit>
<T3, A, 10, 40>
<T4 start>
<T4, D, 0, 50>
<T4, D, 0>
<T4 abort>
--- crash ---
```

A. {T1, T2}
B. {T2, T3, T4}
C. {T3}
D. {T2, T3}

1022. With Silberschatz's algorithm (repeating history from the last checkpoint), how many log records does the redo phase replay on this log? Application

```
<T1 start>
<T1, A, 0, 10>
<T2 start>
<T2, B, 0, 20>
<checkpoint {T1, T2}>
<T3 start>
<T1, C, 0, 30>
<T1 commit>
<T3, A, 10, 40>
<T4 start>
<T4, D, 0, 50>
<T4, D, 0>
<T4 abort>
--- crash ---
```

A. 4
B. 2
C. 3
D. 6

1023. A buffer manager uses no-steal and force policies. Which recovery operations does restart after a crash need? Application

A. Undo only
B. Redo only
C. Both undo and redo
D. Neither undo nor redo

1024. Immediate update, UNDO/REDO. How many write operations must be undone after this crash? Application

```
<T1 start>
<T1, A, 5, 10>
<T2 start>
<T2, B, 15, 20>
<T1, C, 25, 30>
<T1 commit>
<checkpoint {T2}>
<T3 start>
<T3, A, 10, 40>
<T2, C, 30, 35>
<T3 commit>
<T2, B, 20, 45>
--- crash ---
```

A. 2
B. 1
C. 4
D. 3

1025. For this log, what are A, B and C after recovery? Application

```
<T1 start>
<T1, A, 5, 10>
<T2 start>
<T2, B, 15, 20>
<T1, C, 25, 30>
<T1 commit>
<checkpoint {T2}>
<T3 start>
<T3, A, 10, 40>
<T2, C, 30, 35>
<T3 commit>
<T2, B, 20, 45>
--- crash ---
```

A. A = 40, B = 45, C = 35
B. A = 10, B = 15, C = 30
C. A = 40, B = 15, C = 30
D. A = 40, B = 20, C = 30

1026. Deferred update (NO-UNDO/REDO) with a checkpoint. Which transactions are redone after this crash? Application

```
<T1 start>
<T1, A, 10>
<T1 commit>
<checkpoint>
<T2 start>
<T2, B, 20>
<T3 start>
<T3, C, 30>
<T2 commit>
<T4 start>
<T4, A, 40>
--- crash ---
```

A. T1 and T2
B. T2, T3 and T4
C. T2 only
D. T2, with T3 and T4 undone

1027. Which transactions in this log are losers (must be rolled back) at restart? Application

```
<T1 start>
<T1, A, 5, 10>
<T2 start>
<T2, B, 15, 20>
<T1, C, 25, 30>
<T1 commit>
<checkpoint {T2}>
<T3 start>
<T3, A, 10, 40>
<T2, C, 30, 35>
<T3 commit>
<T2, B, 20, 45>
--- crash ---
```

A. T2 only
B. T1 and T2
C. T2 and T3
D. None; the checkpoint protects them

1028. Using Silberschatz's algorithm, which records does the undo phase append to the log for T3, in order? Application

```
<T1 start>
<T1, C, 5, 6>
<T1 commit>
<T3 start>
<T3, A, 10, 20>
<T3, B, 30, 40>
--- crash ---
```

A. <T3, A, 10>, <T3, B, 30>, <T3 abort>
B. <T3, B, 30>, <T3, A, 10>, <T3 abort>
C. <T3, B, 40>, <T3, A, 20>, <T3 abort>
D. <T3 abort> only

1029. In this log, T4 aborted before the crash. Under Silberschatz's algorithm, what happens to T4 during recovery? Application

```
<T1 start>
<T1, A, 0, 10>
<T2 start>
<T2, B, 0, 20>
<checkpoint {T1, T2}>
<T3 start>
<T1, C, 0, 30>
<T1 commit>
<T3, A, 10, 40>
<T4 start>
<T4, D, 0, 50>
<T4, D, 0>
<T4 abort>
--- crash ---
```

A. It is put on the undo-list and undone in the undo phase
B. It is ignored entirely by both phases
C. Its update and redo-only record are replayed in the redo phase, and it is not undone again
D. It is redone and then committed

1030. A buffer manager writes a page that T1 changed (A from 10 to 99) to disk, then the system crashes before the log record <T1, A, 10, 99> reached stable storage. T1 had not committed. What is the consequence? Analysis

A. A = 99 stays on disk with no log record to undo it, so the database is inconsistent
B. Recovery redoes T1, because its page is on disk
C. Nothing, because uncommitted transactions are ignored at restart
D. Recovery undoes A using the after-image 99

1031. Immediate update, UNDO/REDO, with a checkpoint that flushes all buffers (Elmasri-Navathe). Which update records must be redone? Analysis

```
<T1 start>
<T1, A, 10, 20>
<T1 commit>
<T2 start>
<T2, B, 30, 40>
<T3 start>
<T3, C, 50, 60>
<checkpoint {T2, T3}>
<T2, A, 20, 25>
<T2 commit>
<T4 start>
<T4, D, 70, 80>
<T4 commit>
<T3, B, 40, 45>
--- crash ---
```

A. <T2, A, 20, 25> and <T4, D, 70, 80>
B. <T2, B, 30, 40>, <T2, A, 20, 25> and <T4, D, 70, 80>
C. <T1, A, 10, 20>, <T2, A, 20, 25> and <T4, D, 70, 80>
D. <T2, A, 20, 25>, <T4, D, 70, 80> and <T3, B, 40, 45>

1032. Recover this log with Silberschatz's algorithm (redo phase repeats history from the checkpoint, then the undo phase rolls back the undo-list). Which statement is true? Analysis

```
<T1 start>
<T1, A, 10, 20>
<T1 commit>
<T2 start>
<T2, B, 30, 40>
<T3 start>
<T3, C, 50, 60>
<checkpoint {T2, T3}>
<T2, A, 20, 25>
<T2 commit>
<T4 start>
<T4, D, 70, 80>
<T4 commit>
<T3, B, 40, 45>
--- crash ---
```

A. The redo phase skips T3's records because T3 did not commit
B. The undo phase stops at the checkpoint, so <T3, C, 50, 60> is not undone
C. T2 is undone because it was active at the checkpoint
D. The redo phase replays <T3, B, 40, 45> even though T3 is a loser, and the undo phase then restores B = 40 and C = 50

1033. T1 (committed) added 10 to A, changing it from 100 to 110, and the page had already reached disk before the crash. Compare redoing a physical record <T1, A, 100, 110> with blindly redoing a logical record 'A := A + 10'. What value does each give? Analysis

A. Both give 110
B. Both give 120
C. Physical redo gives 100; logical redo gives 110
D. Physical redo gives 110; logical redo gives 120, which is wrong

1034. Immediate update with UNDO/REDO. After recovery from this crash, what are X and Y? Analysis

```
<T1 start>
<T1, X, 1, 2>
<T1 commit>
<T2 start>
<T2, X, 2, 3>
<T2 commit>
<T3 start>
<T3, X, 3, 4>
<checkpoint {T3}>
<T3, Y, 7, 8>
--- crash ---
```

A. X = 4, Y = 8
B. X = 3, Y = 7
C. X = 2, Y = 7
D. X = 1, Y = 7

1035. T1 updates A without committing, T2 reads A and writes B, and then T1 aborts. Which statement is correct about recovery? Analysis

A. T2 must also be rolled back (cascading rollback); strict 2PL would have prevented this
B. Only T1 is rolled back; T2's write to B is unaffected
C. T1's abort is ignored because T2 depends on it
D. Checkpoints prevent this situation

1036. Deferred update, and the disk held A = 0, B = 0, C = 0 before T1 started. Log records hold new values only. After recovery from this crash, what are A, B and C? Analysis

```
<T1 start>
<T1, A, 10>
<T1 commit>
<checkpoint>
<T2 start>
<T2, B, 20>
<T3 start>
<T3, C, 30>
<T2 commit>
<T4 start>
<T4, A, 40>
--- crash ---
```

A. A = 10, B = 20, C = 0
B. A = 40, B = 20, C = 30
C. A = 10, B = 20, C = 30
D. A = 0, B = 20, C = 0

1037. With Silberschatz's algorithm, how far back must the undo phase read in this log? Analysis

```
<T1 start>
<T1, A, 5, 10>
<T2 start>
<T2, B, 15, 20>
<T1, C, 25, 30>
<T1 commit>
<checkpoint {T2}>
<T3 start>
<T3, A, 10, 40>
<T2, C, 30, 35>
<T3 commit>
<T2, B, 20, 45>
--- crash ---
```

A. Back to the checkpoint record only
B. Back to <T3 start>
C. Back to <T2 start>, which lies before the checkpoint
D. To the very first record, <T1 start>

1038. A designer proposes a force policy so that recovery never needs redo. What is the main drawback? Analysis

A. Undo becomes impossible
B. The log can no longer contain before-images
C. Every commit must wait for all of the transaction's changed pages to be written, which slows commits and repeats writes of hot pages
D. Checkpoints can no longer be taken

1039. After recovery from this crash, how many of the items A, B and C end up with a value different from the last new value logged for them? Analysis

```
<T1 start>
<T1, A, 5, 10>
<T2 start>
<T2, B, 15, 20>
<T1, C, 25, 30>
<T1 commit>
<checkpoint {T2}>
<T3 start>
<T3, A, 10, 40>
<T2, C, 30, 35>
<T3 commit>
<T2, B, 20, 45>
--- crash ---
```

A. 2
B. 1
C. 3
D. 0

1040. A power cut stops the database server. Main memory, including the buffer pool, is lost, but the disks are undamaged. How do textbooks classify this failure? Understanding

A. A system crash
B. A transaction failure
C. A disk (media) failure
D. A logical error in the application

1041. Which two ACID properties is the recovery manager mainly responsible for? Understanding

A. Consistency and isolation
B. Isolation and durability
C. Atomicity and durability
D. Atomicity and isolation

1042. Elmasri-Navathe call the old value of an item the BFIM and the new value the AFIM. Which one does an undo operation write back to the database? Understanding

A. The BFIM (before image)
B. The AFIM (after image)
C. The AFIM of the next transaction that wrote the item
D. The value stored in the last archival dump

1043. What does a no-steal buffer policy mean? Understanding

A. Every page a transaction changed must be written to disk when it commits
B. A page may be written to disk at any time, even before the transaction commits
C. A page changed by a transaction cannot be written to disk until that transaction commits
D. A page that has been written to disk can never be read into the buffer again

1044. What does a force buffer policy require? Understanding

A. Log records are written to disk only after the data pages they describe
B. All pages changed by a transaction are written to disk before the transaction is allowed to commit
C. A page holding uncommitted changes may never be written to disk
D. Checkpoints are taken after every transaction commits

1045. In shadow paging, how does a transaction commit? Understanding

A. Every updated page is copied back over its original location on disk
B. The current page table is written to disk and atomically made the new shadow page table
C. The shadow page table is updated one entry at a time as each page changes
D. A commit record is written and all old page versions are redone from the log

1046. Elmasri-Navathe distinguish in-place updating from shadowing. Why does in-place updating require the old value to be logged before the page is written? Understanding

A. The new value is written to a different location, so the log must record where it went
B. In-place updating writes the page only at commit, so the old value is needed for redo
C. The new value overwrites the old one at the same disk location, so the log is the only place the old value survives
D. The old value is needed so that the page can be compressed before writing

1047. In Silberschatz's scheme, when is the log record <Ti abort> written? Understanding

A. As soon as Ti decides to roll back, before any undo is done
B. When Ti is chosen as a deadlock victim, before it releases any locks
C. At the next checkpoint, for every transaction that has not committed
D. After all of Ti's updates have been undone, to show that its rollback is complete

1048. Which of these loses its contents in a system crash? Understanding

A. The log file on disk
B. The data files on disk
C. The database buffer pool in main memory
D. An archival dump on tape

1049. What is the main purpose of a remote backup system in database recovery? Understanding

A. To speed up queries by spreading reads across two sites
B. To remove the need for a log at the primary site
C. To let two sites update the same data independently
D. To keep processing going at another site if the primary site is destroyed by a disaster

1050. What does the dirty bit of a buffer page indicate? Understanding

A. The page contains data from a transaction that has aborted
B. The page has been modified since it was read from disk, so it must be written back before its frame is reused
C. The page is pinned and cannot be replaced
D. The page has been written to the log but not to the database

1051. Why does a DBMS collect log records in a log buffer in main memory instead of writing each one to disk immediately? Understanding

A. Log records in memory are safe from system crashes
B. The log buffer lets the system skip writing log records for committed transactions
C. It allows data pages to be written before their log records
D. Writing a whole block of log records at once is far cheaper than one disk write per record

1052. In ARIES, which structure records the pages that are modified in the buffer but not yet written to disk? Understanding

A. The transaction table
B. The master record
C. The shadow page table
D. The dirty page table

1053. Elmasri-Navathe describe an immediate-update scheme in which all of a transaction's updates are written to disk before it commits. Which recovery algorithm does that give? Understanding

A. NO-UNDO/REDO
B. UNDO/NO-REDO
C. UNDO/REDO
D. NO-UNDO/NO-REDO

1054. What is the main practical limitation of deferred update (NO-UNDO/REDO)? Understanding

A. Committed transactions can never be redone after a crash
B. Each update must be written to disk twice before commit
C. It cannot be used together with checkpoints
D. Every page a transaction changes must stay in the buffer until it commits, so large transactions can run out of buffer space

1055. Which is a recognised disadvantage of shadow paging? Understanding

A. Every update must be undone from the log after a crash
B. The shadow page table is lost in a system crash
C. Updated pages move to new disk locations, so related pages lose their physical clustering and old pages need garbage collection
D. It forces the log to be scanned from the beginning at restart

1056. What is the purpose of the PageLSN stored in each page under ARIES? Understanding

A. It records the LSN of the first update that made the page dirty
B. It records which transaction currently holds a lock on the page
C. It records the LSN of the latest update applied to the page, so redo can skip log records that are already reflected on disk
D. It records the LSN of the last checkpoint taken before the page was read

1057. In ARIES, what does the UndoNextLSN field of a compensation log record (CLR) point to? Understanding

A. The log record that the CLR has just undone
B. The next checkpoint record in the log
C. The first log record written after the crash
D. The next log record of the same transaction that still has to be undone

1058. A disk failure destroys the database, but the log on a separate disk survives. What are the recovery steps? Understanding

A. Use the log alone to undo every transaction that was active at the time of the failure
B. Restore the most recent archival dump, then use the log to redo the transactions that committed after the dump
C. Restore the dump and discard the log, accepting the loss of later work
D. Redo every transaction in the log from the very first record, without a dump

1059. What is group commit? Understanding

A. Committing a group of transactions only if all of them succeed
B. Delaying the forced write of the log briefly so that the commit records of several transactions go to disk in one block write
C. Writing all dirty data pages to disk whenever a group of transactions commits
D. Taking a checkpoint after every fixed group of commits

1060. In a remote backup system, a transaction commits as soon as its commit record is on stable storage at the primary (one-safe). What is the risk? Understanding

A. Transactions cannot commit while the backup site is down
B. If the primary fails before the log reaches the backup, the backup takes over without those committed updates
C. Every commit must wait for a round trip to the backup site
D. The backup site may contain updates of transactions that never committed at the primary

1061. When Silberschatz's scheme rolls back a transaction during normal operation, it logs a redo-only record <Ti, Xj, V1> for each update it undoes. Why? Understanding

A. So that the rollback itself can be undone if the transaction restarts
B. So that the old value V1 can be used for a later archival dump
C. So that other transactions can read the old value without taking a lock
D. So that if the system later crashes, the redo phase replays the rollback as part of repeating history

1062. With fuzzy checkpointing, how does restart find the last checkpoint that can safely be used? Understanding

A. By scanning the log backward and using the first checkpoint record found, whether or not its flush finished
B. By reading the checkpoint record stored in each data page
C. By taking a fresh checkpoint as the first step of recovery
D. Through a pointer at a fixed place on disk that is updated only after all pages marked dirty at that checkpoint have been written

1063. Each ARIES log record stores PrevLSN, the LSN of the previous record written by the same transaction. What does this make possible? Understanding

A. Redo can skip pages that are already up to date on disk
B. The analysis phase can find the most recent checkpoint
C. Two transactions can share one log record
D. Undo can follow one transaction's records backward directly, without reading every log record in between

1064. Immediate update with UNDO/REDO recovery. What are X and Y after recovery from this crash? Application

```
<T1 start>
<T1, X, 20, 30>
<T2 start>
<T2, Y, 5, 15>
<T2 commit>
--- crash ---
```

A. X = 30, Y = 15
B. X = 20, Y = 5
C. X = 20, Y = 15
D. X = 30, Y = 5

1065. This log has no checkpoint. Under UNDO/REDO recovery, which transactions are redone and which are undone? Application

```
<T1 start>
<T1, A, 1, 2>
<T2 start>
<T1 commit>
<T2, B, 3, 4>
<T3 start>
<T3, C, 5, 6>
<T3 commit>
<T2, A, 2, 7>
--- crash ---
```

A. Redo T3 only; undo T1 and T2
B. Redo T1, T2 and T3
C. Redo T1; undo T2 and T3
D. Redo T1 and T3; undo T2

1066. A buffer manager uses no-steal and no-force policies. Which operations can restart after a crash need? Application

A. Undo only
B. Redo only
C. Both undo and redo
D. Neither undo nor redo

1067. A buffer manager uses steal and force policies. Which operations can restart after a crash need? Application

A. Redo only
B. Both undo and redo
C. Undo only
D. Neither undo nor redo

1068. During Silberschatz's undo phase, recovery undoes the log record <T4, Z, 7, 9>. Which record does it append to the log? Application

A. <T4, Z, 9>
B. <T4, Z, 7>
C. <T4, Z, 9, 7>
D. <T4 abort>

1069. T1 was rolled back during normal operation before the crash. Using Silberschatz's algorithm, what are A and B after recovery, and what does the undo phase do? Application

```
<T1 start>
<T1, A, 50, 80>
<T2 start>
<T2, B, 10, 20>
<T1, A, 50>
<T1 abort>
<T2 commit>
--- crash ---
```

A. A = 80, B = 20; the undo phase has nothing to undo
B. A = 50, B = 10; T2 is undone in the undo phase
C. A = 50, B = 20; the undo phase has nothing to undo
D. A = 50, B = 20; T1 is undone again in the undo phase

1070. Using Silberschatz's algorithm, what is the undo-list at the end of the redo phase for this log? Application

```
<T1 start>
<T2 start>
<T1, A, 1, 2>
<checkpoint {T1, T2}>
<T2, B, 3, 4>
<T2 commit>
<T3 start>
<T4 start>
<T3, C, 5, 6>
<T4, D, 7, 8>
<T4, D, 7>
<T4 abort>
<T5 start>
--- crash ---
```

A. {T1, T3, T5}
B. {T3, T5}
C. {T1, T3, T4, T5}
D. {T1, T2, T3, T5}

1071. After the ARIES analysis phase, the dirty page table holds P1 with RecLSN 120, P2 with RecLSN 95 and P3 with RecLSN 140. The last checkpoint record is at LSN 150. At which LSN does the redo phase begin? Application

A. 150
B. 120
C. 140
D. 95

1072. During ARIES redo, the log record at LSN 200 updates page P5. P5 is in the dirty page table with RecLSN 180, and when P5 is read from disk its PageLSN is 230. What does redo do with LSN 200? Application

A. Reapplies it, because 200 is at least the RecLSN of 180
B. Undoes it, because the page is newer than the log record
C. Skips it, because PageLSN 230 shows the update is already on the page
D. Reapplies it and then sets PageLSN to 200

1073. Immediate update with UNDO/REDO. T1 wrote A twice. What are A and B after recovery? Application

```
<T1 start>
<T1, A, 50, 60>
<T2 start>
<T2, B, 5, 6>
<T1, A, 60, 70>
<T2 commit>
--- crash ---
```

A. A = 60, B = 6
B. A = 70, B = 6
C. A = 50, B = 6
D. A = 50, B = 5

1074. Immediate update with UNDO/REDO. What are P and Q after recovery? Application

```
<T1 start>
<T1, P, 4, 8>
<T1 commit>
<T2 start>
<T2, P, 8, 12>
<T2, Q, 1, 3>
--- crash ---
```

A. P = 8, Q = 1
B. P = 4, Q = 1
C. P = 12, Q = 3
D. P = 8, Q = 3

1075. Elmasri-Navathe UNDO/REDO, where each checkpoint flushes all modified buffers. There are two checkpoints in this log. Which transactions must be redone? Application

```
<T1 start>
<T1, A, 1, 2>
<checkpoint {T1}>
<T2 start>
<T1 commit>
<T2, B, 3, 4>
<checkpoint {T2}>
<T3 start>
<T3, C, 5, 6>
<T3 commit>
--- crash ---
```

A. T1 and T3
B. T1, T2 and T3
C. T3 only
D. None of the transactions

1076. An archival dump finished at 09:00. T1 committed at 08:50, T2 started at 09:05 and committed at 09:20, and T3 started at 09:30 and was still running at 10:00 when the data disk failed. The log is on another disk. After restoring the dump, whose updates must be redone from the log? Application

A. T2 only
B. T1 and T2
C. T2 and T3
D. T1, T2 and T3

1077. Using Silberschatz's algorithm, how many log records does the undo phase append for this log? Application

```
<T1 start>
<T1, A, 3, 4>
<T2 start>
<T2, B, 6, 7>
<T1, C, 9, 1>
<T2, D, 2, 8>
--- crash ---
```

A. 6
B. 4
C. 5
D. 3

1078. Deferred update (NO-UNDO/REDO); log records hold new values only. Before T1 and T2 ran, the disk held A = 5, B = 1, C = 2. What are A, B and C after recovery? Application

```
<T1 start>
<T1, A, 15>
<T2 start>
<T2, B, 7>
<T1, A, 25>
<T1 commit>
<T2, C, 9>
--- crash ---
```

A. A = 15, B = 1, C = 2
B. A = 25, B = 7, C = 9
C. A = 5, B = 1, C = 2
D. A = 25, B = 1, C = 2

1079. Transaction T changes X on page P from 5 to 9 and then commits. Which order of disk writes obeys the write-ahead logging rule? Application

A. Write page P, then force <T, X, 5, 9>, then force <T commit>
B. Force <T, X, 5, 9>, then write page P, then force <T commit>
C. Force <T commit>, then force <T, X, 5, 9>, then write page P
D. Write page P, then force <T, X, 5, 9> and <T commit> together

1080. Using Silberschatz's algorithm, which records does the undo phase append, in order? Application

```
<T1 start>
<T1, X, 1, 2>
<T2 start>
<T2, Y, 3, 4>
<checkpoint {T1, T2}>
<T2, Z, 5, 6>
<T2 commit>
<T3 start>
<T3, W, 7, 8>
<T1, Y, 4, 9>
--- crash ---
```

A. <T1, Y, 4>, <T3, W, 7>, <T3 abort>, <T1, X, 1>, <T1 abort>
B. <T3, W, 7>, <T3 abort>, <T1, Y, 4>, <T1, X, 1>, <T1 abort>
C. <T1, Y, 4>, <T1, X, 1>, <T1 abort>, <T3, W, 7>, <T3 abort>
D. <T1, Y, 4>, <T3, W, 7>, <T3 abort>, <T1 abort>

1081. Using Silberschatz's algorithm, what are A, B, C and D after recovery? Application

```
<T1 start>
<T1, A, 10, 11>
<T2 start>
<T2, B, 20, 21>
<T2, B, 20>
<T2 abort>
<checkpoint {T1}>
<T3 start>
<T3, B, 20, 22>
<T1, C, 30, 31>
<T3 commit>
<T4 start>
<T4, D, 40, 41>
--- crash ---
```

A. A = 10, B = 22, C = 30, D = 40
B. A = 11, B = 22, C = 30, D = 40
C. A = 10, B = 20, C = 30, D = 40
D. A = 10, B = 22, C = 31, D = 41

1082. T1 was rolling back when the system crashed, and T2 had not committed. Which updates does the ARIES undo phase undo, in order? Application

```
LSN 10: T1 update A        (PrevLSN -)
LSN 20: T2 update B        (PrevLSN -)
LSN 30: T1 update C        (PrevLSN 10)
LSN 40: CLR for T1 undoing LSN 30, UndoNextLSN 10
--- crash ---
```

A. LSN 30, then LSN 20, then LSN 10
B. LSN 20, then LSN 10
C. LSN 10, then LSN 20
D. LSN 20 only

1083. After the ARIES analysis phase on this log, where does the redo phase start, and which transactions are losers? Application

```
LSN 50: checkpoint  transaction table {T1: LastLSN 40}
                    dirty page table   {P1: RecLSN 30}
LSN 60: T2 update P2
LSN 70: T1 update P1
LSN 80: T1 commit
LSN 90: T2 update P3
--- crash ---
```

A. Redo starts at LSN 50; T2 is the only loser
B. Redo starts at LSN 30; T1 and T2 are losers
C. Redo starts at LSN 60; T2 is the only loser
D. Redo starts at LSN 30; T2 is the only loser

1084. During the ARIES redo phase, which of these log records are actually reapplied? Application

```
Dirty page table after analysis: P1 RecLSN 20, P2 RecLSN 50
On-disk PageLSN:                 P1 = 40,  P2 = 30

LSN 20: update P1
LSN 30: update P3
LSN 40: update P1
LSN 50: update P2
LSN 60: update P2
--- crash ---
```

A. LSN 50 and LSN 60
B. All five records
C. LSN 40, LSN 50 and LSN 60
D. Every record except LSN 30

1085. A student recovers this log by setting A to 10, the oldest value logged for A. What is wrong with this? Analysis

```
<T1 start>
<T1, A, 10, 20>
<T1 commit>
<T2 start>
<T2, A, 20, 30>
--- crash ---
```

A. A should be 20; setting it to 10 also erases T1's committed update
B. Nothing is wrong; undo always restores the oldest value in the log
C. A should be 30, because T2's write was the last one before the crash
D. A should be 10, but only after T1 is redone first

1086. A DBMS uses a no-steal policy. A batch job updates more pages than the whole buffer pool can hold, all in one transaction. What happens? Analysis

A. The buffer fills with pages that cannot be written out, so the job cannot finish as a single transaction
B. The extra pages are written to disk early and undone later if needed
C. The DBMS switches the job to a force policy, so it finishes normally
D. The job finishes, but recovery will have to redo all of its pages

1087. To reduce latency, a system tells the client "committed" as soon as T's commit record is placed in the log buffer, before the buffer is forced to disk. The system then crashes. What can go wrong? Analysis

A. The commit record may be lost, so recovery treats T as a loser and undoes work the client was told was committed
B. T will be redone twice, so its updates are applied twice
C. Nothing, because the log buffer survives a system crash
D. Other transactions that read T's data will be cascaded back

1088. Silberschatz's redo phase replays the updates of loser transactions too, even though the undo phase will reverse them right after. Why is this a sensible design? Analysis

A. It is needed because losers' updates are never written to disk
B. It restores the exact state at the crash, so the undo phase can roll back losers just as it would in a normal rollback
C. It lets the undo phase skip transactions that were active at the checkpoint
D. It avoids the need to write any log records during undo

1089. A DBA wants to change checkpoints from once an hour to once a minute. What is the expected effect? Analysis

    A. Restart gets slower, because there are more checkpoint records to read
    B. Transactions no longer need to write commit records
    C. Undo is no longer needed for long-running transactions
    D. Restart after a crash gets faster, but normal processing pays more for the frequent checkpoints

1090. A designer logs only new values, <Ti, X, new>, to save log space, and keeps a steal/no-force buffer manager. What is the flaw? Analysis

    A. Committed transactions cannot be redone, because redo needs old values
    B. The checkpoint can no longer list the active transactions
    C. An uncommitted page written to disk cannot be undone, because the old value was never logged
    D. The log grows larger, not smaller

1091. The first recovery had logged <T2, C, 9> when the system crashed again. The second recovery uses Silberschatz's algorithm, whose undo phase ignores redo-only records. Which records does its undo phase append? Analysis

```
<T1 start>
<T1, A, 5, 6>
<T2 start>
<T2, B, 7, 8>
<T2, C, 9, 10>
<T1 commit>
--- crash ---
<T2, C, 9>
--- crash during recovery ---
```

    A. <T2, C, 9>, <T2, B, 7>, <T2 abort>
    B. <T2, B, 7>, <T2 abort>
    C. <T2, C, 10>, <T2, B, 8>, <T2 abort>
    D. <T2 abort> only

1092. The schedule was not strict: T2 overwrote A while T1 was still uncommitted. Both are losers. What is A after recovery, and why does the undo order matter? Analysis

```
<T1 start>
<T1, A, 10, 20>
<T2 start>
<T2, A, 20, 30>
<T2, B, 1, 2>
--- crash ---
```

    A. A = 20, because only T2's write of A needs undoing
    B. A = 20, because undo restores T1's before image first and then T2's
    C. A = 10, because undo scans backward, restoring T2's before image 20 and then T1's 10
    D. A = 30, because T2's write was the last one before the crash

1093. Compare how the two textbook algorithms treat this log. Which statement is true? Analysis

```
<T1 start>
<T1, X, 1, 2>
<T2 start>
<T2, Y, 3, 4>
<checkpoint {T1, T2}>
<T2, Z, 5, 6>
<T2 commit>
<T3 start>
<T3, W, 7, 8>
<T1, Y, 4, 9>
--- crash ---
```

A. T2 must be undone, because it was active when the checkpoint was taken
B. Silberschatz's redo phase replays <T3, W, 7, 8> and <T1, Y, 4, 9>, and its undo phase then reverses both
C. The undo scan can stop at the checkpoint, because every loser write comes after it
D. Elmasri-Navathe's method redoes T1, because T1 was active at the checkpoint

1094. With Silberschatz's algorithm, compare the values right after the redo phase with the final values. How many of A, B, C and D does the undo phase change? Analysis

```
<T1 start>
<T1, A, 10, 11>
<T2 start>
<T2, B, 20, 21>
<T2, B, 20>
<T2 abort>
<checkpoint {T1}>
<T3 start>
<T3, B, 20, 22>
<T1, C, 30, 31>
<T3 commit>
<T4 start>
<T4, D, 40, 41>
--- crash ---
```

A. 3
B. 2
C. 4
D. 1

1095. ARIES takes fuzzy checkpoints. Why can't its redo phase simply start at the last checkpoint record? Analysis

A. The checkpoint record is written before the log is forced, so it may be lost
B. Redo must always start from the very first record in the log
C. Losers' updates before the checkpoint must be redone in reverse order
D. A fuzzy checkpoint does not flush dirty pages, so updates logged before it may still be missing from disk

1096. The log disk can do 200 block writes per second, and each commit forces one log block. With group commit, each forced block carries the commit records of 10 transactions. What is the most commits per second, and what is the cost? Analysis

A. 200 commits per second, because each commit still needs its own disk write
B. 2,000 commits per second, with no effect on any transaction's commit time
C. 2,000 commits per second, but each transaction may wait a little longer for its commit
D. 20 commits per second, because each block now holds 10 records

1097. A bank wants no committed transaction to be lost even if the primary site is destroyed, but it does not want processing to stop whenever the backup site is down. Which commit mode fits? Analysis

A. One-safe
B. Two-safe
C. Two-very-safe
D. Group commit at the primary only

1098. Shadow paging needs no log in a single-user system. Why do multi-user systems that use it still keep a log and take checkpoints? Analysis

A. Concurrent transactions change pages under one page table, so a single table switch cannot commit one transaction without the others
B. Shadow paging loses the shadow page table in every crash
C. The page table cannot be stored on disk when there are several users
D. Shadow pages can be read only by the transaction that created them

1099. A small company keeps its log files on the same disk as its data files and takes a dump every Sunday. The disk fails on Thursday. What can be recovered? Analysis

A. Everything up to the failure, by redoing the log after restoring the dump
B. Only the state as of Sunday's dump, because the log describing Monday to Thursday was lost with the disk
C. Nothing, because a dump is useless without the log
D. Everything up to the failure, by undoing the transactions active on Thursday

Indexing & File Organization MCQs for NSCT

1100. In Elmasri-Navathe's terms, what is a primary index? Understanding

A. An index on the ordering key field of a file that is physically sorted on that key
B. Any index built on the primary key, whether or not the file is sorted on it
C. An index on a non-ordering field with one entry per record
D. An index on an ordering field that is not a key

1101. What is the difference between a dense and a sparse index? Understanding

A. A dense index is stored in RAM; a sparse index is stored on disk
B. A dense index uses hashing; a sparse index uses a B+ tree
C. A dense index is always clustered; a sparse index is always non-clustered
D. A dense index has an entry for every search-key value (or record); a sparse index has entries for only some, such as one per block

1102. In a B+ tree index, where are the pointers to data records stored? Understanding

A. In every node, internal and leaf
B. Only in the leaf nodes, which are also linked to each other in key order
C. Only in the root node
D. Only in the internal nodes

1103. For which kind of query is a hash index a poor choice? Understanding

A. A range query such as salary BETWEEN 40000 AND 60000
B. An equality query such as id = 1042
C. Looking up one customer by email address
D. A join on equal key values

1104. A file is physically ordered on dept_no, which is not unique. In Elmasri-Navathe's terms, what is an index on dept_no called? Understanding

A. A primary index
B. A secondary index
C. A hash index
D. A clustering index

1105. How many primary or clustering indexes can a single data file have? Understanding

A. Exactly two
B. One per key attribute
C. Any number
D. At most one

1106. Why is a secondary index on a key field always dense? Understanding

A. Secondary indexes are always stored as hash tables
B. Block anchors make it dense automatically
C. The file is not ordered on that field, so every record needs its own index entry
D. A secondary index stores the whole record in each entry

1107. Why does a B+ tree usually have a higher fan-out than a B-tree for the same block size? Understanding

A. Its internal nodes hold only keys and tree pointers, not data pointers, so more entries fit in a block
B. Its leaves are linked together
C. It allows nodes to be less than half full
D. It stores keys in sorted order

1108. In extendible hashing, when does the directory double in size? Understanding

A. Every time any bucket overflows
B. When a full bucket must split and its local depth equals the global depth
C. When the file has twice as many records as buckets
D. When a bucket becomes empty

1109. What distinguishes linear hashing from extendible hashing? Understanding

A. Linear hashing never allows overflow buckets
B. Linear hashing doubles a directory on every overflow
C. Linear hashing supports range queries efficiently
D. Linear hashing has no directory; buckets are split one at a time in a fixed order

1110. In SQL Server-style terminology, why can a table have only one clustered index but many non-clustered indexes? Understanding

A. A clustered index is always on the primary key
B. Non-clustered indexes are stored in memory only
C. The clustered index decides the physical order of the rows, and rows can be stored in only one order
D. Each clustered index needs its own copy of the table

1111. What is the main trade-off of a heap (unordered) file? Understanding

A. Inserting is cheap, but searching needs a linear scan
B. Searching is fast, but inserting needs re-sorting
C. It supports binary search but not deletion
D. It needs a hash function for every insert

1112. What is the main weakness of static hashing as a file grows? Understanding

A. The number of buckets is fixed, so overflow chains grow and look-ups slow down
B. It cannot handle equality searches
C. Every insertion needs the whole file rehashed
D. It needs the file to be sorted

1113. In a multilevel index, what is the fan-out (fo)? Understanding

A. The blocking factor of the index, bfri: the number of index entries per block
B. The number of data records per data block
C. The number of levels in the index
D. The number of data blocks divided by the number of records

1114. Why is the cost of a B+ tree look-up predictable? Understanding

A. The root always holds the searched key
B. Leaves are linked in key order
C. Every leaf is at the same depth, so every search reads one node per level
D. Nodes are always completely full

1115. In a primary index, what is the block anchor? Understanding

A. The last index block in the file
B. The first record in each data block, whose key is stored in the index entry for that block
C. The pointer from one leaf node to the next
D. The record with the largest key in the file

1116. A file has r = 30,000 records of R = 100 bytes, stored unspanned in blocks of B = 1,024 bytes. What are the blocking factor and the number of blocks? Application

A. bfr = 10.24, b = 2,930
B. bfr = 11, b = 2,728
C. bfr = 10, b = 3,000
D. bfr = 100, b = 300

1117. A static hash file uses h(K) = K mod 7, with buckets numbered 0 to 6. In which bucket is the record with key 50 placed? Application

A. 1
B. 7
C. 0
D. 6

1118. A file has r = 30,000 records of R = 100 bytes, stored unspanned in blocks of B = 1,024 bytes, in no particular order (heap file). Using Elmasri-Navathe's estimate, how many blocks does a linear search read on average to find one record by a key field? Application

A. 3,000
B. 1,500
C. 12
D. 30,000

1119. A file has r = 30,000 records of R = 100 bytes, stored unspanned in blocks of B = 1,024 bytes, sorted on the key field. About how many block accesses does a binary search on that key need? Application

A. 1,500
B. 15
C. 11
D. 12

1120. A file has r = 30,000 records of R = 100 bytes, stored unspanned in blocks of B = 1,024 bytes, sorted on a 9-byte key. A primary index uses 6-byte block pointers. Using Elmasri-Navathe's formulas, how many block accesses does a search through the index need? Application

A. 6
B. 12
C. 45
D. 7

1121. A file has r = 30,000 records of R = 100 bytes, stored unspanned in blocks of B = 1,024 bytes. A secondary index is built on a non-ordering 9-byte key field, with 6-byte block pointers. Using Elmasri-Navathe's formulas, how many index blocks are there, and how many block accesses does a search need? Application

A. 45 index blocks; 7 accesses
B. 442 index blocks; 10 accesses
C. 442 index blocks; 9 accesses
D. 3,000 index blocks; 13 accesses

1122. A dense first-level index occupies 442 blocks, with a fan-out of fo = 68 entries per block. If it is turned into a multilevel index, how many levels are there, and how many block accesses does a search need? Application

A. 2 levels; 3 accesses
B. 3 levels; 4 accesses
C. 3 levels; 10 accesses
D. 4 levels; 5 accesses

1123. B+ tree (Elmasri-Navathe): block size 512 bytes, key 10 bytes, tree (block) pointer 6 bytes. Using $p \times P + (p - 1) \times V \leq B$, what is the order p of an internal node? Application

A. 31
B. 32
C. 33
D. 29

1124. For the same B+ tree (block 512 bytes, key 10 bytes, block pointer 6 bytes, record pointer 7 bytes), what is the leaf order p_leaf, using $p_leaf \times (V + Pr) + P \leq B$? Application

A. 32
B. 30
C. 23
D. 29

1125. A B+ tree has internal order p = 32 and leaf order p_leaf = 29. If it has a root, one internal level and a leaf level, and every node is completely full, how many record pointers can it hold? Application

A. 29,696
B. 32,768
C. 28,768
D. 927

1126. Keys 12, 25, 33, 40, 18 and 47 are inserted into a static hash file with h(K) = K mod 7. Which bucket receives the most keys, and how many? Application

A. Bucket 4, with 2 keys
B. Bucket 5, with 3 keys
C. Bucket 6, with 4 keys
D. Bucket 5, with 4 keys

1127. In Silberschatz's B+ tree with n = 5 (at most 5 pointers per node), what is the minimum number of children of a non-root internal node, and the minimum number of keys in a non-root leaf? Application

A. 2 children; 2 keys
B. 3 children; 3 keys
C. 5 children; 4 keys
D. 3 children; 2 keys

1128. Silberschatz's B+ tree with n = 4 (at most 3 keys per leaf; a full leaf splits by keeping the first ceil(n/2) = 2 keys). Insert 10, 20, 30, 40, 50, 60, 70, 80 into an empty tree in that order. What does the final tree look like? Application

A. Root [40] over two internal nodes
B. Root [30, 60] over three leaves
C. Root [30, 50, 70] over four leaves of two keys each
D. Root [50] over leaves [10 20 30 40] and [50 60 70 80]

1129. An extendible hash file uses the high-order bits of the hash value and has global depth 2. A key's hash value begins 1011. Which directory entry is used? Application

A. 11
B. 01
C. 10
D. 1011

1130. A file has r = 30,000 records of R = 100 bytes, stored unspanned in blocks of B = 1,024 bytes (3,000 blocks). A query matches 20% of the records (6,000) through a non-clustered secondary index, and each matching record may lie in a different block. Which access path is cheaper? Analysis

A. The secondary index, about 10 block reads
B. The secondary index, about 1,500 block reads
C. A full file scan, about 3,000 block reads
D. Binary search, about 12 block reads

1131. Extendible hashing on high-order bits. Buckets hold 2 records; the file starts with global depth 0 and one bucket. Records with hash values 0010, 1010, 0110, 1100 and 0001 are inserted in that order. What are the final global depth and number of buckets? Analysis

A. Global depth 2; 4 buckets
B. Global depth 1; 2 buckets
C. Global depth 2; 3 buckets
D. Global depth 3; 4 buckets

1132. Using Silberschatz's bound, a B+ tree look-up follows a path of at most ceil(log_ceil(n/2)(N)) nodes. With n = 60 and N = 500,000 search keys, what is that bound? Analysis

A. 3
B. 5
C. 4
D. 19

1133. A table of orders is queried with order_date BETWEEN '2026-01-01' AND '2026-01-31', which matches several thousand rows. Why does a clustered index on order_date beat a non-clustered one? Analysis

A. A clustered index has fewer levels than any non-clustered index
B. Matching rows are stored together in consecutive blocks, so far fewer blocks are read
C. A non-clustered index cannot answer range queries
D. A clustered index stores only the dates, not the rows

1134. A Silberschatz B+ tree with n = 4 indexes 1,000 distinct keys. Leaves hold between ceil((n−1)/2) = 2 and n − 1 = 3 keys (assume the tree is large enough that no leaf is the root). Between what limits must the number of leaves lie? Analysis

A. 250 and 500
B. 334 and 500
C. 334 and 1,000
D. 500 and 1,000

1135. A file has r = 30,000 records of R = 100 bytes, stored unspanned in blocks of B = 1,024 bytes, sorted on its 9-byte key. Index entries are 15 bytes (bfri = 68). Compare a sparse primary index with a dense index on the same key. How many index blocks does each need? Analysis

A. Sparse: 45 blocks; dense: 442 blocks
B. Sparse: 442 blocks; dense: 45 blocks
C. Both need 442 blocks
D. Sparse: 3,000 blocks; dense: 30,000 blocks

1136. Keys 16, 24, 40, 64, 72 and 88 (all multiples of 8) are hashed with h(K) = K mod M. Compare M = 8 with M = 7. What is the largest number of keys in any one bucket in each case? Analysis

A. M = 8: 1 key; M = 7: 1 key
B. M = 8: 6 keys; M = 7: 2 keys
C. M = 8: 2 keys; M = 7: 6 keys
D. M = 8: 3 keys; M = 7: 3 keys

1137. Silberschatz's B+ tree with n = 4 (a full leaf keeps its first 2 keys when it splits; an overfull internal node keeps its first ceil((n+1)/2) = 3 pointers). Insert keys 1 to 10 in increasing order into an empty tree. How many levels and how many nodes does the final tree have? Analysis

A. 3 levels, 8 nodes
B. 2 levels, 6 nodes
C. 3 levels, 7 nodes
D. 4 levels, 9 nodes

1138. A file of employees is physically sorted on dept_no (not unique). Most queries are WHERE dept_no = :d, returning about 50 employees each. Which index fits best? Analysis

A. A dense secondary index on dept_no
B. A hash index on emp_name
C. No index; a linear scan is always cheaper for 50 rows
D. A clustering index on dept_no, since matching rows are contiguous and one look-up finds the first block

1139. Block size 512 bytes, key 10 bytes, block pointer 6 bytes, record pointer 7 bytes. A B-tree node needs $p \times P + (p - 1) \times (Pr + V) \leq B$. What is the B-tree order p, compared with the B+ tree internal order of 32? Analysis

A. 32, the same as the B+ tree
B. 29, the B+ tree's leaf order
C. 23, lower than the B+ tree's 32
D. 38, higher than the B+ tree's 32

1140. What is the difference between spanned and unspanned record organisation? Understanding

A. Spanned stores records in sorted order; unspanned stores them in arrival order
B. Spanned uses a hash function to place records; unspanned uses an index
C. Spanned stores only fixed-length records; unspanned stores only variable-length ones
D. Spanned lets one record continue from one block into the next; unspanned keeps every record inside a single block

1141. Why is inserting a record into a file kept physically sorted on a key expensive? Understanding

A. The right position must be found and later records shifted to make room, which can rewrite many blocks
B. Every insertion must rebuild the hash function
C. A sorted file cannot store records with duplicate non-key values
D. The whole file must be read to compute a new blocking factor

1142. Records are almost always fetched by an exact value of one field, such as account_no = 'PK-4471'. Which primary file organisation usually needs the fewest block accesses for this? Understanding

A. A heap (unordered) file
B. A file sorted on account_no, searched with binary search
C. A hash file on account_no
D. A file sorted on a different field

1143. For which kind of column is a bitmap index best suited? Understanding

A. A column whose values are all unique, such as a national ID number
B. A column updated many times a second by many users
C. A column with few distinct values, such as gender or province
D. A long free-text comments column

1144. A composite index is built on (dept, salary). In what order are its entries kept? Understanding

A. Sorted by salary, and by dept within each salary
B. Sorted by dept only; salary values are in insertion order
C. Sorted by dept, and by salary within each dept
D. In two separate sorted lists, one for each column

1145. What is a covering index for a query? Understanding

A. An index that covers every row of the table, which makes it dense
B. An index that contains every column the query needs, so the query is answered without reading the table rows
C. An index built on every column of a table
D. An index that is stored on the same disk blocks as the table

1146. What is the main weakness of an ISAM (indexed sequential access method) index compared with a B+ tree? Understanding

A. Its index levels are fixed when the file is built, so new records go to overflow chains that slow searches over time
B. It cannot support equality searches on the key
C. It must be completely rebuilt after every single insertion
D. It does not keep the data file in key order

1147. In a hash file, what is a collision? Understanding

A. Two records have the same key value
B. A bucket is read by two transactions at once
C. Two different keys hash to the same address
D. The hash function returns an address outside the file

1148. How does chaining resolve collisions in a hash file? Understanding

A. The record is placed in the next free slot after its home address
B. A second hash function is applied to find another address
C. The whole file is rehashed with a larger number of buckets
D. Overflow records are kept in an overflow area and linked to their home bucket by pointers

1149. Which of these makes a file's records variable-length? Understanding

A. A fixed-size integer key in every record
B. Storing the records in sorted order
C. Keeping the records in a hash file
D. A field such as a name whose length differs from record to record

1150. A table gets several new indexes. What is the usual cost of this? Understanding

A. SELECT queries on the indexed columns get slower
B. The table can no longer have a primary key
C. The table rows must be stored in a hash file
D. INSERT, UPDATE and DELETE get slower, because every index must be kept in step with the table

1151. In Silberschatz's terms, what is a search key of an index? Understanding

A. The attribute or set of attributes used to look up records through the index
B. The primary key of the table, which every index must use
C. The physical address of a record on disk
D. The hash value stored in a bucket header

1152. What does a record pointer hold that a block pointer does not? Understanding

A. The position of the record inside the block, as well as the block's address
B. A copy of the record's key value
C. The record's length in bytes
D. The address of the next block in the file

1153. Elmasri-Navathe build a secondary index on a non-key field with one entry per distinct value. How does each entry reach the many records that share that value? Understanding

A. It points to the first matching record, and the rest follow it physically
B. It points to a block (or chain of blocks) of record pointers, one for each matching record
C. It stores the matching records themselves inside the index
D. It points to the hash bucket where those records are stored

1154. After a deletion, a B+ tree leaf has fewer keys than the minimum. What does the tree do? Understanding

A. It leaves the leaf under-full, because B+ trees allow any occupancy
B. It always merges with a sibling, even if the merged node overflows
C. It borrows a key from a sibling if the sibling can spare one; otherwise it merges with the sibling and removes a separator from the parent
D. It rebuilds the whole tree from the remaining keys

1155. Why is bulk loading preferred to inserting entries one by one when building a B+ tree on a large existing table? Understanding

A. Bulk loading allows the tree to have leaves at different depths
B. The entries are sorted and the tree is built bottom-up, which avoids repeated splits and random I/O
C. Bulk loading stores record pointers in internal nodes as well
D. Bulk loading builds a hash index instead of a tree

1156. How does open addressing with linear probing handle a collision? Understanding

A. It checks the following positions one by one until it finds an empty slot
B. It links the new record to the full bucket with an overflow pointer
C. It applies a second hash function to the key
D. It doubles the number of buckets

1157. In extendible hashing, a bucket has local depth d' and the directory has global depth d. How many directory entries point to that bucket? Understanding

A. 2^(d − d')
B. d − d'
C. 2^d'
D. Exactly one

1158. In linear hashing, a record overflows bucket 5 while the split pointer n is 2. Which bucket is split? Understanding

A. Bucket 5, the one that overflowed
B. Bucket 2, the one the split pointer names
C. Every bucket from 2 to 5
D. No bucket; the directory doubles instead

1159. To make insertions easier with a clustering index, what do Elmasri-Navathe suggest for the data file? Understanding

A. Store each new record at the end of the file and rebuild the index each night
B. Hash the clustering field to choose a block
C. Keep one index entry per record instead of one per value
D. Reserve a whole block, or a cluster of contiguous blocks, for each value of the clustering field

1160. Silberschatz describes building a B+ tree on a non-unique search key. What is the usual way to make every key in the tree unique? Understanding

A. Keep only the first record for each key value
B. Store duplicates in a hash table outside the tree
C. Add the record identifier (or primary key) to the search key as an extra attribute
D. Use a B-tree instead, since B-trees cannot hold duplicates

1161. A table has bitmap indexes on gender and province. How is WHERE gender = 'F' AND province = 'Punjab' answered? Understanding

A. The bitmap for 'F' is ORed with the bitmap for 'Punjab'
B. Only one bitmap can be used, and the other condition is checked row by row
C. The two bitmaps are added together as integers
D. The bitmap for 'F' is ANDed with the bitmap for 'Punjab', and the 1 bits give the matching rows

1162. A hash file performs badly even though its total number of records is well within capacity. What is the most likely cause? Understanding

A. The file has too few records for hashing to work
B. Skew: many keys hash to a few buckets, so those buckets overflow while others stay nearly empty
C. The records are of fixed length
D. The hash function returns addresses in the range 0 to M − 1

1163. Why can a sparse index not be built on a field the file is not sorted on? Understanding

A. A sparse entry stands for a whole block only because the records in that block fall between consecutive index keys
B. Sparse indexes need the field to be numeric
C. A sparse index must use hashing to find blocks
D. The DBMS allows only one sparse index per database

1164. A file has r = 10,000 records of R = 300 bytes, stored unspanned in blocks of B = 4,096 bytes. What are the blocking factor and the number of blocks? Application

A. bfr = 14, b = 715
B. bfr = 13, b = 733
C. bfr = 13, b = 770
D. bfr = 13.65, b = 733

1165. Records of R = 300 bytes are stored unspanned in blocks of B = 4,096 bytes. How many bytes are unused in each full block? Application

A. 96
B. 0
C. 300
D. 196

1166. A static hash file uses h(K) = K mod 11, with buckets 0 to 10. In which bucket is the record with key 123 placed? Application

A. 3
B. 2
C. 1
D. 11

1167. An extendible hash file has global depth 3. One bucket has local depth 1. How many directory entries point to it? Application

A. 2
B. 1
C. 4
D. 8

1168. A table has 1,000,000 rows. A bitmap index is built on a column with 4 distinct values, using one bit per row per value. How large are the bitmaps in total, ignoring headers? Application

A. 125,000 bytes
B. 4,000,000 bytes
C. 1,000,000 bytes
D. 500,000 bytes

1169. The file has r = 10,000 records of R = 300 bytes and blocks of B = 4,096 bytes. If spanned organisation is used, about how many blocks does it need? Application

A. 770
B. 733
C. 732
D. 714

1170. A file has r = 50,000 records of R = 200 bytes in blocks of B = 2,048 bytes, unspanned and sorted on a 12-byte key. A primary index uses 8-byte block pointers. Using Elmasri-Navathe's formulas, how many block accesses does a search through the primary index need? Application

A. 13
B. 7
C. 6
D. 8

1171. A file is ordered on dept_no, which has 500 distinct values. A clustering index has 10-byte entries (4-byte dept_no + 6-byte block pointer) in 1,024-byte blocks. How many index blocks are there, and how many index-block accesses does a binary search of the index need? Application

A. 500 index blocks; 9 accesses
B. 5 index blocks; 5 accesses
C. 50 index blocks; 6 accesses
D. 5 index blocks; 3 accesses

1172. r = 50,000 records, B = 2,048 bytes. A dense secondary index on a 12-byte key uses 8-byte pointers (bfri = 102). It is turned into a multilevel index. How many levels does it have, and how many block accesses does a search need? Application

A. 2 levels; 3 accesses
B. 3 levels; 4 accesses
C. 3 levels; 3 accesses
D. 9 levels; 10 accesses

1173. B+ tree (Elmasri-Navathe): block 1,024 bytes, key V = 12 bytes, block pointer P = 8 bytes, record pointer Pr = 10 bytes. Using $p \times P + (p - 1) \times V \le B$ and $p_{leaf} \times (V + Pr) + P \le B$, what are p and p_{leaf}? Application

A. p = 52, p_{leaf} = 46
B. p = 51, p_{leaf} = 50
C. p = 46, p_{leaf} = 51
D. p = 51, p_{leaf} = 46

1174. Silberschatz's B+ tree with n = 4 (at most 3 keys per leaf; a full leaf splits by keeping its first 2 keys and copying the first key of the new leaf up). Insert 15, 5, 25, 10, 20, 30, 35 into an empty tree. What is the final tree? Application

A. Root [15, 25] over leaves [5 10], [15 20], [25 30 35]
B. Root [15] over leaves [5 10], [15 20 25 30 35]
C. Root [10, 20, 30] over leaves [5], [10 15], [20 25], [30 35]
D. Root [20] over leaves [5 10 15], [20 25 30 35]

1175. A linear hash file started with M = 4 buckets and uses h0(K) = K mod 4 and h1(K) = K mod 8. Bucket 0 has been split, so the split pointer is n = 1. Which bucket holds the record with key 20? Application

A. 4
B. 0
C. 1
D. 5

1176. A hash table has 10 slots (0 to 9), h(K) = K mod 10, and linear probing. Keys 22, 32, 42, 15 and 25 are inserted in that order. In which slot is 25 stored? Application

A. 5
B. 7
C. 6
D. 2

1177. A B+ tree has n = 100 (each node up to 100 pointers; each leaf up to 99 keys). If every node is completely full, what is the fewest levels that can index 1,000,000 keys? Application

A. 3
B. 5
C. 2
D. 4

1178. Given this index, which WHERE clause cannot use it for an index search (only a full scan of the table or index)? Application

```
CREATE INDEX ix_dept_sal ON emp (dept, salary);
```

A. WHERE dept = 'IT'
B. WHERE dept = 'IT' AND salary > 50000
C. WHERE dept = 'IT' AND city = 'Lahore'
D. WHERE salary > 50000

1179. Silberschatz's B+ tree with n = 5 (at most 4 keys per leaf; a full leaf keeps its first 3 keys when it splits). Insert 8, 3, 12, 6, 1, 15, 10, 4 into an empty tree. What is the final tree? Application

A. Root [6] over leaves [1 3 4], [6 8 10 12 15]
B. Root [8] over leaves [1 3 4 6], [8 10 12 15]
C. Root [6, 10] over leaves [1 3 4], [6 8], [10 12 15]
D. A single leaf [1 3 4 6 8 10 12 15]

1180. A B+ tree has internal order p = 51 and leaf order p_leaf = 46. It has a root, one internal level and a leaf level, and every node is completely full. How many record pointers can it hold? Application

A. 2,346
B. 119,646
C. 132,651
D. 97,336

1181. Nodes in a B+ tree are on average only partly full. Assume each internal node has 35 pointers and each leaf 32 record pointers. With a root, one internal level and a leaf level, about how many record pointers does the tree hold? Application

A. 39,200
B. 35,840
C. 42,875
D. 1,120

1182. Extendible hashing on high-order bits, buckets of 2 records, starting from global depth 0 and one bucket. Hash values 1100, 0101, 1110, 1001 and 1111 are inserted in that order. What is the local depth of the bucket holding 1001, and how many directory entries point to it? Application

A. Local depth 3; 1 entry
B. Local depth 1; 4 entries
C. Local depth 2; 1 entry
D. Local depth 2; 2 entries

1183. Linear hashing (Elmasri-Navathe): M = 2 initial buckets, 2 records per bucket, h_j(K) = K mod (2^j × M). Any overflow splits the bucket named by the split pointer n, which starts at 0. Keys 4, 8, 5, 12, 7, 9, 15 are inserted in order. How many buckets are there at the end, and where is 15? Application

A. 3 buckets; 15 is in bucket 1
B. 4 buckets; 15 is in bucket 3
C. 4 buckets; 15 is in bucket 1
D. 2 buckets; 15 is in an overflow chain of bucket 1

1184. Silberschatz's B+ tree with n = 4 (at most 3 keys per leaf; a full leaf keeps its first 2 keys when it splits). Keys 1 to 20 are inserted one at a time in increasing order. How many leaves does the final tree have? Application

A. 10
B. 7
C. 20
D. 9

1185. An audit_log table receives 5,000 inserts a second and is queried only once a day, by date. It carries 6 secondary indexes added over the years. Inserts have become slow. What is the best first step? Analysis

A. Add a covering index for every column
B. Convert the table to a sorted file on the primary key
C. Drop the indexes that no query uses and keep one on the date column
D. Increase the number of indexes so that each insert touches fewer blocks

1186. In a data warehouse, a B+ tree index on gender (2 values) of a 10-million-row table is never used by the optimiser. Queries often combine gender with region and age_band. What fits better? Analysis

A. Bitmap indexes on gender, region and age_band, combined with bitwise AND
B. A hash index on gender
C. A unique index on gender
D. A clustered index on gender only

1187. Both queries use the index to find the rows with dept = 'IT'. Why does Query 1 need fewer block accesses? Analysis

```
CREATE INDEX ix_dept_sal ON emp (dept, salary);

-- Query 1
SELECT dept, salary FROM emp WHERE dept = 'IT';
-- Query 2
SELECT name, salary FROM emp WHERE dept = 'IT';
```

A. Query 1 is covered by the index, so it never reads the table; Query 2 must fetch each row to get name
B. Query 1 returns fewer rows than Query 2
C. Query 2 cannot use the index at all, so it scans the table
D. Query 1 uses a hash look-up, while Query 2 uses the B+ tree

1188. A sensor table receives a constant stream of inserts and is read only by a nightly job that processes every row. Which primary file organisation fits best? Analysis

A. A file sorted on sensor_id, so the nightly job can use binary search
B. A hash file on sensor_id
C. A heap file, because inserts just go at the end and the nightly job reads everything anyway
D. A file sorted on the reading's timestamp, with a dense index on every column

1189. An orders table is queried mostly with WHERE customer_id = ? (each customer has about 200 orders spread over years) and rarely by date. Which column should get the table's single clustered index? Analysis

A. customer_id, so each customer's orders are stored together and read in a few blocks
B. order_date, because dates are always the best clustering key
C. The order_id primary key, since only a primary key can be clustered
D. No column, because clustering helps only range queries

1190. A hash file stores one record per day for 2026, keyed by the date as an integer YYYYMMDD (e.g. 20260315), with h(K) = K mod 100 and 100 buckets. What goes wrong? Analysis

A. Every record lands in bucket 26, the year's last two digits
B. The keys spread evenly, about 3 or 4 per bucket
C. Only 12 buckets are used, one for each month
D. K mod 100 is just the day of the month, so only 31 buckets are ever used and some get 12 records

1191. An extendible hash file has buckets of 4 records. A batch inserts 6 records whose keys all have exactly the same hash value. What happens? Analysis

A. The bucket splits once and the records divide 3 and 3
B. Splits cannot separate them, so the directory keeps doubling without help; an overflow bucket is needed
C. Linear probing moves the extra records to the next bucket
D. The global depth stays the same because only local depth changes

1192. A file has 5,000 blocks. A non-clustered B+ tree on city has 3 levels, and the entries for city = 'Quetta' fit in 2 leaves. 300 rows match, each in a different block. Roughly how do the two access paths compare? Analysis

A. The index costs about 3 + 1 + 300 = 304 block reads, far below the 5,000 of a full scan
B. The full scan is cheaper, because a non-clustered index always reads every block
C. Both cost about the same, because 300 rows means 300 blocks either way
D. The index costs 3 block reads in total, because it holds the city values

1193. Silberschatz's B+ tree with n = 4 (leaves hold 2 to 3 keys). The tree is root [20] over leaves [5 10 15] and [20 25]. Key 25 is deleted. What is the tree afterwards? Analysis

A. A single leaf [5 10 15 20]
B. Root [20] over leaves [5 10 15] and [20]
C. Root [20] over leaves [5 10] and [15 20]
D. Root [15] over leaves [5 10] and [15 20]

1194. Silberschatz's B+ tree with n = 4 (leaves hold 2 to 3 keys). The tree is root [20] over leaves [5 10] and [20 25]. Key 25 is deleted. What happens? Analysis

A. 15 is borrowed from the left leaf, giving [5 10] and [15 20]
B. The two leaves merge into [5 10 20], the root is left with one child and is removed, so the tree loses a level
C. The right leaf stays as [20] and the root is unchanged
D. The root key 20 moves down into the right leaf, giving [20 20]

1195. A file indexed with ISAM grows from 100,000 to 400,000 records, with most new keys in the middle of the key range. The same file could instead have a B+ tree. How do searches compare after the growth? Analysis

A. ISAM searches stay the same because its index is static
B. Both slow down equally, because both are multilevel indexes
C. ISAM searches slow down as they follow overflow chains; B+ tree searches still read one node per level, with perhaps one extra level
D. The B+ tree slows down more, because it must split nodes during searches

1196. Block 512 bytes, key 10 bytes, block pointer 6 bytes, record pointer 7 bytes, giving B+ tree orders p = 32, p_leaf = 29 and B-tree order p = 23. With every node full, how many levels does each need to index 500,000 records? Analysis

A. B+ tree: 5 levels; B-tree: 4 levels
B. Both need 4 levels
C. B+ tree: 4 levels; B-tree: 5 levels
D. B+ tree: 3 levels; B-tree: 5 levels

1197. A linear hash file has buckets of 2 records and M = 2. Keys 4, 8 and 12 all hash to bucket 0, so 12 causes an overflow and bucket 0 is split with h1(K) = K mod 4. Why does bucket 0 still hold an overflow record afterwards? Analysis

A. Linear hashing never moves records during a split
B. The new bucket 2 is created only when the directory doubles
C. All three keys give 0 under K mod 4, so the split moves none of them to the new bucket 2
D. The split pointer moved to bucket 1 before the records were rehashed

1198. A file is sorted on emp_id. A frequent query only asks whether a given emp_id exists. Compare a sparse primary index with a dense index on emp_id. Analysis

A. The sparse index answers from the index alone, because it is smaller
B. Neither can answer it without a full file scan
C. The dense index answers from the index alone; the sparse index must still read the data block to check
D. Both must read the data block, so they cost the same

1199. A users table gets two queries: 90% are WHERE email = ? and 10% are WHERE age BETWEEN 25 AND 35 ORDER BY age. You may build one index on each column. Which pair fits best? Analysis

A. A B+ tree on email and a hash index on age
B. A hash index on email and a B+ tree on age
C. Hash indexes on both columns
D. Bitmap indexes on both columns

Query Processing & Optimization MCQs for NSCT

1200. In the order a DBMS performs them, which are the three main steps of query processing? Understanding

A. Optimisation, parsing and translation, evaluation
B. Evaluation, optimisation, parsing and translation
C. Compilation, linking, execution
D. Parsing and translation, optimisation, evaluation

1201. What is a query evaluation plan (execution plan)? Understanding

A. The SQL text after the parser has removed comments and spaces
B. A list of the tables a query reads, in alphabetical order
C. The final result table that the query returns to the user
D. A relational algebra expression annotated with the algorithm and access path for each operation

1202. Silberschatz measures the cost of a query evaluation plan mainly by which quantity? Understanding

A. The number of rows in the final answer
B. The number of disk block transfers and disk seeks
C. The number of clauses in the SQL statement
D. The number of CPU registers the query uses

1203. Which rule is a standard heuristic in query optimisation? Understanding

A. Perform Cartesian products before selections
B. Perform selection operations as early as possible
C. Delay projections until after every join
D. Always join the two largest relations first

1204. In query evaluation, what does pipelining mean? Understanding

A. Running each operation to completion and storing its full result on disk
B. Passing tuples from one operation straight to its parent without writing a temporary relation to disk
C. Splitting one query across several CPUs that each hold a full copy of the data
D. Sorting every relation before any operation starts

1205. Which join algorithm can evaluate any join condition, including a non-equality condition such as r.a < s.b? Understanding

A. Hash join
B. Merge (sort-merge) join
C. Nested-loop join
D. Grace hash join

1206. In the catalogue statistics used for cost estimation, what does V(A, r) denote? Understanding

A. The number of tuples in relation r
B. The number of distinct values of attribute A in relation r
C. The number of blocks holding relation r
D. The size in bytes of one tuple of r

1207. What does a cost-based query optimiser try to do? Understanding

A. Find the plan that is provably the fastest on every run
B. Choose the equivalent plan with the lowest estimated cost
C. Rewrite the query so it returns fewer rows
D. Choose the plan with the fewest operators regardless of cost

1208. In materialised evaluation, what happens to the result of each intermediate operation? Understanding

A. It is sent to the user as soon as the first tuple is produced
B. It is discarded and recomputed whenever the parent needs a tuple
C. It is converted into an index on the base relation
D. It is stored as a temporary relation, usually on disk, before the parent operation reads it

1209. Which operation is blocking, meaning it cannot output its first result tuple until it has read all of its input? Understanding

A. Sorting
B. Selection on a single relation
C. Projection without duplicate elimination
D. UNION ALL

1210. Which equivalence rule allows a selection with a conjunctive condition to be split into two separate selections? Understanding

A. $\sigma_{\theta 1 \wedge \theta 2}(E) = \sigma_{\theta 1}(\sigma_{\theta 2}(E))$
B. $\pi_{L1}(\pi_{L2}(E)) = \pi_{L1}(E)$
C. $E1 \bowtie E2 = E2 \bowtie E1$
D. $\sigma_{\theta}(E1 \cup E2) = \sigma_{\theta}(E1) \cup \sigma_{\theta}(E2)$

1211. An operator offers open(), next() and close(), and its parent calls next() whenever it needs another tuple. What kind of pipelining is this? Understanding

A. Producer-driven (eager) pipelining
B. Materialised evaluation
C. Demand-driven (lazy) pipelining
D. Parallel hash partitioning

1212. A System R-style optimiser uses dynamic programming but restricts the join trees it considers. Which trees does it consider? Understanding

A. Bushy join trees only
B. Trees in which no join has a base relation as an input
C. Only the order written in the FROM clause
D. Left-deep join trees, where the right input of every join is a base relation

1213. Why can a range query using a secondary (non-clustering) index cost more than a full linear scan? Understanding

A. A secondary index can only answer equality conditions, never ranges
B. A secondary index always has more levels than the relation has blocks
C. Linear scans never need to read the whole relation
D. Each matching record may sit in a different block, so the query can need one seek and transfer per record

1214. In a hash join, which relation should normally be used as the build input? Understanding

A. The smaller relation, so its partitions are more likely to fit in memory
B. The larger relation, so fewer partitions are created
C. Whichever relation is listed first in the FROM clause
D. The relation that has an index on the join attribute

1215. An optimiser keeps a slightly more expensive sub-plan because its output is sorted on the join attribute of a later join. What is this sorted output called? Understanding

A. A clustering index
B. An interesting sort order
C. A materialised view
D. A covering index

1216. A relation student has 10,000 tuples and V(city, student) = 50. Assuming a uniform distribution, what is the standard estimate for the size of σcity='Multan'(student)? Application

A. 50
B. 200
C. 10,000
D. 500

1217. enrol occupies 300 blocks and course occupies 40 blocks. If course fits entirely in memory, how many block transfers does a nested-loop join of the two need? Application

A. 12,000
B. 12,040
C. 300
D. 340

1218. enrol has 6,000 tuples in 300 blocks; course has 400 tuples in 40 blocks. Using a tuple-at-a-time nested-loop join with enrol as the outer relation and one buffer block per relation, how many block transfers are needed in the worst case? Application

A. 240,300
B. 12,300
C. 120,040
D. 340

1219. r has 120 blocks and s has 500 blocks. With M = 22 memory blocks, a block nested-loop join uses r as the outer relation, giving it M − 2 blocks. How many block transfers are needed? Application

A. 60,120
B. 620
C. 3,120
D. 2,620

1220. enrol (300 blocks) and course (40 blocks) are both already sorted on course_id. With enough memory for each group of matching tuples, how many block transfers does a merge join need? Application

A. 340
B. 1,020
C. 12,000
D. 680

1221. Using the simplified Silberschatz estimate 3(b_r + b_s) for a hash join without recursive partitioning, what is the cost of joining enrol (300 blocks) with course (40 blocks)? Application

A. 1,020 block transfers
B. 340 block transfers
C. 680 block transfers
D. 12,300 block transfers

1222. exam has 5,000 tuples, and marks ranges from min = 0 to max = 100, assumed uniform. What is the estimated size of $\sigma_{marks \geq 80}(exam)$? Application

A. 4,000
B. 80
C. 1,000
D. 50

1223. student has 10,000 tuples. The condition dept = 'CS' has an estimated size of 500 tuples and year = 4 has 2,000. Assuming independence, what is the estimated size of $\sigma_{dept='CS' \wedge year=4}(student)$? Application

A. 100
B. 500
C. 2,400
D. 2,500

1224. student has 10,000 tuples. dept = 'EE' matches an estimated 500 tuples and year = 1 matches 2,000. Assuming independence, what is the estimated size of $\sigma_{dept='EE' \vee year=1}(student)$? Application

A. 2,500
B. 2,400
C. 100
D. 1,250

1225. takes(ID, course_id) has 10,000 tuples, and takes.ID is a non-null foreign key referencing student.ID, the primary key of student (5,000 tuples). How many tuples does student $\bowtie$ takes contain? Application

A. Exactly 5,000
B. 50,000,000
C. Exactly 10,000
D. Exactly 15,000

1226. r(A, B) has 1,000 tuples and s(B, C) has 2,000 tuples. B is a key of neither. V(B, r) = 50 and V(B, s) = 100. What is the standard estimate for the size of r ⋈ s? Application

A. 40,000
B. 2,000,000
C. 20,000
D. 3,000

1227. This script runs in SQLite. What detail does EXPLAIN QUERY PLAN report for the SELECT? Application

```
CREATE TABLE emp(id INTEGER PRIMARY KEY, name TEXT, dept TEXT, salary INT);
CREATE INDEX idx_dept ON emp(dept);
EXPLAIN QUERY PLAN SELECT name FROM emp WHERE dept = 'IT';
```

A. SCAN emp
B. SEARCH emp USING INDEX idx_dept (dept=?)
C. SEARCH emp USING INTEGER PRIMARY KEY (rowid=?)
D. USE TEMP B-TREE FOR ORDER BY

1228. How many different join orders exist for joining 4 relations, counting bushy trees and treating r1 ⋈ r2 and r2 ⋈ r1 as different? Application

A. 24
B. 12
C. 64
D. 120

1229. A 1,000-block relation is sorted by external sort-merge with M = 11 buffer blocks. Using Silberschatz's formula $b_r(2\lceil \log_{M-1}(b_r/M) \rceil + 1)$, which does not count writing the final result, how many block transfers are needed? Application

A. 3,000
B. 7,000
C. 5,000
D. 2,000

1230. Relations R(A, B) and S(A, B) are union-compatible. Which of these rewrites is NOT valid for all instances of R and S? Analysis

A. $\sigma_{A=1}(R \cup S) = \sigma_{A=1}(R) \cup \sigma_{A=1}(S)$
B. $\sigma_{A=1}(R - S) = \sigma_{A=1}(R) - S$
C. $\pi_A(R - S) = \pi_A(R) - \pi_A(S)$
D. $\pi_A(R \cup S) = \pi_A(R) \cup \pi_A(S)$

1231. student has 5,000 tuples with V(dept, student) = 25; takes has 10,000 tuples, and takes.ID is a non-null foreign key to the key student.ID. What is the estimated size of σdept='Physics'(student) ⋈ takes? Analysis

A. 200
B. 10,000
C. 400
D. 2,000,000

1232. A selection produces a 50-block result that feeds a projection. Compared with materialising the selection's result on disk, how many block transfers does pipelining save? Analysis

A. 100
B. 50
C. 0
D. 25

1233. r has 300 blocks and s has 40 blocks, with M = 12 memory blocks. For a block nested-loop join giving the outer relation M − 2 blocks, which outer relation is cheaper, and at what cost? Analysis

A. r as outer: 1,500 block transfers
B. s as outer: 12,040 block transfers
C. s as outer: 1,240 block transfers
D. Either order: both cost 12,300 block transfers

1234. A hash join has a build input of 6,400 blocks. Recursive partitioning is avoided when M > n_h + 1, where n_h = [b_s / M]. Which of these memory sizes is the smallest that avoids recursive partitioning? Analysis

A. 100 blocks
B. 50 blocks
C. 20 blocks
D. 8 blocks

1235. A relation has 1,000 blocks. A query matches 400 records through a secondary B+-tree index of height 3. With tS = 4 ms and tT = 0.1 ms, which access path should the optimiser choose? Analysis

A. Secondary index, about 16 ms versus about 104 ms for the scan
B. Linear scan, about 104 ms versus about 1,652 ms for the index
C. Secondary index, because an index is always faster than a scan
D. Linear scan, because a secondary index cannot answer this query

1236. r (800 blocks) and s (200 blocks) are both already sorted on the join attribute A, and M = 52. Which algorithm has the lowest block-transfer cost for the equi-join r ⋈ s on A? Analysis

A. Hash join, 3,000 transfers
B. Block nested-loop with s as outer, 3,400 transfers
C. Block nested-loop with r as outer, 3,200 transfers
D. Merge join, 1,000 transfers

1237. A student rewrites σA<C(R(A, B) ⋈ S(B, C)) as σA<1(R) ⋈ S to 'push the selection down'. Why is this rewrite wrong? Analysis

A. The condition compares attributes of both R and S, so it cannot be pushed to R alone
B. Selections may never be moved below a join
C. Natural join is not commutative, so R and S must swap places
D. A selection must always be applied after all projections

1238. Table t has a B+-tree index on A but no index on B. For σA=5 ∨ B=7(t), why can the optimiser not answer the query using the index on A alone? Analysis

A. B+-tree indices cannot answer equality conditions
B. OR conditions are always evaluated by sorting the relation first
C. The index on A would return tuples that fail A = 5
D. Tuples with B = 7 but A ≠ 5 also qualify, and finding them needs a scan or an index on B

1239. r1(A, B) has 1,000 tuples, r2(B, C) 2,000 and r3(C, D) 500. V(B, r1) = 100, V(B, r2) = 500, V(C, r2) = 50, V(C, r3) = 500. To keep the first intermediate result smallest for r1 ⋈ r2 ⋈ r3, which join should run first? Analysis

A. r1 ⋈ r2, estimated 4,000 tuples
B. r1 ⋈ r3, estimated 1,500 tuples
C. r1 ⋈ r2, estimated 20,000 tuples
D. r2 ⋈ r3, estimated 2,000 tuples

1240. During query processing, what does the parser do with an SQL query? Understanding

A. Estimates the cost of each possible plan and keeps the cheapest one
B. Reads the data blocks from disk and returns the result tuples
C. Checks its syntax, verifies that the named relations and attributes exist, and translates it into an internal form such as relational algebra
D. Builds indexes on the attributes used in the WHERE clause

1241. Where does a cost-based optimiser get values such as n_r, b_r and V(A, r) for its estimates? Understanding

A. By running the query once and counting the tuples it returns
B. From the SQL text of the query itself
C. From the statistics stored in the system catalogue (data dictionary)
D. From the transaction log written by the recovery manager

1242. In Silberschatz's catalogue statistics, what does the blocking factor f_r of a relation r mean? Understanding

A. The number of blocks that r occupies on disk
B. The number of tuples of r that fit in one disk block
C. The fraction of r's tuples that satisfy a selection
D. The number of buffer blocks reserved for r in memory

1243. Which selection algorithm can be applied to any file, whatever its ordering and whether or not it has indexes? Understanding

A. Linear search (a full file scan)
B. Binary search on the selection attribute
C. Primary B+-tree index lookup
D. Hash index lookup

1244. Binary search can be used to evaluate σA=v(r) only under which condition? Understanding

A. The file is physically ordered on attribute A
B. A secondary hash index exists on attribute A
C. The relation fits entirely in main memory
D. Attribute A has no duplicate values and no index

1245. How does a DBMS usually implement a projection that must remove duplicates, as in SELECT DISTINCT dept FROM emp? Understanding

A. By sorting or hashing the projected tuples so that duplicates become adjacent or fall in the same bucket
B. By building a B+-tree index on every attribute of emp
C. By running a nested-loop join of emp with itself
D. By pipelining the tuples without storing any of them

1246. Which property of the natural join lets an optimiser evaluate (r1 ⋈ r2) ⋈ r3 as r1 ⋈ (r2 ⋈ r3)? Understanding

A. Commutativity
B. Idempotence
C. Associativity
D. Distributivity over union

1247. Which equivalence rule turns a selection over a Cartesian product into a theta join? Understanding

A. σθ(E1 ∪ E2) = σθ(E1) ∪ σθ(E2)
B. πL(E1 × E2) = πL(E1) × πL(E2)
C. σθ(E1 × E2) = E1 ⋈θ E2
D. E1 × E2 = E2 × E1

1248. In producer-driven (eager) pipelining, how do tuples move between operators? Understanding

A. The parent calls next() on its child each time it needs another tuple
B. Each operator generates tuples as fast as it can and places them in a buffer for its parent, without waiting to be asked
C. Each operator writes its whole result to disk before the parent starts
D. The operators run one at a time, in alphabetical order of their names

1249. What is the selectivity of a selection condition? Understanding

A. The number of distinct values of the attribute used in the condition
B. The fraction of a relation's tuples that satisfy the condition
C. The number of disk blocks read to evaluate the condition
D. The number of indexes available on the condition's attributes

1250. What must exist for an index nested-loop join to be used? Understanding

A. An index on every attribute of the outer relation
B. An index on the join attribute of the inner relation
C. Both relations sorted on the join attribute
D. Enough memory to hold the outer relation completely

1251. In the first stage of external sort-merge with M memory blocks, how large is each sorted run written to disk? Understanding

A. Exactly one block
B. M − 1 blocks, because one block is kept for output
C. The whole relation, because it is sorted in one run
D. Up to M blocks, because each run is built by sorting one memory-full of the relation

1252. Which equivalence rule lets a sequence of projections be replaced by the last (outermost) one? Understanding

A. $\pi L(E1 - E2) = \pi L(E1) - \pi L(E2)$
B. $\sigma\theta1(\sigma\theta2(E)) = \sigma\theta1 \vee \theta2(E)$
C. $\pi L1(\pi L2(\ldots(\pi Ln(E))\ldots)) = \pi L1(E)$, provided $L1 \subseteq L2 \subseteq \ldots \subseteq Ln$
D. $\pi L(\sigma\theta(E)) = \sigma\theta(E)$ for any list L

1253. In PostgreSQL, how does EXPLAIN ANALYZE differ from plain EXPLAIN? Understanding

A. EXPLAIN ANALYZE refreshes the table statistics but does not show a plan
B. EXPLAIN ANALYZE shows the plan in relational algebra notation
C. EXPLAIN ANALYZE shows only the estimated cost, without the plan tree
D. EXPLAIN ANALYZE actually runs the query and reports real row counts and times beside the estimates

1254. Why do many DBMSs keep histograms on attributes instead of only V(A, r), min and max? Understanding

A. Histograms let the DBMS answer queries without reading the table
B. Histograms replace indexes, so no B+-trees need to be built
C. Histograms are needed to check primary-key constraints
D. Real data is often skewed, and a histogram lets the optimiser estimate selectivity without assuming a uniform distribution

1255. Why do optimisers often rewrite a correlated subquery as a join (decorrelation)? Understanding

A. Joins always return fewer rows than subqueries
B. Evaluated naively, a correlated subquery runs once per outer tuple, while a join can use set-at-a-time algorithms such as hash or merge join
C. SQL does not allow a correlated subquery in the WHERE clause
D. A join can be evaluated without reading the inner relation

1256. In the partitioning phase of a hash join of r and s, why must the same hash function h be applied to the join attribute of both relations? Understanding

A. So that both relations end up sorted on the join attribute
B. So that every partition has exactly the same number of tuples
C. So that the build relation fits in one disk block
D. So that tuples with equal join values land in partitions with the same number, and only matching partition pairs need to be joined

1257. What is semantic query optimisation? Understanding

A. Choosing join algorithms only from their names
B. Using integrity constraints, such as keys and check constraints, to simplify or rewrite a query into a cheaper equivalent
C. Translating SQL keywords into another natural language
D. Running a query on a sample of the data to guess its result

1258. Why do optimisers usually minimise total resource consumption rather than response time? Understanding

A. Response time depends on buffer contents and parallel overlap, which are hard to predict, while resource use can be estimated from statistics
B. Response time cannot be measured on a real DBMS
C. Minimising resource use always gives the fastest response time too
D. The SQL standard forbids optimising for response time

1259. While sorting tuples for GROUP BY dept, the DBMS computes AVG(salary) as groups are formed. What must it keep for each group? Understanding

A. The running sum and the running count of salaries
B. Only the running average so far
C. The minimum and maximum salary
D. Every salary in the group until the end

1260. What does a hybrid hash join do differently from a plain hash join? Understanding

A. It switches to a nested-loop join for every partition
B. It keeps the first partition of the build relation in memory, so that partition is never written to disk and read back
C. It sorts both relations before hashing them
D. It hashes only the probe relation, not the build relation

1261. For $\sigma A \geq v(r)$ on a file ordered on A with a clustering (primary) B+-tree index on A, why is the index so efficient? Understanding

A. The index finds the first tuple with $A \geq v$, and the rest of the result is then read sequentially from the ordered file
B. The index returns all matching tuples without reading any data blocks
C. The index turns the comparison into an equality test
D. The index removes the need for disk seeks altogether

1262. What does a command such as ANALYZE (PostgreSQL, SQLite) or UPDATE STATISTICS do for the optimiser? Understanding

A. It forces the optimiser to use every available index
B. It scans the data and refreshes the catalogue statistics that the cost estimates are based on
C. It runs the query and caches the result for later calls
D. It rebuilds all indexes in sorted order

1263. In the merge stage of external sort-merge with M memory blocks, how many runs can be merged in a single pass? Understanding

A. M, because every block can hold one input run
B. 2, because merging is always pairwise
C. M − 1, because one block is kept as the output buffer
D. M − 2, because one block is kept for the sort key and one for output

1264. A relation of 500 blocks is stored contiguously. Using Silberschatz's linear-search cost $tS + b_r \times tT$ with $tS = 4$ ms and $tT = 0.1$ ms, how long does a full scan take? Application

A. 54 ms
B. 50 ms
C. 2,000 ms
D. 2,054 ms

1265. A relation occupies 800 blocks. A linear search looks for the tuple with a given value of the primary key. On average, how many blocks does it read before it can stop? Application

A. 800
B. 400
C. 10
D. 1

1266. A relation has 12,000 tuples and a blocking factor of 40 tuples per block. How many blocks does it occupy? Application

A. 480,000
B. 3,000
C. 300
D. 30

1267. r has 300 tuples and s has 40 tuples, and they have no attributes in common. How many tuples does $r \bowtie s$ (natural join) contain? Application

A. 12,000
B. 340
C. 300
D. 0

1268. account has 8,000 tuples and V(status, account) = 4. Assuming a uniform distribution, what is the estimated size of σstatus ≠ 'closed'(account)? Application

A. 2,000
B. 8,000
C. 4,000
D. 6,000

1269. A 1,000-block file is sorted on A. Using the binary-search cost $\lceil \log_2(b_r) \rceil \times (tT + tS)$ for an equality on A, with $tT = 0.1$ ms and $tS = 4$ ms, what is the cost? Application

A. 54 ms
B. 1.0 ms
C. 41 ms
D. 104 ms

1270. An equality selection on a candidate key uses a B+-tree index of height $hi = 3$. Using Silberschatz's cost $(hi + 1) \times (tT + tS)$ with $tT = 0.1$ ms and $tS = 4$ ms, what is the cost? Application

A. 12.3 ms
B. 4.4 ms
C. 16.0 ms
D. 16.4 ms

1271. σdept='CS'(emp) uses a secondary B+-tree index on dept of height 3, and 20 tuples match, each in a different block. Using (hi + n) × (tT + tS) with tT = 0.1 ms and tS = 4 ms, what is the cost? Application

A. 14.3 ms
B. 82.0 ms
C. 94.3 ms
D. 16.4 ms

1272. r has 200 blocks and s has 50 blocks. In the worst case of block nested-loop join, memory holds only one block of each relation. With r as the outer relation, how many block transfers are needed? Application

A. 10,050
B. 10,200
C. 250
D. 10,000

1273. The outer relation r has 2,000 tuples in 100 blocks. The inner relation s has a B+-tree index on the join attribute, and each lookup costs 4 block accesses. Counting block accesses as b_r + n_r × c, what does an index nested-loop join cost? Application

A. 8,000
B. 2,100
C. 400
D. 8,100

1274. A 5,000-block relation is sorted by external sort-merge with M = 21 buffer blocks. How many merge passes are needed, using ⌈log_{M−1}(b_r / M)⌉? Application

A. 2
B. 3
C. 1
D. 8

1275. A hash join of r (400 blocks) and s (100 blocks) uses n_h = 5 partitions. Including the partly filled blocks, Silberschatz's cost is 3(b_r + b_s) + 4n_h. How many block transfers is that? Application

A. 1,520
B. 1,500
C. 1,505
D. 2,000

1276. What does SQLite 3.50 report in EXPLAIN QUERY PLAN for the SELECT in this script? Application

```
CREATE TABLE emp(id INTEGER PRIMARY KEY, name TEXT, dept TEXT, salary INT);
CREATE INDEX i ON emp(dept, name);
EXPLAIN QUERY PLAN SELECT name FROM emp WHERE dept = 'CS';
```

A. SEARCH emp USING COVERING INDEX i (dept=?)
B. SEARCH emp USING INDEX i (dept=?)
C. SCAN emp
D. SCAN emp USING COVERING INDEX i

1277. emp has no index other than its INTEGER PRIMARY KEY. What does SQLite 3.50's EXPLAIN QUERY PLAN report for this query? Application

```
CREATE TABLE emp(id INTEGER PRIMARY KEY, name TEXT, dept TEXT, salary INT);
EXPLAIN QUERY PLAN SELECT name FROM emp ORDER BY name;
```

A. SEARCH emp USING INTEGER PRIMARY KEY (rowid=?)
B. SCAN emp, then USE TEMP B-TREE FOR ORDER BY
C. SCAN emp only, because rows are already stored in name order
D. SCAN emp, then USE TEMP B-TREE FOR DISTINCT

1278. exam has 4,000 tuples, and marks ranges uniformly from min = 0 to max = 100. What is the estimated size of σmarks ≤ 30(exam), using n_r × (v − min) / (max − min)? Application

A. 2,800
B. 1,200
C. 40
D. 1,333

1279. r has 20,000 tuples. Three conditions have estimated sizes s1 = 2,000, s2 = 5,000 and s3 = 400. Assuming independence, what is the estimated size of σθ1∧θ2∧θ3(r)? Application

A. 400
B. 500
C. 10
D. 7,400

1280. Counting bushy trees and treating r1 ⋈ r2 and r2 ⋈ r1 as different, how many join orders exist for 5 relations, using (2(n − 1))! / (n − 1)!? Application

A. 120
B. 1,680
C. 30,240
D. 14

1281. r (outer, 100 blocks) and s (1,000 blocks) are joined by block nested-loop with M = 12. Silberschatz gives ⌈b_r/(M−2)⌉ × b_s + b_r transfers and 2⌈b_r/(M−2)⌉ seeks. With tT = 0.1 ms and tS = 4 ms, what is the total time? Application

A. 1,090 ms
B. 1,010 ms
C. 982 ms
D. 1,050 ms

1282. An equi-width histogram on age (integers) has buckets 20–29: 400 tuples, 30–39: 600 tuples and 40–49: 1,000 tuples. Assuming values are uniform within each bucket, what is the estimated size of σ35 ≤ age ≤ 44(person)? Application

A. 1,600
B. 800
C. 667
D. 500

1283. σA ≥ 50(r) uses a clustering B+-tree index on A of height 3, and the matching tuples fill 50 consecutive blocks. Using hi × (tT + tS) + b × tT with tT = 0.1 ms and tS = 4 ms, what is the cost? Application

A. 217.3 ms
B. 21.4 ms
C. 5.0 ms
D. 17.3 ms

1284. emp has separate indexes ia on dept and ib on salary. According to SQLite 3.50's EXPLAIN QUERY PLAN, how is this OR query evaluated? Application

```
CREATE TABLE emp(id INTEGER PRIMARY KEY, name TEXT, dept TEXT, salary INT);
CREATE INDEX ia ON emp(dept);
CREATE INDEX ib ON emp(salary);
EXPLAIN QUERY PLAN
SELECT * FROM emp WHERE dept = 'CS' OR salary = 5000;
```

A. It searches ia for dept = 'CS' and ib for salary = 5000 separately and combines the rowids (MULTI-INDEX OR)
B. It scans the whole table, because an OR condition can never use an index
C. It searches only ia for dept = 'CS' and tests salary in memory
D. It searches ia for both conditions at once, using (dept=? OR salary=?)

1285. A 1,000-block relation has a secondary B+-tree index (height 3) on A. A query matches only 5 records, each in a different block. With tT = 0.1 ms and tS = 4 ms, which access path is cheaper? Analysis

A. The linear scan: 104 ms, versus 32.8 ms for the index
B. The linear scan, because a secondary index is always slower than a scan
C. The index: (3 + 5) × 4.1 = 32.8 ms, versus 104 ms for a linear scan
D. They cost the same, because both must read every block

1286. In SQLite, emp has an index on name, yet SELECT * FROM emp WHERE upper(name) = 'ALI' reports SCAN emp. What is the best explanation and fix? Analysis

A. SQLite never uses indexes in a WHERE clause; drop the index
B. The table has too few rows; insert more rows and the index will be used
C. The index stores name, not upper(name), so the condition cannot be matched to it; create an index on the expression upper(name) or store a normalised column
D. upper() makes the query invalid; the plan is only a warning

1287. After a bulk load of 2 million rows into orders, a report became slow. EXPLAIN shows the optimiser still expects 10 rows from orders and has chosen a nested-loop join. What is the most likely cause? Analysis

A. Nested-loop join is always the wrong algorithm, so it must be a bug
B. The orders table needs to be sorted on disk after every insert
C. The query must be rewritten without any joins
D. The catalogue statistics are stale, so the estimates no longer match the data; refreshing them with ANALYZE should give a better plan

1288. A report combines two queries with UNION. The two results can never share a row, but the report is slow on large data. Which change helps, and why? Analysis

A. Use INTERSECT, because it returns the same rows faster
B. Add DISTINCT to each query, because it reduces the union's work
C. Use UNION ALL, because it builds an index on the result
D. Use UNION ALL, because UNION must remove duplicates by sorting or hashing the whole result, and that work is wasted here

1289. r1(A, B), r2(B, C) and r3(C, D) are to be joined. Which join order should a heuristic optimiser avoid? Analysis

A. (r1 ⋈ r2) ⋈ r3, because it joins on B first
B. (r2 ⋈ r3) ⋈ r1, because it joins on C first
C. r1 ⋈ (r2 ⋈ r3), because it is not left-deep
D. (r1 ⋈ r3) ⋈ r2, because r1 and r3 share no attribute and their join is a Cartesian product

1290. r has 1,000 blocks and s has 100 blocks, both unsorted, with M = 12. Which method costs the fewest block transfers? Analysis

A. Hash join with s as build input: 3 × (1,000 + 100) = 3,300
B. Block nested-loop with s as outer: 10,100
C. Block nested-loop with r as outer: 11,000
D. Merge join: 1,100, because it reads each relation once

1291. Outer r has 5,000 tuples in 100 blocks; inner s has 2,000 blocks and an index on the join attribute costing 3 block accesses per lookup. With M = 52, which join costs fewer block transfers? Analysis

A. Index nested-loop, about 15,100, versus 200,100 for block nested-loop
B. Index nested-loop, because an index join is always cheaper than a scan-based join
C. Block nested-loop, about 200,100, versus 15,100 for index nested-loop
D. Block nested-loop, about 4,100, versus 15,100 for index nested-loop

1292. r has 10,000 tuples. $\sigma_{A=5}(r)$ is estimated at 500 tuples and $\sigma_{B=7}(r)$ at 200. A student estimates $\sigma_{A=5 \wedge B=7}(r)$ as 500 + 200 = 700. Assuming independence, what is the right estimate? Analysis

A. 700
B. 200
C. 100,000
D. 10

1293. A query ends with ORDER BY the join attribute. Plan P1 uses a hash join costing 3,000 transfers, then sorts the result for 2,000 more. Plan P2 uses a merge join costing 4,200 in total, including sorting its inputs. Which should the optimiser pick? Analysis

A. P1, because its join step (3,000) is cheaper than P2's (4,200)
B. P2 at 4,200, because merge join's output is already in the required order, so no final sort is needed
C. P1 at 3,000, because the ORDER BY can be ignored when costing
D. Either, because both cost the same once the sort is added

1294. r has 2,000 blocks and s has 200 blocks, with M = 20. Using $n_h = \lceil b_{build} / M \rceil$ and requiring $M > n_h + 1$ to avoid recursive partitioning, which hash-join setup is best? Analysis

A. Build on r: $n_h = 100$, no recursive partitioning, about 6,600 transfers
B. Build on s: recursive partitioning is needed, because $n_h = 10 > M$
C. Build on r, because the larger relation should always be the build input
D. Build on s: $n_h = 10$, no recursive partitioning, about 3 × 2,200 = 6,600 transfers

1295. In SQLite 3.50, emp has a composite index ab on (dept, salary). Why does this query report SCAN emp instead of using the index? Analysis

```
CREATE TABLE emp(id INTEGER PRIMARY KEY, name TEXT, dept TEXT, salary INT);
CREATE INDEX ab ON emp(dept, salary);
EXPLAIN QUERY PLAN SELECT * FROM emp WHERE salary = 5000;
```

A. Composite indexes can only be used for ORDER BY, never for WHERE
B. SQLite uses only the INTEGER PRIMARY KEY for searches
C. The index is used, but SCAN is just SQLite's name for an index search
D. salary is not the leftmost column of ab, so the index's (dept, salary) order cannot locate salary values directly

1296. For σA=5∧B=7(r), secondary indexes exist on A and on B. A=5 matches 2,000 tuples and B=7 matches 20. If the optimiser may use only one index, which plan is best? Analysis

A. Use the index on A to fetch the 2,000 tuples, then test B = 7 in memory
B. Use a linear scan, because a conjunction can never use an index
C. Use the index on B to fetch the 20 tuples, then test A = 5 on each in memory
D. Use the index on A, because the first condition written must be evaluated first

1297. takes has 10,000 tuples, and takes.ID is a foreign key to student.ID, the key of student. 1,000 takes rows have ID = NULL. How many tuples does student ⋈ takes (on ID) produce? Analysis

A. 10,000
B. 11,000
C. 9,000
D. 1,000

1298. emp (1,000 tuples, 100 blocks) is queried with a correlated subquery that scans dept (20 blocks) once per emp tuple. Rewritten as a hash join, the cost is 3(b_r + b_s). Which is cheaper, and by how much? Analysis

A. The hash join: 360 transfers versus 20,100 for the correlated version
B. The correlated subquery: 20,100 transfers versus 36,000 for the hash join
C. They cost the same, because both read emp and dept
D. The hash join: 120 transfers versus 2,000 for the correlated version

1299. In SQLite 3.50, SELECT * FROM emp WHERE dept = 'CS' ORDER BY salary must avoid both a full scan and a temporary sort B-tree. Which single index does this? Analysis

A. CREATE INDEX x ON emp(dept)
B. CREATE INDEX x ON emp(salary)
C. CREATE INDEX x ON emp(dept, salary)
D. CREATE INDEX x ON emp(salary, dept)

Database Security MCQs for NSCT

1300. Which statement lets the user ali read the rows of table marks? Understanding

A. GRANT READ ON marks TO ali;
B. ALLOW SELECT ON marks FOR ali;
C. GRANT SELECT ON marks TO ali;
D. GRANT ali SELECT ON marks;

1301. What does WITH GRANT OPTION add to GRANT SELECT ON marks TO ali? Understanding

A. Ali becomes the owner of the marks table
B. Ali may also update and delete rows in marks
C. Ali's privilege can no longer be revoked
D. Ali may pass the SELECT privilege on marks to other users

1302. Discretionary access control (DAC) in SQL databases is mainly implemented with which mechanism? Understanding

A. Security labels on every row and clearances on every user, set by the system
B. Privileges that owners grant and revoke with GRANT and REVOKE
C. Encrypting every column with a separate key
D. Firewalls that block the database port

1303. Which statement best describes mandatory access control (MAC)? Understanding

A. The system enforces access using security classes on data and clearances on users, and owners cannot override it
B. Each object's owner decides who may access it
C. Users get privileges through job roles such as 'clerk' or 'manager'
D. Every user may read everything but write only their own rows

1304. What is SQL injection? Understanding

A. Inserting a large number of rows to slow a database down
B. Adding a new index to speed up a query
C. An attack where user input is interpreted as part of the SQL statement, changing what it does
D. Copying a database to another server without permission

1305. Which is the most effective primary defence against SQL injection? Understanding

A. Hiding database error messages from users
B. Renaming tables so attackers cannot guess them
C. Using HTTPS between the browser and the web server
D. Parameterised queries (prepared statements) that send data separately from the SQL text

1306. In SQL, what is a role? Understanding

A. A named collection of privileges that can be granted to users or other roles
B. A user account with no password
C. A view that hides certain columns
D. A log of the statements a user has run

1307. What is the difference between authentication and authorisation in a DBMS? Understanding

A. Authentication decides what the user may do; authorisation checks who the user is
B. They are two names for the same password check
C. Authentication applies to tables; authorisation applies only to views
D. Authentication checks who the user is; authorisation decides what that user may do

1308. What does GRANT SELECT ON notices TO PUBLIC do? Understanding

A. Gives every current and future user the SELECT privilege on notices
B. Makes the notices table visible on the internet
C. Gives only the DBA the SELECT privilege
D. Makes notices read-only for its owner

1309. What is a database audit trail? Understanding

A. A backup copy of the database taken every night
B. The list of indexes on each table
C. A diagram of the database schema
D. A log recording which user performed which operations on which data, and when

1310. Under the Bell-LaPadula model, what does the simple security property state? Understanding

A. A subject may read an object only if the subject's clearance is at least the object's classification
B. A subject may write an object only if the object's classification is lower than the subject's clearance
C. A subject may read any object it created
D. A subject may write an object only if it can also delete it

1311. What is the purpose of the Bell-LaPadula *-property (star property), 'no write down'? Understanding

A. To stop low-cleared users from reading high-level objects
B. To stop users from deleting objects they did not create
C. To force all writes to be logged in an audit trail
D. To stop information read at a high level from being copied into objects that lower-cleared users can read

1312. How do views support database security? Understanding

A. A user can be granted access to a view that shows only certain rows or columns, without any privilege on the base table
B. A view encrypts the data stored in the base table
C. A view stops the DBA from reading the base table
D. A view automatically logs every query against it

1313. A statistical database allows only aggregate queries such as COUNT, SUM and AVG. What is the main security risk? Understanding

A. Aggregate functions bypass GRANT and REVOKE
B. Inference: combining several aggregate results can reveal an individual's data
C. Aggregate queries always need the DBA's password
D. Aggregates make SQL injection impossible, which hides attacks

1314. Why is the REFERENCES privilege controlled separately from SELECT? Understanding

A. REFERENCES lets the user read every column of the table
B. REFERENCES lets the user drop the table
C. A foreign key referencing a table can block deletes and updates of rows in that table
D. REFERENCES is needed to create an index on the table

1315. Encrypting database files at rest (for example with transparent data encryption) mainly protects against which threat? Understanding

A. Someone reading data from stolen disks or backup files
B. SQL injection through the web application
C. A legitimate user misusing privileges they hold
D. Two transactions updating the same row at once

1316. What does this Python program print? (It runs on SQLite.) Application

```
import sqlite3
con = sqlite3.connect(":memory:")
con.execute("CREATE TABLE users(name TEXT, pass TEXT)")
con.executemany("INSERT INTO users VALUES (?, ?)",
                [("ali", "x1"), ("sara", "x2"), ("admin", "x3")])
u = "ali"
p = "' OR '1'='1"
sql = "SELECT * FROM users WHERE name = '" + u + "' AND pass = '" + p + "'"
print(len(con.execute(sql).fetchall()))
```

A. 1
B. 3
C. 0
D. An sqlite3.OperationalError message

1317. A login query is built as "SELECT * FROM users WHERE name = '" + u + "' AND pass = '" + p + "'". Which username, with any password, returns the admin row? Application

A. admin"--
B. --admin
C. admin'--
D. admin' AND '1'='2

1318. ali must be able to change only the salary column of emp. Which statement does this? Application

A. GRANT UPDATE ON emp (salary) TO ali;
B. GRANT UPDATE (salary) ON emp TO ali;
C. GRANT UPDATE ON emp TO ali WHERE column = 'salary';
D. GRANT ALTER (salary) ON emp TO ali;

1319. The DBA grants SELECT on emp to A WITH GRANT OPTION. A then grants SELECT on emp to B. The DBA runs REVOKE SELECT ON emp FROM A CASCADE. Who still holds SELECT on emp (apart from the DBA)? Application

A. Only B
B. Neither A nor B
C. Only A
D. A and B, unchanged

1320. The DBA grants SELECT on emp WITH GRANT OPTION to both A and B. A and B each grant SELECT on emp to C. The DBA then revokes A's privilege with CASCADE. Who holds SELECT now (apart from the DBA)? Application

A. Only B
B. Only C
C. A, B and C
D. B and C

1321. A holds SELECT on emp WITH GRANT OPTION from the DBA and has granted SELECT on emp to B. What happens when the DBA runs REVOKE SELECT ON emp FROM A RESTRICT? Application

A. A and B both lose the privilege
B. The REVOKE fails, because B's privilege depends on A's
C. Only A loses it; B keeps it
D. Only B loses it; A keeps it

1322. A holds SELECT on emp WITH GRANT OPTION and has not granted it to anyone. The owner runs REVOKE GRANT OPTION FOR SELECT ON emp FROM A CASCADE. What can A do afterwards? Application

A. Nothing on emp; A has lost SELECT entirely
B. Grant SELECT to others but not query emp
C. Still SELECT from emp, but no longer grant SELECT to others
D. Everything as before; the statement has no effect

1323. What does this Python program print? Application

```
import sqlite3
con = sqlite3.connect(":memory:")
con.execute("CREATE TABLE users(name TEXT, pass TEXT)")
con.executemany("INSERT INTO users VALUES (?, ?)",
                [("ali", "x1"), ("sara", "x2"), ("admin", "x3")])
u = "' OR '1'='1"
rows = con.execute("SELECT * FROM users WHERE name = ?", (u,)).fetchall()
print(len(rows))
```

A. 0
B. 3
C. 1
D. An sqlite3.OperationalError message

1324. What does this Python program print? Application

```
import sqlite3
con = sqlite3.connect(":memory:")
con.execute("CREATE TABLE orders(id INT, cust TEXT)")
con.executemany("INSERT INTO orders VALUES (?, ?)",
                [(1, "ali"), (2, "sara"), (3, "ali"), (4, "omar")])
oid = "0 OR 1=1"          # value typed into a form field
sql = "SELECT * FROM orders WHERE id = " + oid
print(len(con.execute(sql).fetchall()))
```

A. 0
B. 4
C. 1
D. An sqlite3.OperationalError message

1325. Under Bell-LaPadula, a subject has Secret clearance. Classes are ordered Top Secret > Secret > Confidential > Unclassified. Which operation is permitted? Application

A. Reading a Confidential object
B. Reading a Top Secret object
C. Writing a Confidential object
D. Writing an Unclassified object

1326. A clerk must see employee names and departments but never salaries in emp(id, name, dept, salary). Which approach meets this? Application

A. GRANT SELECT ON emp TO clerk; then REVOKE SELECT ON emp.salary FROM clerk;
B. GRANT SELECT ON emp TO clerk WITH GRANT OPTION;
C. CREATE VIEW emp_pub AS SELECT id, name, dept FROM emp; then GRANT SELECT ON emp_pub TO clerk;
D. GRANT SELECT ON emp TO clerk; then ask the clerk not to select salary

1327. These run in order: CREATE ROLE teacher; GRANT teacher TO ali; GRANT SELECT ON marks TO teacher. Can ali (with the role active) SELECT from marks? Application

A. No; ali only gets privileges the role had when ali was added
B. No; roles can only hold privileges on views
C. Yes; privileges granted to a role later still reach its members
D. Only if the role was granted WITH ADMIN OPTION

1328. A search page runs "SELECT name, price FROM products WHERE name LIKE '%" + s + "%'". The input ' UNION SELECT pass FROM users-- causes an error. Why? Application

A. UNION cannot be used inside a WHERE clause
B. Both sides of UNION must return the same number of columns, and the injected SELECT returns one instead of two
C. The -- comment is not valid SQL
D. SQLite does not allow UNION with a LIKE condition

1329. DBA → A and A → B are grants of SELECT WITH GRANT OPTION. Then B grants it to C WITH GRANT OPTION, and C grants it back to B WITH GRANT OPTION. The DBA revokes A's privilege with CASCADE. Who still holds SELECT (apart from the DBA)? Application

A. B and C
B. Only B
C. Nobody
D. Only C

1330. Only aggregate queries are allowed. SUM(salary) for the 10 CS staff is 1,250,000. SUM(salary) for the 9 CS staff aged under 50 is 1,040,000. You know exactly one CS staff member, Dr Raza, is 50 or over. What is revealed? Analysis

A. Only the average CS salary, 125,000
B. Nothing, because both queries return aggregates
C. The average salary of staff under 50, about 115,556, and nothing about Dr Raza
D. Dr Raza's salary is 210,000

1331. Why do prepared statements stop SQL injection, while escaping input by hand is considered weaker? Analysis

A. Prepared statements encrypt the input before sending it
B. The SQL structure is parsed before the data is bound, so input can never change the statement, whereas escaping can miss contexts
C. Prepared statements reject any input containing quotes
D. Prepared statements run with fewer privileges than ordinary queries

1332. A developer doubles every single quote in user input before building "... WHERE id = " + oid. An attacker still dumps every row with the input 0 OR 1=1. Why did escaping fail? Analysis

A. Doubling quotes is never valid SQL
B. OR is a reserved word that escaping cannot handle
C. The attacker must have known the DBA password
D. The value is placed in a numeric context with no quotes, so the attack needs no quote characters

1333. A sign-up form stores the username admin'-- safely with a parameterised INSERT. Later, a password-change routine builds UPDATE users SET pass = '...' WHERE name = '<stored name>' by string concatenation. What is the flaw? Analysis

A. Second-order injection: stored data is trusted later and concatenated into a new query
B. The parameterised INSERT should have rejected the username, so the INSERT is the flaw
C. There is no flaw, because the data was stored safely
D. The flaw is that passwords are stored in a table called users

1334. An injected input in a shop's web app ran DROP TABLE orders. The app connects as the schema owner. After switching to parameterised queries, which further change best limits the damage of any future flaw? Analysis

A. Hide all database error messages from users
B. Rename the orders table to something hard to guess
C. Grant the app account WITH GRANT OPTION so it can manage its own privileges
D. Connect with an account that has only the SELECT, INSERT and UPDATE privileges the app needs, and no DROP rights

1335. DBA → A (WITH GRANT OPTION), A → B (WITH GRANT OPTION), B → C (no grant option) and DBA → B (no grant option) are SELECT grants on emp. The DBA revokes A's privilege with CASCADE. What is the result? Analysis

A. B and C both keep SELECT, and B keeps the grant option
B. B and C both lose SELECT
C. B keeps SELECT but loses the grant option, and C loses SELECT
D. B keeps SELECT with the grant option, and C loses SELECT

1336. The DBA wants to remove A's SELECT on emp, which A received WITH GRANT OPTION and passed to B. B must keep access. What should the DBA do? Analysis

A. Revoke A's privilege with RESTRICT; B will keep access automatically
B. Revoke A's privilege with CASCADE; B keeps access because B used it recently
C. Grant SELECT on emp to B directly first, then revoke A's privilege with CASCADE
D. Revoke B's privilege first, then A's

1337. To fix injection, a developer moves the login query into a stored procedure that builds the SQL by concatenating its parameters and runs it with EXECUTE. Is the procedure safe? Analysis

A. No; dynamic SQL built by concatenation inside a procedure is still injectable
B. Yes; stored procedures are always immune to SQL injection
C. Yes, provided the procedure is owned by the DBA
D. No, because stored procedures cannot accept string parameters

1338. A login page only ever replies 'found' or 'not found'. The username admin' AND substr(pass,1,1)='x'-- gives 'found', and the same input with 'y' gives 'not found'. What does the attacker learn, and what is this technique? Analysis

A. Nothing, because the page shows no data; injection has failed
B. The admin password starts with x; boolean-based blind SQL injection
C. The admin password is x; UNION-based injection
D. The page is safe; it is using a prepared statement

1339. B legitimately holds SELECT on salary. A program B runs copies all salaries into a table B owns and grants SELECT on it to C. Why can DAC not stop this, and what can? Analysis

A. DAC can stop it if the salary owner uses REVOKE ... RESTRICT
B. Nothing can stop it, because B was authorised to read the data
C. Encrypting the salary table at rest blocks the copy
D. DAC controls access to each object, not where data flows after it is read; MAC's no-write-down rule can block the copy

1340. A flood of requests makes the university's database server unreachable during exam registration. Which security goal has been violated? Understanding

A. Confidentiality
B. Integrity
C. Non-repudiation
D. Availability

1341. In SQL, which privileges does a user automatically hold on a table they have just created? Understanding

A. Only SELECT, until the DBA grants more
B. All privileges on it, including the right to grant them to others, because the creator is the owner
C. No privileges, until the DBA grants them
D. Only the privileges held by PUBLIC

1342. What does GRANT ALL PRIVILEGES ON marks TO ali give ali? Understanding

A. Ownership of marks, so ali can drop it
B. DBA rights over the whole database
C. Every privilege on every table in the database
D. Every privilege on marks that the grantor is allowed to grant

1343. Which SQL statement takes away a privilege that was previously given to a user? Understanding

A. DENY ALL
B. DROP USER
C. REVOKE
D. ROLLBACK

1344. What does the principle of least privilege mean for database accounts? Understanding

A. Each user or program gets only the privileges needed for its job, and no more
B. Every user gets the same minimum set of privileges
C. The DBA account is used only for the least important tasks
D. Privileges are granted only to the user with the lowest clearance

1345. A hospital moves nurses between wards every month. Why does role-based access control make this easier than granting privileges to each nurse directly? Understanding

A. Privileges are attached to roles such as ward_nurse, so moving a nurse means changing only which roles they hold
B. Roles let nurses grant privileges to each other freely
C. Roles remove the need for nurses to log in
D. Roles encrypt the data each nurse can see

1346. In mandatory access control, what is a clearance? Understanding

A. The security class assigned to a table or tuple
B. A privilege granted with GRANT
C. The security class assigned to a subject (user or program)
D. The password a user types to log in

1347. Which measure protects data travelling between an application server and the database server from eavesdroppers on the network? Understanding

A. Encrypting the database files on disk
B. Granting SELECT only through views
C. Encrypting the connection with TLS
D. Keeping an audit trail of all queries

1348. How should a database store user passwords? Understanding

A. As plain text in a column only the DBA can read
B. Encrypted with a key stored in the same table
C. As unsalted MD5 hashes
D. As salted hashes from a slow password-hashing function such as bcrypt or Argon2

1349. An attacker adds a condition that makes the database pause for 5 seconds only when a guess is true, then times the page's response. What is this technique called? Understanding

A. UNION-based SQL injection
B. Error-based SQL injection
C. Time-based blind SQL injection
D. Stored cross-site scripting

1350. What does separation of duties mean in database administration? Understanding

A. Each table is stored on a separate disk
B. Developers and users connect through different networks
C. Every user has a separate copy of the database
D. A sensitive task is split so that no single person can complete it alone, for example one user creates a payment and another approves it

1351. A change to exam marks is made by someone with no right to make it. Which security goal has been violated? Understanding

A. Availability
B. Confidentiality
C. Integrity
D. Anonymity

1352. Support staff see card numbers as XXXX-XXXX-XXXX-4321 while the full number stays in the database. What is this control called? Understanding

A. Polyinstantiation
B. Data masking
C. Transparent data encryption
D. Referential integrity

1353. In the access matrix model of authorisation, what do the rows, columns and cells represent? Understanding

A. Rows are tables, columns are attributes, and cells are data values
B. Rows are transactions, columns are locks, and cells are lock modes
C. Rows are passwords, columns are users, and cells are hashes
D. Rows are subjects (users), columns are objects (tables), and each cell lists the subject's rights on that object

1354. What does the strong *-property (strict star property) add to Bell-LaPadula's ordinary *-property? Understanding

A. A subject may read only objects at exactly its own level
B. A subject may write down if the DBA approves
C. A subject may write only to objects at exactly its own level, so no write up is allowed either
D. A subject may write to any level but must log each write

1355. In a multilevel secure database, what is polyinstantiation? Understanding

A. Copying a table to several servers for availability
B. Creating many views on one base table
C. Granting the same privilege to many users at once
D. Several tuples with the same primary key exist at different security levels, each showing users at that level a different version

1356. A statistical database rejects any query whose result set has fewer than k records. Why is this control alone not enough? Understanding

A. It blocks every aggregate query
B. An attacker can combine several allowed queries, for example totals over large sets that differ by one person (a tracker), to isolate an individual
C. It lets users read individual records directly
D. It works only for COUNT and never for SUM

1357. How does adding random noise (perturbation) to aggregate results protect a statistical database? Understanding

A. It makes the stored data unreadable to the DBA
B. Answers stay useful on average, but subtracting two answers no longer reveals one person's exact value
C. It stops all queries that return fewer than k rows
D. It encrypts each answer with the user's key

1358. What is row-level security (also called a virtual private database)? Understanding

A. Every row is encrypted with a different key
B. Only the first row of each table can be read without a password
C. A privilege that lets a user insert exactly one row
D. The DBMS automatically adds a policy predicate, such as branch = the user's branch, to every query on a table

1359. A view is defined as SELECT * FROM emp WHERE dept = 'CS' WITH CHECK OPTION. What does WITH CHECK OPTION prevent? Understanding

A. Reading rows through the view without a password
B. Granting privileges on the view to other users
C. Deleting the view without the DBA's approval
D. Inserting or updating rows through the view so that they would not satisfy dept = 'CS'

1360. In multilevel security, what is a covert channel? Understanding

A. An encrypted connection between two database servers
B. A hidden admin account created by the DBA
C. A view that hides some columns of a table
D. An indirect way to pass information from a high level to a low level, for example by varying resource use or timing that a low subject can observe

1361. Following Silberschatz, which privileges does a user receive on a view they create over emp? Understanding

A. All privileges on the view, whatever they hold on emp
B. Ownership of emp as well as of the view
C. Only the privileges held by PUBLIC on emp
D. No more than they already hold on emp; for example, SELECT on the view requires SELECT on emp

1362. Why does Silberschatz suggest granting privileges with GRANTED BY CURRENT_ROLE instead of as the individual user? Understanding

A. If the grantor later leaves and is revoked, the grants stay because they belong to the role rather than to the person
B. It makes the grantee's privileges expire automatically
C. It hides the grant from the audit trail
D. It lets the grantee bypass mandatory access control

1363. Which part of an SQL statement can a prepared-statement placeholder (?) NOT stand for? Understanding

A. A string value in a WHERE condition
B. A number in a WHERE condition
C. A value in an INSERT's VALUES list
D. An identifier such as a table or column name

1364. ali must no longer be able to delete rows from marks, but should keep every other privilege. Which statement does this? Application

A. REVOKE ALL PRIVILEGES ON marks FROM ali;
B. REVOKE DELETE ON marks FROM ali;
C. GRANT DELETE ON marks TO ali WITH REVOKE;
D. DROP DELETE ON marks FOR ali;

1365. The role clerk must be able to add and remove orders, but not read or change them. Which statement does this? Application

A. GRANT ALL PRIVILEGES ON orders TO clerk;
B. GRANT INSERT, DELETE ON orders TO clerk;
C. GRANT INSERT, UPDATE ON orders TO clerk;
D. GRANT clerk TO INSERT, DELETE ON orders;

1366. Under Bell-LaPadula's *-property (no write down), a subject with Confidential clearance may write to which object? Levels: Top Secret > Secret > Confidential > Unclassified. Application

A. An Unclassified object
B. No object at any level
C. A Secret object
D. Only objects it can also read at a lower level

1367. A user holds only UPDATE (salary) on emp. They run UPDATE emp SET dept = 'HR' WHERE id = 3. What happens? Application

A. The row is updated, because the user has UPDATE on emp
B. Only the WHERE clause runs; no row changes but no error is raised
C. The statement is rejected, because the user may update only the salary column
D. The dept column is set to NULL instead

1368. DBA → A and A → B are SELECT grants WITH GRANT OPTION, and B → C is a SELECT grant without it. A runs REVOKE SELECT ON emp FROM B CASCADE. Who holds SELECT on emp afterwards, apart from the DBA? Application

A. A and C
B. A only
C. A, B and C
D. Nobody

1369. The DBA grants SELECT on emp to A and to B, both WITH GRANT OPTION. A and B each grant SELECT to C. A then revokes the privilege it gave C. Does C still hold SELECT? Application

A. Yes, because C's grant from B is still in place
B. No, because any revoke removes all of C's grants
C. No, because C first received the privilege from A
D. Yes, but only until B logs in again

1370. B holds SELECT on emp but not WITH GRANT OPTION. B runs GRANT SELECT ON emp TO C. What is the result? Application

A. C receives SELECT, because B holds it
B. The grant is not allowed, because B lacks the grant option, so C gets nothing
C. C receives SELECT WITH GRANT OPTION
D. B loses its own SELECT privilege

1371. These statements run in order: CREATE ROLE instructor; CREATE ROLE dean; GRANT SELECT ON marks TO instructor; GRANT instructor TO dean; GRANT dean TO sara. Can sara, with the dean role active, SELECT from marks? Application

A. Yes, because dean inherits instructor's privileges and sara holds dean
B. No, because roles cannot be granted to other roles
C. No, because SELECT was granted to instructor, not to dean
D. Only if sara is also granted instructor directly

1372. ali holds SELECT on marks directly and also INSERT on marks through the role teacher. The DBA runs REVOKE teacher FROM ali. What can ali do on marks now? Application

A. Nothing, because revoking a role removes every privilege on marks
B. SELECT only, because the INSERT came only through the role
C. SELECT and INSERT, because role privileges are copied to the user
D. INSERT only, because direct grants are revoked with the role

1373. What does this Python program print? Application

```
import sqlite3
con = sqlite3.connect(":memory:")
con.execute("CREATE TABLE products(name TEXT, price INT)")
con.execute("CREATE TABLE users(name TEXT, pass TEXT)")
con.executemany("INSERT INTO products VALUES (?, ?)",
                [("pen", 50), ("ink", 120)])
con.executemany("INSERT INTO users VALUES (?, ?)",
                [("ali", "x1"), ("sara", "x2"), ("admin", "x3")])
s = "zzz' UNION SELECT name, pass FROM users--"
sql = "SELECT name, price FROM products WHERE name = '" + s + "'"
print(len(con.execute(sql).fetchall()))
```

A. 0
B. 5
C. 3
D. An sqlite3 error, because UNION is not allowed in a WHERE clause

1374. A clerk is given access only to the view cs_emp. What does this Python program print? Application

```
import sqlite3
con = sqlite3.connect(":memory:")
con.execute("CREATE TABLE emp(name TEXT, dept TEXT, salary INT)")
con.executemany("INSERT INTO emp VALUES (?, ?, ?)",
    [("ali", "CS", 90), ("sara", "CS", 95), ("omar", "EE", 80),
     ("hina", "CS", 70), ("bilal", "EE", 85)])
con.execute("CREATE VIEW cs_emp AS SELECT name, dept FROM emp WHERE dept = 'CS'")
cur = con.execute("SELECT * FROM cs_emp")
print(len(cur.fetchall()), len(cur.description))
```

A. 3 2
B. 5 3
C. 3 3
D. 5 2

1375. A statistical database has 100 records and uses query-set-size control with k = 5: a query is answered only if its set has at least k and at most n − k records. Which query set size is answered? Application

A. 3
B. 50
C. 97
D. 98

1376. Bell-LaPadula with categories: subject S has class (Secret, {Nuclear, Army}). A subject may read an object only if its class dominates the object's (level ≥ and categories ⊇). Which object can S read? Application

A. (Confidential, {Nuclear})
B. (Secret, {Navy})
C. (Top Secret, {Nuclear})
D. (Unclassified, {Nuclear, Navy})

1377. clerk holds only INSERT (name, dept) on emp(id, name, dept, salary). Which of these statements will clerk be allowed to run? Application

A. INSERT INTO emp(name, dept) VALUES ('Ali', 'CS');
B. INSERT INTO emp(name, dept, salary) VALUES ('Ali', 'CS', 90000);
C. UPDATE emp SET dept = 'CS' WHERE name = 'Ali';
D. SELECT name, dept FROM emp;

1378. Which Python call safely looks up an order whose id comes from a web form in the variable oid? Application

A. con.execute("SELECT * FROM orders WHERE id = %s" % oid)
B. con.execute(f"SELECT * FROM orders WHERE id = {oid}")
C. con.execute("SELECT * FROM orders WHERE id = ?", (oid,))
D. con.execute("SELECT * FROM orders WHERE id = '" + oid.replace("'", "''") + "'")

1379. An attacker submits the id 7 AND (CASE WHEN substr(pass,1,1)='a' THEN sleep(5) ELSE 0 END)=0 and the page takes 5 seconds longer than usual. Assuming sleep() pauses the database, what has the attacker learned? Application

A. The password column does not exist
B. The page is immune to SQL injection
C. The password is exactly one character long
D. The first character of the password column is 'a'

1380. SELECT grants on emp: DBA → A, DBA → B, A → C and B → C, all WITH GRANT OPTION, and C → D. The DBA revokes A's privilege with CASCADE, then B revokes C's privilege with CASCADE. Who holds SELECT now, apart from the DBA? Application

A. B and C
B. B only
C. B, C and D
D. B and D

1381. Bell-LaPadula with categories: subject S has class (Secret, {Nuclear}). Under the *-property, S may write to an object only if the object's class dominates S's. Which object may S write to? Application

A. (Top Secret, {Nuclear, Army})
B. (Secret, {})
C. (Confidential, {Nuclear})
D. (Top Secret, {Army})

1382. What does this Python program print? Application

```
import sqlite3
con = sqlite3.connect(":memory:")
con.execute("CREATE TABLE logs(msg TEXT)")
con.execute("CREATE TABLE notes(txt TEXT)")
t = "hi'); DROP TABLE logs; --"
con.executescript("INSERT INTO notes VALUES ('" + t + "');")
print([r[0] for r in con.execute(
    "SELECT name FROM sqlite_master WHERE type = 'table'")])
```

A. ['logs', 'notes']
B. ['notes']
C. ['notes', 'logs']
D. Nothing; it raises sqlite3.ProgrammingError for running more than one statement

1383. Aggregates only: the average salary of the 4 lecturers in the Maths department is 50,000. After Dr Asif joins Maths as its fifth lecturer, the average becomes 52,000. What is Dr Asif's salary? Application

A. 52,000
B. 2,000
C. 10,000
D. 60,000

1384. A holds SELECT on emp without the grant option. A creates the view v AS SELECT name, dept FROM emp and then runs GRANT SELECT ON v TO B. What happens? Application

A. B gets SELECT on v, because A owns the view
B. The grant fails, because granting on a view requires the grant option on the underlying table
C. B gets SELECT on v and on emp
D. The view cannot be created, because A lacks the grant option

1385. A web app's database files are encrypted at rest. An attacker uses SQL injection through the app and reads the customers table. Why did the encryption not help? Analysis

A. Queries that come through the DBMS see decrypted data, so encryption at rest protects only the stored files and backups
B. The encryption key was too short
C. SQL injection decrypts files by brute force
D. Encryption at rest works only on indexes, not tables

1386. All five developers and the web app log in with the same database account, dev_all. After a table is wrongly deleted, the audit log shows only 'dev_all'. What is the main weakness? Analysis

A. The audit log was not encrypted
B. dev_all should have a longer password
C. Shared accounts destroy accountability; each person and the app need their own account so the audit trail identifies who did what
D. The table should have had more indexes

1387. A form uses JavaScript in the browser to reject quotes before submitting the search box. A tester still performs SQL injection. How? Analysis

A. By disabling the database's TLS connection
B. By guessing the DBA password
C. By sending the HTTP request directly, for example with a proxy or curl, so the browser-side check never runs
D. By using a different SQL dialect that has no quotes

1388. A leaked users table stores unsalted SHA-256 password hashes. Many rows share the same hash. Why is this dangerous? Analysis

A. Without salt, identical passwords give identical hashes, so one cracked hash or precomputed table reveals every user with that password
B. SHA-256 hashes can be decrypted with the database's master key
C. Identical hashes mean the database is corrupted
D. Unsalted hashes cannot be checked at login

1389. To hide salaries, the DBA creates the view emp_pub (id, name, dept) and grants SELECT on it to clerk. Months later, clerk is found reading salaries. Which earlier action most likely explains this? Analysis

A. clerk was also granted SELECT on the base table emp, which the view cannot override
B. Views always expose every column of their base table
C. clerk used ORDER BY on the view
D. The view was created WITH CHECK OPTION

1390. Under Bell-LaPadula with the ordinary *-property, a Secret process must add entries to an audit log classified Top Secret. Which operations on that log are permitted? Analysis

A. It may append (write up) to the log but may not read it
B. It may read the log but may not write to it
C. It may both read and write the log
D. It may neither read nor write the log

1391. SELECT grants on emp, all WITH GRANT OPTION: DBA → A, DBA → B, A → C, C → D, D → C and B → D. The DBA revokes A's privilege with CASCADE. Who holds SELECT now, apart from the DBA? Analysis

A. B, C and D
B. B only
C. B and D
D. Nobody

1392. This search uses a parameter, yet it prints 3, returning every user. What is going on? Analysis

```
import sqlite3
con = sqlite3.connect(":memory:")
con.execute("CREATE TABLE users(name TEXT)")
con.executemany("INSERT INTO users VALUES (?)",
                [("ali",), ("sara",), ("admin",)])
term = "%"
rows = con.execute("SELECT name FROM users WHERE name LIKE ?",
                   (term,)).fetchall()
print(len(rows))
```

A. It is SQL injection: the parameter was pasted into the SQL text
B. SQLite ignores parameters in LIKE conditions
C. It is not SQL injection: % is a LIKE wildcard inside the bound value, so it matches everything; escape % and _ in the term if literal matching is wanted
D. The ? placeholder is replaced by a SQL comment

1393. A developer writes stmt = conn.prepare("SELECT * FROM users WHERE name = '" + u + "'") and then executes stmt. They say it is safe because it uses a prepared statement. Are they right? Analysis

A. Yes; preparing any statement makes it immune to injection
B. Yes, as long as the database account has least privilege
C. No; the input was concatenated into the SQL before preparing, so it is part of the statement's structure and can still inject
D. No, but only because prepare() is slower than execute()

1394. A statistical database adds fresh random noise to every answer. An analyst sends the same query 1,000 times and averages the answers. Why is this a problem, and what is a standard counter? Analysis

A. Averaging makes the answers less accurate, so nothing is revealed
B. Independent noise averages out, revealing the true value; give the same noisy answer to repeated identical queries or limit the total query budget
C. The DBMS will crash after 1,000 queries
D. Noise is removed automatically after 100 queries

1395. A login form shows raw database errors. Typing a single quote into the username shows: unrecognized token: "'" AND pass = '". What does this tell an attacker, and what is the best fix? Analysis

A. Only that the username is wrong; no fix is needed
B. That the database is down; restart the server
C. That quotes are blocked; the form is already safe
D. The input is pasted into SQL text, so the form is injectable; use parameterised queries and show users a generic error while logging the details

1396. The application account has SELECT, INSERT and DELETE on all tables, including audit_log, which records security events. Why is this a weakness even if the app is free of injection today? Analysis

A. Anyone who later compromises the app account can delete the audit entries that would expose them; the log should be append-only or kept where the app cannot delete
B. Audit logs must be stored in the same table as the data they describe
C. INSERT on audit_log lets attackers read all other tables
D. DELETE privileges slow down every SELECT query

1397. An Unclassified user tries to insert a tuple whose key already exists as a Secret tuple. Why do multilevel databases usually polyinstantiate instead of rejecting the insert as a duplicate key? Analysis

A. Rejecting the insert would violate the no-read-up rule directly
B. A rejection would tell the Unclassified user that hidden Secret data with that key exists, which is an inference leak
C. Duplicate keys are always allowed in relational databases
D. Polyinstantiation encrypts the Secret tuple so it can be shown

1398. A login page gives only 'found' or 'not found'. The admin password has 8 characters, each from a set of 64. Using boolean-based blind injection with binary search (one yes/no query per halving), how many queries are needed at most? Analysis

A. 512
B. 256
C. 8
D. 48

1399. The developer parameterised item but concatenated owner. What does this Python program print? Analysis

```
import sqlite3
con = sqlite3.connect(":memory:")
con.execute("CREATE TABLE cart(owner TEXT, item TEXT)")
con.executemany("INSERT INTO cart VALUES (?, ?)",
    [("ali", "pen"), ("ali", "ink"), ("sara", "pen"), ("omar", "cap")])
owner = "x' OR '1'='1"
con.execute("DELETE FROM cart WHERE item = ? AND owner = '" + owner + "'",
            ("pen",))
print(con.execute("SELECT COUNT(*) FROM cart").fetchone()[0])
```

A. 2
B. 4
C. 0
D. 3

Distributed Databases MCQs for NSCT

1400. What is a distributed database? Understanding

A. A database copied to a USB drive for each user
B. Several unrelated databases that happen to share a server
C. A database that is backed up to two different disks
D. A logically single database whose data is stored at several sites connected by a network

1401. Horizontal fragmentation divides a relation by what? Understanding

A. Rows (tuples), using a selection condition such as branch = 'Lahore'
B. Columns (attributes), using a projection
C. Indexes, putting each index on its own site
D. Transactions, putting each transaction on its own site

1402. In vertical fragmentation, why must every fragment include the primary key (or a tuple-id)? Understanding

A. So each fragment can be indexed by the DBMS
B. So the fragments can be combined with UNION
C. So the original relation can be rebuilt by joining the fragments on that key
D. So each fragment is stored at a different site

1403. Which operation rebuilds a relation from its horizontal fragments? Understanding

A. Natural join
B. Intersection
C. Cartesian product
D. Union

1404. What does fragmentation transparency mean in a distributed DBMS? Understanding

A. Users can see the fragments of every relation
B. Users query a relation without knowing how it is split into fragments
C. Every fragment is stored unencrypted
D. Fragments are replicated at every site

1405. In the CAP theorem, what do C, A and P stand for? Understanding

A. Consistency, Availability, Partition tolerance
B. Concurrency, Atomicity, Persistence
C. Consistency, Atomicity, Partitioning
D. Capacity, Availability, Performance

1406. With full replication, every site holds a copy of the whole database. What is its main drawback? Understanding

A. Reads must always go to a remote site
B. Availability falls when one site fails
C. Every update must be applied at every site, which makes writes expensive
D. Queries cannot use indexes

1407. What is the purpose of the two-phase commit (2PC) protocol? Understanding

A. To make a transaction that updates several sites either commit at all of them or abort at all of them
B. To lock data in two phases, growing and shrinking
C. To copy each update to two replicas
D. To split a query into two sub-queries

1408. What makes a distributed database heterogeneous? Understanding

A. Every site runs the same DBMS software
B. Data is fragmented horizontally
C. Each site has a different amount of data
D. Different sites may run different DBMS software or data models

1409. In synchronous (eager) replication, when are the replicas updated? Understanding

A. Within the same transaction, before it commits
B. Some time after the transaction commits
C. Only during the nightly backup
D. Only when a user reads a replica

1410. Which three rules must a correct fragmentation satisfy? Understanding

A. Atomicity, consistency and durability
B. Completeness, reconstruction and disjointness
C. Consistency, availability and partition tolerance
D. Locality, replication and transparency

1411. What is derived horizontal fragmentation? Understanding

A. Fragmenting a relation according to the fragments of another relation it references, using a semijoin
B. Fragmenting a relation by columns derived from a view
C. Combining horizontal and vertical fragmentation on one relation
D. Making a fragment by copying another site's fragment

1412. Why is two-phase commit called a blocking protocol? Understanding

A. Every participant locks the entire database during phase 1
B. It blocks all reads at every site until commit
C. A participant that voted ready may have to wait, holding locks, until a failed coordinator recovers
D. It forbids more than two participants

1413. According to the CAP theorem, when must a distributed system trade consistency against availability? Understanding

A. When a network partition occurs
B. On every read, even with no failures
C. Only when a site's disk fails
D. Only when more than half the sites are down

1414. What does BASE, often contrasted with ACID, stand for? Understanding

A. Balanced, Atomic, Serial, Encrypted
B. Basic Availability, Strict Ending
C. Backup, Archive, Store, Encrypt
D. Basically Available, Soft state, Eventual consistency

1415. In distributed query processing, why is a semijoin $r \ltimes s$ used? Understanding

A. To join r and s without any network messages
B. To reduce the data shipped between sites by sending only the r tuples that will join with s
C. To guarantee the result is sorted
D. To replicate r at the site of s

1416. A data item has N = 5 replicas and every write updates W = 3 of them. What is the smallest read quorum R that makes every read overlap the latest write (R + W > N)? Application

A. 2
B. 1
C. 4
D. 3

1417. emp(id, name, dept, salary) is split into F1 = πid,name,dept(emp) and F2 = πid,salary(emp). Which expression rebuilds emp? Application

A. F1 ∪ F2
B. F1 ⋈ F2 (natural join on id)
C. F1 × F2
D. F1 − F2

1418. staff is split into σsalary<50000(staff) and σsalary>50000(staff). Which correctness rule does this break? Application

A. Completeness: a staff member earning exactly 50,000 is in no fragment
B. Disjointness: some staff appear in both fragments
C. Reconstruction: horizontal fragments cannot be unioned
D. None: the fragmentation is correct

1419. staff is split into σage≤30(staff) and σage≥30(staff). Which correctness rule does this break? Application

A. Completeness: staff aged 30 are in no fragment
B. Reconstruction: the union does not give back staff
C. Disjointness: staff aged exactly 30 appear in both fragments
D. Durability: rows can be lost after a crash

1420. A data item has N = 7 replicas. Using the quorum consensus rules R + W > N and 2W > N, which (R, W) pair is valid? Application

A. R = 3, W = 5
B. R = 4, W = 3
C. R = 2, W = 5
D. R = 5, W = 3

1421. emp has rows (1, Ali, CS, 100), (2, Ali, EE, 200) and (3, Sara, CS, 150) for (id, name, dept, salary). It is split into F1 = πname,dept(emp) and F2 = πname,salary(emp). How many tuples does F1 ⋈ F2 contain? Application

A. 3
B. 5
C. 4
D. 9

1422. emp (10,000 rows × 100 bytes) is stored in Lahore and dept (100 rows × 50 bytes) in Karachi. The result of emp ⋈ dept is needed in Lahore. What is the least data that must be shipped? Application

A. 1,000,000 bytes, by sending emp to Karachi
B. 1,005,000 bytes, by sending both relations to a third site
C. 0 bytes, because joins never need data shipping
D. 5,000 bytes, by sending dept to Lahore

1423. r(A, B) at site S1 holds (1, 10), (2, 20), (3, 30), (4, 10), (5, 40). s(B, C) at S2 holds (10, x), (30, y), (50, z). Using a semijoin to compute r ⋈ s at S2, how many r tuples are shipped to S2? Application

A. 5
B. 2
C. 1
D. 3

1424. A coordinator runs two-phase commit with 4 participant sites and the transaction commits. Counting prepare, vote, commit and acknowledgement messages, how many messages are exchanged? Application

A. 8
B. 12
C. 16
D. 4

1425. In two-phase commit, three participants vote ready and one votes abort. What does the coordinator decide? Application

A. Global commit, because most participants voted ready
B. Commit at the three ready sites and abort at the fourth
C. It waits until the fourth participant changes its vote
D. Global abort, sent to the participants

1426. A data item uses read-one-write-all (ROWA) over 4 replicas, and one replica site goes down. What happens? Application

A. Writes can still succeed, but reads are blocked
B. Both reads and writes continue normally
C. Both reads and writes are blocked
D. Reads can still succeed, but writes cannot complete until the site returns

1427. customer is first split horizontally by region into C_N and C_S, and each is then split vertically into (cid, name, region) and (cid, limit). How is customer rebuilt? Application

A. Join the two vertical fragments of each region on cid, then union the two results
B. Union all four fragments directly
C. Join all four fragments on cid in one step
D. Intersect the two horizontal fragments

1428. Five replicas have weights 1, 1, 1, 2 and 2 (total S = 7), and the write quorum is Qw = 4. Using Qr + Qw > S, what is the smallest read quorum weight Qr? Application

A. 3
B. 4
C. 2
D. 5

1429. A store keeps N = 3 replicas. Which (R, W) setting can return stale data even with no failures? Application

A. R = 2, W = 2
B. R = 1, W = 1
C. R = 1, W = 3
D. R = 3, W = 1

1430. During a network partition, a replicated store refuses writes on the minority side so that no two sides accept conflicting updates. How is it classified under CAP? Analysis

A. AP: it gives up consistency to stay available
B. CA: it keeps both consistency and availability
C. CP: it gives up availability to keep consistency
D. It does not tolerate partitions, so CAP does not apply

1431. A bank has branches in Lahore, Karachi and Peshawar. About 90% of queries on account come from the branch that owns the account. Which design best reduces network traffic? Analysis

A. Vertical fragmentation of account by column, with each column at a different city
B. Horizontal fragmentation of account by branch, with each fragment stored at its branch
C. One central copy of account in Islamabad
D. Full replication of account at all three branches

1432. In 2PC, the coordinator crashes after collecting the votes but before anyone learns the decision. All surviving participants voted ready and none received a decision. What must they do? Analysis

A. Commit at once, because everyone voted ready
B. Abort at once, because the coordinator failed
C. Wait (block) until the coordinator recovers, because commit and abort are both still possible
D. Elect a new coordinator, which can safely decide either way

1433. A system uses N = 5 replicas with W = 3 and R = 2. A client writes a new value, then reads it immediately and gets the old value with no failures. Why? Analysis

A. W = 3 is not a majority of 5
B. R + W = 5 is not greater than N, so the read set can miss every replica the write reached
C. Reads always return old values until the next write
D. Two reads are needed because R is even

1434. A designer splits student(roll_no, name, dept, cgpa) into πname,dept and πname,cgpa, then rebuilds it with a natural join. Queries return students with CGPAs that are not theirs. What went wrong? Analysis

A. Vertical fragments must be rebuilt with UNION, not join
B. The fragments break the disjointness rule because name appears in both
C. The shared attribute name is not a key, so the join produces spurious tuples
D. Natural join loses rows when CGPAs are equal

1435. emp (1,000 rows × 100 bytes) is at S1; dept (50 rows) is at S2, with a 4-byte dept_id. Every employee's dept_id appears in dept, and the join result is needed at S2. Should a semijoin be used? Analysis

A. Yes: the semijoin always reduces the data shipped
B. Yes: it ships only 200 bytes in total
C. No: emp $\ltimes$ dept still contains all 1,000 rows, so the semijoin adds 200 bytes to the 100,000 shipped
D. No: semijoins cannot be used when a foreign key exists

1436. At site S1, T1 waits for T2. At site S2, T2 waits for T1. Each site's local wait-for graph has no cycle. What is the situation? Analysis

A. No deadlock, because neither local graph has a cycle
B. A local deadlock at S1 only
C. Starvation of T1, which resolves itself when T2 commits
D. A global deadlock that only a global wait-for graph (or timeouts) can detect

1437. Which statement about the CAP theorem is correct? Analysis

A. A system must always drop one of the three properties, even with no failures
B. CAP's 'consistency' means the same as the C in ACID
C. Without a partition a system can be consistent and available; the trade-off appears when a partition happens
D. A single-site database is AP

1438. A primary database commits a payment and replies to the client, then crashes before the change is sent to its asynchronous replica. The replica is promoted. What happens to the payment? Analysis

A. It is lost on the new primary, although the client was told it committed
B. It is safe, because a committed transaction is durable at every replica
C. It is rolled back at the client, which is notified automatically
D. It appears twice, once from the log and once from the replica

1439. orders is fragmented as orders ⋉ customer_north and orders ⋉ customer_south. There is no foreign key, and order 104 has cust_id 9, which no customer has. What happens to order 104? Analysis

A. It is placed in both fragments
B. It is in no fragment, so the fragmentation is not complete
C. It goes to the north fragment by default
D. The semijoin raises an error

1440. In a distributed database, what is a global transaction? Understanding

A. A transaction that reads or updates data stored at more than one site
B. A transaction that is submitted by a user at the central site
C. A transaction that only reads data and never updates it
D. A transaction that runs at one site but is logged at every site

1441. What does location transparency mean in a distributed DBMS? Understanding

A. Every site can see the physical disk layout of every other site
B. A user can name a relation or fragment without knowing which site stores it
C. Each user must state the site name in every query
D. Data is stored at the site closest to the database administrator

1442. What does replication transparency hide from users? Understanding

A. That the relation has been split into fragments
B. That the network may be slow between sites
C. That several copies of a data item exist and must be kept consistent
D. That different sites use different operating systems

1443. What makes a distributed database homogeneous? Understanding

A. All sites store exactly the same data
B. All sites are in the same city
C. All sites use asynchronous replication
D. All sites run the same DBMS software and know about each other

1444. What is partial replication? Understanding

A. Some data items are copied to some sites, but not every item is stored at every site
B. Every data item is stored at every site
C. Each data item is stored at exactly one site
D. Only the system catalogue is stored at more than one site

1445. In asynchronous (lazy) replication, when do the other replicas receive an update? Understanding

A. Before the transaction is allowed to commit
B. After the transaction commits at the primary, in a later propagation step
C. Only when an administrator copies the database manually
D. Never; each replica keeps its own version

1446. In primary-copy (single-master) replication, where must every update to a data item be applied first? Understanding

A. At whichever replica the client happens to contact
B. At every replica at the same instant
C. At the one site that holds the designated primary copy of that item
D. At the site with the most free disk space

1447. What is mixed (hybrid) fragmentation? Understanding

A. Storing some fragments on disk and others in memory
B. Replicating some fragments and not others
C. Splitting a relation into fragments of random size
D. Applying horizontal and vertical fragmentation one after the other to the same relation

1448. In distributed databases, what does local autonomy mean? Understanding

A. Each site manages its own data and operations with as little dependence on other sites as possible
B. Each site can read the private data of every other site
C. A central site approves every local query
D. Each site must use a different DBMS product

1449. What does the data allocation problem in distributed database design decide? Understanding

A. How many columns each table should have
B. At which sites each fragment or replica is stored
C. Which users are allowed to read each fragment
D. Which index type each site should use

1450. In classic distributed query optimisation over a wide-area network, which cost is usually treated as dominant? Understanding

A. The cost of sorting rows in memory
B. The cost of parsing the SQL text
C. The cost of transferring data between sites over the network
D. The cost of checking user passwords

1451. Under eventual consistency, what is guaranteed about replicas of a data item? Understanding

A. Every read returns the latest committed write
B. Replicas are updated within the same transaction
C. Replicas never disagree, even for a moment
D. If no new updates arrive, all replicas will eventually hold the same value

1452. Why does three-phase commit (3PC) add a pre-commit phase to two-phase commit? Understanding

A. So that participants can reach a decision without waiting for a failed coordinator, assuming no network partition
B. So that the transaction can commit even if one participant votes abort
C. So that fewer messages are sent than in 2PC
D. So that the coordinator does not need to write a log

1453. What does the PACELC model add to the CAP theorem? Understanding

A. A system can have all three CAP properties if it uses fast networks
B. Even when there is no partition, a system trades latency against consistency
C. Partition tolerance can be dropped in a geo-distributed system
D. Availability and consistency never conflict

1454. In two-phase commit, what must a participant do before it sends a ready (yes) vote? Understanding

A. Commit the transaction locally and release its locks
B. Ask the other participants how they will vote
C. Force a <ready T> record and the transaction's log records to stable storage
D. Delete the transaction's undo information

1455. During phase 1 of two-phase commit, one participant crashes and never replies to the prepare message. What does the coordinator do when its timeout expires? Understanding

A. Decide global commit, because the others voted ready
B. Wait forever for the crashed site to reply
C. Commit at the sites that voted ready and abort at the crashed site
D. Decide global abort, because a missing vote cannot be counted as ready

1456. In centralised distributed deadlock detection, what does the deadlock-detection coordinator do? Understanding

A. Combines the local wait-for graphs from all sites into a global graph and searches it for cycles
B. Aborts every transaction that waits longer than one second
C. Asks each site to check only its own local graph
D. Prevents deadlock by ordering all sites alphabetically

1457. How does a distributed DBMS usually make transaction timestamps unique across sites? Understanding

A. It uses the local clock alone, since clocks never match
B. It combines a local clock or counter value with the site's unique identifier
C. It asks one central site for every timestamp, with no other option
D. It uses the transaction's number of operations

1458. In the primary-copy locking protocol for replicated data, how is a data item locked? Understanding

A. By locking every replica of the item
B. By locking a majority of the replicas
C. By requesting the lock only at the site that holds the item's primary copy
D. By locking whichever replica answers first

1459. In the majority protocol for locking replicated data, when is a transaction granted a lock on item Q? Understanding

A. When it has obtained the lock at any one replica
B. When it has obtained the lock at every replica
C. When the primary site of Q grants it
D. When it has obtained the lock at more than half of the sites holding a replica of Q

1460. What is a federated (multidatabase) system? Understanding

A. A layer that lets users query several independent, possibly different, databases that each keep their own autonomy
B. A single DBMS that stores all data at one central site
C. A homogeneous system in which every site runs the same DBMS
D. A backup copy of a database kept at another site

1461. In the biased protocol for replicated data, how do shared and exclusive locks differ? Understanding

A. Both lock types must be taken at a majority of replicas
B. A shared lock is taken at one replica, while an exclusive lock must be taken at every replica
C. A shared lock is taken at every replica, while an exclusive lock is taken at one
D. Both lock types are taken only at the primary copy

1462. Simple predicates p1: salary < 50000 and p2: dept = 'CS' are used for primary horizontal fragmentation. What are minterm predicates? Understanding

A. The simple predicates p1 and p2 used on their own
B. Disjunctions such as salary < 50000 OR dept = 'CS'
C. Conjunctions that take every simple predicate in its true or negated form, such as salary < 50000 AND dept <> 'CS'
D. Predicates on the primary key only

1463. What is a phantom (false) deadlock in distributed deadlock detection? Understanding

A. A deadlock that involves transactions at only one site
B. A deadlock that resolves itself when the network recovers
C. A deadlock between a transaction and itself
D. A cycle the detector sees in the global wait-for graph that does not exist, caused by delayed or out-of-order messages

1464. A data item has N = 9 replicas and each write updates W = 6 of them. What is the smallest read quorum R that guarantees every read overlaps the latest write? Application

A. 3
B. 5
C. 6
D. 4

1465. With majority quorums over N = 5 replicas, what is the smallest write quorum W that satisfies 2W > N? Application

A. 3
B. 2
C. 4
D. 5

1466. student is horizontally fragmented by campus into F_LHR (campus = 'Lahore') and F_KHI (campus = 'Karachi'), and these are the only campuses. Which fragment must the DDBMS read for SELECT name FROM student WHERE campus = 'Karachi'? Application

A. F_LHR only
B. F_KHI only
C. Both fragments, then join them
D. Both fragments, then union them

1467. A relation with 1,000 rows is split into three horizontal fragments that satisfy completeness and disjointness. F1 has 420 rows and F2 has 310 rows. How many rows does F3 have? Application

A. 730
B. 580
C. 270
D. 420

1468. Under synchronous replication, a write must be acknowledged by all 3 replicas before commit. They acknowledge after 20 ms, 35 ms and 80 ms (in parallel). When can the write commit? Application

A. After 20 ms
B. After 45 ms
C. After 135 ms
D. After 80 ms

1469. emp(id, name, dept, salary) is vertically fragmented into F1(id, name, dept) at Lahore and F2(id, salary) at Karachi. Which fragments does the query 'name and salary of employee 7' need? Application

A. F1 and F2, joined on id
B. F1 only
C. F2 only
D. F1 and F2, combined with union

1470. Five replicas have weights 3, 2, 2, 1 and 1. The read quorum is Qr = 4. Using Qr + Qw > S and 2Qw > S, where S is the total weight, what is the smallest valid write quorum Qw? Application

A. 5
B. 6
C. 4
D. 9

1471. A data item has N = 8 replicas. Using R + W > N and 2W > N, which (R, W) pair is valid? Application

A. R = 4, W = 4
B. R = 2, W = 6
C. R = 3, W = 6
D. R = 5, W = 4

1472. A replicated item uses N = 5, R = 3 and W = 3. How many replica sites can be down while writes still succeed? Application

A. 3
B. 1
C. 0
D. 2

1473. emp (5,000 rows × 80 bytes) is at S1 and proj is at S2, where the result of emp ⋈ proj on emp_id is needed. πemp_id(proj) has 400 values of 4 bytes, and 600 emp rows match. What does the semijoin strategy transfer in total? Application

A. 49,600 bytes
B. 48,000 bytes
C. 400,000 bytes
D. 401,600 bytes

1474. R (1,000 KB) is at S1 and S (200 KB) is at S2. The join result is 50 KB and is needed at S3. Which plan transfers the least data? Application

A. Ship R and S to S3 and join there: 1,200 KB
B. Ship S to S1, join there, ship the result to S3: 250 KB
C. Ship R to S2, join there, ship the result to S3: 1,050 KB
D. Ship only S to S3: 200 KB

1475. customer_north holds customers with ids 1, 3 and 5. orders is fragmented by derived horizontal fragmentation: orders_north = orders ⋉ customer_north. The orders have cust_id values 1, 2, 3, 3, 5, 6, 1. How many rows does orders_north hold? Application

A. 3
B. 7
C. 5
D. 4

1476. Primary horizontal fragmentation of emp uses the simple predicates salary < 50000 and dept = 'CS'. How many minterm fragments are defined (before removing any that are empty)? Application

A. 2
B. 3
C. 8
D. 4

1477. staff is split into σdept='CS'(staff), σdept='EE'(staff) and σdept≠'CS'(staff). Which correctness rule does this fragmentation break? Application

A. Disjointness: every EE row appears in two fragments
B. Completeness: some staff rows appear in no fragment
C. Reconstruction: union cannot rebuild staff
D. None; the fragmentation is correct

1478. A participant in two-phase commit crashes and restarts. Its log has a <ready T> record but no <commit T> or <abort T> record. What must it do for T? Application

A. Abort T on its own, because it never saw a commit
B. Keep T in doubt and ask the coordinator (or other sites) for the decision
C. Commit T on its own, because it voted ready
D. Ignore T, because the coordinator will redo it

1479. Local wait-for edges: S1 has T1 → T2 and T3 → T4; S2 has T2 → T3; S3 has T4 → T1. Which transactions are in a deadlock? Application

A. Only T1 and T2
B. Only T3 and T4
C. T1, T2, T3 and T4
D. None, since no site has a cycle

1480. An item is replicated at 6 sites and locked with the majority protocol. At least how many sites must grant the lock? Application

A. 3
B. 6
C. 1
D. 4

1481. Global timestamps are pairs (local counter, site id), compared on the counter first and the site id second. Which timestamp is the oldest? Application

A. (14, 9)
B. (15, 2)
C. (15, 3)
D. (16, 1)

1482. Five replicas have weights 3, 2, 2, 1 and 1 (S = 9), and the write quorum is Qw = 6. Which set of failed sites still lets a write succeed? Application

A. The weight-3 site and one weight-1 site
B. One weight-2 site and one weight-1 site
C. Both weight-2 sites
D. One weight-2 site and both weight-1 sites

1483. r (1,000 rows × 60 bytes) is at S1 and s is at S2, where r ⋈ s is needed. πB(s) has 500 values of 4 bytes, and 300 r rows have a matching B. How many bytes does the semijoin plan using r ⋉ s transfer? Application

A. 18,000
B. 60,000
C. 20,000
D. 62,000

1484. An item has N = 7 replicas and reads use R = 2. What is the smallest W that satisfies R + W > N and 2W > N, and how many failed replicas can writes then tolerate? Application

A. W = 5; writes tolerate 2 failed replicas
B. W = 4; writes tolerate 3 failed replicas
C. W = 6; writes tolerate 2 failed replicas
D. W = 6; writes tolerate 1 failed replica

1485. A shop's product catalogue is read millions of times a day from three continents and changes a few times a day. Showing a price that is a few seconds old is acceptable. Which replication design fits best? Analysis

A. Replicas near each region, updated asynchronously from a primary
B. One copy only, at the head office
C. Synchronous replication to every region before each update commits
D. Horizontal fragmentation with no replicas

1486. A sensor log table at site S3 receives thousands of inserts per minute from S3, and other sites read it only for a monthly report. Where should it be stored? Analysis

A. Full replication at every site
B. A single copy at S3, with the monthly report reading it remotely
C. Synchronous copies at every site that runs the report
D. Vertical fragments spread across all sites

1487. A team picks three-phase commit to avoid blocking. Their sites are in different cities and links sometimes partition. What is the main risk? Analysis

A. 3PC always blocks, exactly like 2PC
B. 3PC cannot commit transactions that touch more than two sites
C. 3PC assumes no network partitions, so the two sides of a partition can reach different decisions
D. 3PC removes the need to write a log

1488. A replicated store keeps working through partitions, and on a healthy network it answers from the nearest replica without waiting for the others. How is it classified under PACELC? Analysis

A. PC/EC
B. PA/EC
C. PC/EL
D. PA/EL

1489. Head office runs payroll queries that read only (id, salary) for every employee. Branch offices read (id, name, dept, phone) for their own staff. Which design suits both? Analysis

A. Vertical fragmentation: (id, salary) at head office and (id, name, dept, phone) at the branches, split horizontally by branch
B. Horizontal fragmentation only, by branch
C. Full replication of emp at every site
D. One copy of emp at head office

1490. In 2PC the coordinator writes <commit T>, sends commit to P1, then crashes before telling P2. P2 voted ready and is in doubt. What is the best action for P2? Analysis

A. Abort, because it never heard from the coordinator
B. Ask P1; since P1 received commit, P2 can safely commit
C. Commit only if more than half the participants voted ready
D. Wait for the coordinator, because no other site can help

1491. Participants P1 and P2 vote ready; P3 crashes before it votes. The coordinator times out and decides abort. Why is this safe? Analysis

A. P3 will commit when it recovers, so the others should abort
B. Abort is always safe in 2PC, even after a commit has been sent
C. No participant could have committed, because a commit needs every vote to be ready
D. P1 and P2 may abort on their own after voting ready

1492. T1 releases a lock at S1 and then waits at S2 for a lock T2 holds. A centralised detector receives the new S2 edge before the release from S1, where T2 was waiting for T1. It reports a deadlock. What happened? Analysis

A. A real global deadlock that must be broken
B. A local deadlock at S2
C. A failure of two-phase commit
D. A false deadlock: the delayed release message made an old edge look current

1493. An item has N = 5 replicas with R = 3 and W = 3. A partition splits the sites into a group of 3 and a group of 2. What can each side do? Analysis

A. The group of 3 can read and write; the group of 2 can do neither
B. Both sides can read and write, and they reconcile later
C. The group of 2 can read but not write
D. Neither side can do anything until the partition heals

1494. Users update their profile on the primary and then reload a page served by an asynchronous read replica, which shows the old profile. Which fix gives read-your-writes for those users with the least cost? Analysis

A. Make every write synchronous to all replicas
B. Route each user's reads to the primary for a short time after that user writes
C. Add more read replicas
D. Switch from replication to horizontal fragmentation

1495. A designer uses N = 4, R = 3 and W = 2. R + W = 5 > 4, so they say the design is safe. Which problem remains? Analysis

A. Reads can miss the latest write
B. Writes need all four replicas
C. Two concurrent writes can succeed on disjoint replica pairs, because 2W is not greater than N
D. Reads need all four replicas

1496. student is fragmented by campus into two disjoint fragments with the same columns. A developer rebuilds it with F_LHR ⋈ F_KHI (natural join). What will the query return? Analysis

A. All students, as with a union
B. Duplicate rows for every student
C. Only Lahore students
D. No rows, because the natural join on all common columns keeps only rows found in both fragments

1497. Under timestamp ordering, site S2's clock runs well behind the other sites, and S2's transactions keep being rolled back. What explains this, and what is a standard fix? Analysis

A. S2 issues timestamps that look older than items' read/write timestamps; advance a site's clock when it sees a larger timestamp
B. S2's transactions get timestamps that are too new; slow down the other clocks
C. Timestamp ordering always rolls back remote transactions; switch to 2PC
D. S2 has too many locks; add more lock managers

1498. Two data centres keep accepting writes during a partition and resolve conflicts by last-writer-wins on timestamps. Both sides update the same account balance. What happens when the partition heals? Analysis

A. Both updates are applied in order
B. One side's update is silently discarded
C. The system blocks until an operator merges the values
D. Both values are kept and the database rejects reads

1499. An item is read 200 times a day at S1, 500 at S2 and 100 at S3, and updated 400 times a day from S1. Each remote read or update propagation is one message. Which allocation needs the fewest messages? Analysis

A. A single copy at S2: 700 messages
B. Full replication at all three sites: 800 messages
C. A single copy at S1: 600 messages
D. A single copy at S3: 1,100 messages

NoSQL & Modern Databases MCQs for the NSCT

1500. Which NoSQL data model stores self-describing records such as JSON documents and lets you query on fields inside each record? Understanding

A. Key-value store
B. Graph database
C. Document store
D. Column-family store

1501. What best describes a key-value store? Understanding

A. Data is stored as nodes and edges with properties on both
B. Rows are grouped into column families and each row can have different columns
C. Records must follow a fixed schema declared with CREATE TABLE
D. Each value is looked up by a unique key, and the database treats the value as an opaque blob

1502. Which kind of query is a graph database designed to answer efficiently? Understanding

A. Summing sales per region over ten years of history
B. Fetching a session object when you already know its session ID
C. Storing large video files in fixed-size blocks
D. Finding the shortest chain of connections between two people in a social network

1503. In NoSQL literature, what does BASE stand for? Understanding

A. Basically Available, Soft state, Eventually consistent
B. Balanced, Atomic, Serialisable, Encrypted
C. Basically Atomic, Strongly consistent, Eventually durable
D. Batch, Availability, Sharding, Elasticity

1504. How is data organised in a column-family (wide-column) store? Understanding

A. Every table column is stored in its own file and every row must have a value for every column
B. Data is stored only as key-value pairs with no grouping of columns
C. Rows are identified by a row key, columns are grouped into column families, and each row stores only the columns it actually has
D. Each record is a JSON document inside a collection

1505. Which statement about the CAP theorem is accurate? Understanding

A. A system can provide consistency, availability and partition tolerance together if its replicas are fast enough
B. When a network partition occurs, a distributed system must choose between consistency and availability
C. Scaling out horizontally requires giving up partition tolerance
D. The C in CAP means the same thing as the C in ACID

1506. What does Consistency mean in the CAP theorem? Understanding

A. Every transaction preserves the database's integrity constraints
B. Every request receives a non-error response
C. Every read returns the most recent write, as if there were a single copy of the data
D. The system keeps working when messages between nodes are lost

1507. What is horizontal scaling (scaling out)? Understanding

A. Moving the database to a single server with more CPU and RAM
B. Adding more columns to the largest table
C. Adding more servers and spreading the data and load across them
D. Splitting one table into two tables to remove redundancy

1508. How does sharding differ from replication? Understanding

A. Sharding puts different subsets of the data on different nodes; replication keeps copies of the same data on several nodes
B. Sharding copies every record to every node; replication splits the data into subsets
C. Both terms mean the same thing in NoSQL systems
D. Sharding is used only by graph databases; replication only by key-value stores

1509. A document database is often called 'schemaless'. Which statement is most accurate? Understanding

A. The database cannot store any structure, so every value is a plain string
B. Each document must declare its full schema before it can be inserted
C. Documents in one collection can have different fields, but the application still relies on an implicit schema when it reads them
D. Schemaless means the data cannot be indexed

1510. Sadalage and Fowler describe key-value, document and column-family stores as aggregate-oriented. Which NoSQL model is NOT aggregate-oriented? Understanding

A. Document store
B. Graph database
C. Key-value store
D. Column-family store

1511. What does eventual consistency guarantee? Understanding

A. Every read immediately returns the latest write
B. Replicas never disagree, even during a network partition
C. A write is rejected unless every replica acknowledges it
D. If no new updates are made, all replicas will converge to the same value after some time

1512. In the property graph model, where can key-value properties be attached? Understanding

A. Only on nodes
B. On both nodes and relationships
C. Only on relationships
D. Only on a separate schema table

1513. Which statement correctly separates a column-family store from a column-oriented (columnar) relational warehouse? Understanding

A. A columnar warehouse stores each column's values together to speed up scans and aggregates; a column-family store groups columns into families under a row key to suit sparse, key-based access
B. They are the same thing: both store a table one column per file
C. A column-family store requires every row to have the same columns, but a columnar warehouse does not
D. A columnar warehouse is a NoSQL model, while a column-family store is a relational engine

1514. What is NewSQL? Understanding

A. A query language that replaces SQL in document stores
B. Databases that keep SQL and ACID transactions while scaling out across many nodes
C. Key-value stores that drop transactions to gain speed
D. The newest version of the SQL standard

1515. An app keeps orders in a relational database, sessions in a key-value store and product reviews in a document store. What is this approach called? Understanding

A. Polyglot persistence
B. Sharding
C. Denormalisation
D. Eventual consistency

1516. A web app stores each user's shopping cart and only ever reads or writes the whole cart using the session ID. Which store fits best? Application

A. Graph database
B. Column-oriented data warehouse
C. Relational database with the cart split across five normalised tables
D. Key-value store

1517. A social app wants to suggest 'people you may know' based on mutual friends up to three hops away. Which store fits best? Application

A. Graph database
B. Key-value store
C. Column-oriented data warehouse
D. Document store with one document per user and no links

1518. An online store sells laptops (RAM, CPU, screen size) and shirts (size, colour, fabric). Each category has different attributes, and pages show one product at a time. Which store fits best? Application

A. Key-value store, because products never need to be filtered by any field
B. Graph database, because products have attributes
C. Document store, with one document per product holding only its own attributes
D. A single relational table with a column for every attribute of every category

1519. Thousands of sensors each send a reading every second. Queries ask for one sensor's readings over a time range. Which design fits best? Application

A. Document store that appends every reading to one ever-growing document per sensor
B. Column-family store with the sensor ID in the row key and readings ordered by timestamp
C. Graph database with a node for every reading
D. A single relational server with no partitioning

1520. Hash ring positions 0-99, clockwise. Nodes: A at 10, B at 40, C at 75. A key is stored on the first node at or after its hash, wrapping past 99 to the start. Which node stores a key whose hash is 80? Application

A. C
B. B
C. No node; the key is rejected
D. A

1521. Hash ring positions 0-99, clockwise. Nodes: A at 10, B at 40, C at 75. A key is stored on the first node at or after its hash, wrapping past 99 to the start. Keys have hashes 5, 12, 40 and 60. Which keys are stored on B? Application

A. 40 and 60
B. 12 and 40
C. 12 only
D. 5 and 12

1522. Hash ring positions 0-99, clockwise. Nodes: A at 10, B at 40, C at 75. A key is stored on the first node at or after its hash, wrapping past 99 to the start. Keys have hashes 12, 50, 60 and 80. A new node D is added at position 55. Which keys change owner? Application

A. Keys 50 and 60, which both move from C to D
B. Every key is reassigned, because the number of nodes changed
C. Only key 50, which moves from C to D
D. Only key 50, which moves from B to D

1523. A replicated store has N = 5 replicas. Which read quorum R and write quorum W guarantee that every read overlaps the latest successful write? Application

A. R = 3, W = 3
B. R = 2, W = 3
C. R = 3, W = 2
D. R = 1, W = 4

1524. A document store holds 3 customers who have placed 7 orders in total. If each customer's orders are embedded inside that customer's document, how many documents are in the customers collection? Application

A. 7
B. 3
C. 10
D. 21

1525. A column-family store holds 4 rows. Row u1 has name and city; u2 has name, email and phone; u3 has name; u4 has name, city and email. How many column values (cells) are actually stored? Application

A. 16
B. 4
C. 9
D. 12

1526. A graph has these undirected FRIENDS edges: Ali-Sara, Ali-Omar, Sara-Hina, Omar-Hina, Omar-Bilal, Hina-Zain, Bilal-Ali. Which people are friends of Ali's friends but are not Ali and not already Ali's friends? Application

A. Hina and Zain
B. Hina and Bilal
C. Hina, Bilal and Omar
D. Hina only

1527. A key-value store has 3 nodes numbered 0, 1 and 2, and places each key on node (user_id mod 3). On which node is user_id 11 stored? Application

A. Node 1
B. Node 2
C. Node 0
D. Node 3

1528. Most queries on an orders store ask for all orders between two dates. Which partitioning scheme serves these queries best, and why? Application

A. Hash partitioning on order date, because hashing keeps neighbouring dates together
B. Range partitioning on order date, because neighbouring dates sit on the same node and a range query touches few nodes
C. Hash partitioning on order ID, because every range query then hits exactly one node
D. Random placement, because it balances load perfectly

1529. A replicated store has N = 5 and writes complete after only W = 1 acknowledgement. What is the smallest read quorum R that still guarantees reading the latest write? Application

A. 3
B. 4
C. 1
D. 5

1530. Keys 0 to 99 are placed with server = key mod 4. A fifth server is added and placement becomes key mod 5. How many of the 100 keys now map to a different server? Analysis

A. 80
B. 20
C. 25
D. 0

1531. During a network partition, wallet-service nodes on the minority side refuse withdrawals until the partition heals. Which trade-off has the system made? Analysis

A. It favours availability over consistency (AP)
B. It provides consistency and availability by dropping partition tolerance (CA)
C. It favours consistency over availability (CP)
D. It has avoided the CAP trade-off by using a quorum

1532. A shopping-cart service keeps accepting 'add item' requests on both sides of a network partition. What follows from this design when the partition heals? Analysis

A. Replicas are guaranteed to be identical because every write was accepted
B. The system must roll back every write made during the partition
C. Replicas may hold different carts and must be reconciled, for example by merging the items
D. Nothing, because accepting writes on both sides makes the system CP

1533. A store uses N = 3, W = 1 and R = 1. A user updates their profile and immediately sees the old version on reload. What is the most effective fix? Analysis

A. Increase to N = 5 while keeping R = 1 and W = 1
B. Lower W to 0 so writes return faster
C. Switch from hash partitioning to range partitioning
D. Raise the quorums so that R + W > N, for example R = 2 and W = 2

1534. A banking module needs multi-row ACID transactions between accounts, ad hoc reporting with joins, and its data fits comfortably on one large server. Which choice is most defensible? Analysis

A. A relational database
B. A key-value store
C. A graph database
D. A column-family store tuned for writes

1535. In a document store, blog posts can collect tens of thousands of comments, and the site shows comments 20 at a time. How should comments be modelled? Analysis

A. As separate comment documents that reference the post's ID
B. Embedded as an array inside each post document
C. Embedded inside each user's document only
D. As one key-value entry holding all comments for the whole site

1536. Sensor writes are range-partitioned on timestamp. One node is overloaded while the others sit idle. What is the cause and the best fix? Analysis

A. The cluster has too few replicas; add replicas to the busy node
B. Timestamps are not unique; switch to a key-value store
C. The idle nodes have failed; restart them
D. New timestamps always fall into the newest range; partition on a hash of the sensor ID, or a hashed prefix plus time, to spread writes

1537. A relational query that finds everyone within 4 friendship hops of a user is very slow. Why does a graph database usually handle this better? Analysis

A. Graph databases store all data in RAM, while relational databases never cache
B. Each extra hop in SQL adds another self-join on the friendship table, while a graph database follows stored edges from node to node
C. SQL cannot express a 4-hop query at all
D. Graph databases need no indexes or storage at all

1538. For which feature is eventual consistency least acceptable? Analysis

A. Showing the number of likes on a post
B. Showing a friend's 'last seen' time
C. Counting product page views for a daily report
D. Preventing the last seat on a flight from being sold twice

1539. Hash ring positions 0-99, clockwise. Nodes: A at 10, B at 40, C at 75. A key is stored on the first node at or after its hash, wrapping past 99 to the start. Keys have hashes 5, 12, 40, 60 and 80. Node B fails and is removed from the ring. Which keys need a new owner, and which node takes them? Analysis

A. Keys 12 and 40, both moving to C
B. Key 12 moves to A and key 40 moves to C
C. Every key is reassigned across A and C
D. Keys 12 and 40, both moving to A

1540. MongoDB belongs to which NoSQL family? Understanding

A. Document store
B. Key-value store
C. Column-family store
D. Graph database

1541. Redis is most often described as which kind of database? Understanding

A. A graph database
B. An in-memory key-value store
C. A column-family store
D. A relational database

1542. Apache Cassandra and HBase belong to which NoSQL family? Understanding

A. Document stores
B. Graph databases
C. Column-family (wide-column) stores
D. In-memory key-value caches

1543. Cypher is the query language of which database? Understanding

A. MongoDB
B. Redis
C. Cassandra
D. Neo4j

1544. In which format does MongoDB store documents internally? Understanding

A. BSON, a binary encoding of JSON-like documents
B. CSV text files
C. XML
D. Fixed-length relational rows

1545. In BASE, what does 'soft state' mean? Understanding

A. Data is stored only in memory and lost on restart
B. The state of the system may change over time, even without new input, as replicas converge
C. The schema can change at any time
D. Transactions can be rolled back at any point

1546. What is a shard key (partition key)? Understanding

A. The password that unlocks a shard
B. A key that encrypts each partition
C. The field whose value decides which shard or node stores a record
D. The primary key of the system catalogue

1547. Why do NoSQL designs often denormalise data, for example by embedding related data in one document? Understanding

A. Because NoSQL stores cannot hold numbers
B. To make updates to shared data cheaper
C. Because normalisation is impossible without SQL
D. So that a common query can read one aggregate without joins across nodes

1548. What is vertical scaling (scaling up)? Understanding

A. Giving one server more CPU, memory or faster disks
B. Adding more servers and spreading data across them
C. Splitting a table into column groups
D. Copying data to servers in other regions

1549. In MongoDB, what is the closest equivalent of a relational table? Understanding

A. A document
B. A collection
C. A field
D. A database

1550. A key-value cache stores each entry with a time-to-live (TTL) of 300 seconds. What does the TTL do? Understanding

A. The entry can be read at most 300 times
B. The entry is copied to 300 nodes
C. The entry expires automatically 300 seconds after it is set
D. The entry is locked for 300 seconds after each write

1551. In a MongoDB replica set, where do writes go by default? Understanding

A. To any secondary, which forwards them to the primary
B. To every member at the same time
C. To the member with the fewest documents
D. To the primary, which replicates them to the secondaries

1552. What are vector clocks used for in Dynamo-style stores? Understanding

A. To tell whether two versions of a value are causally ordered or were written concurrently
B. To keep the wall-clock time the same on every node
C. To decide which node owns a key
D. To compress values before they are stored

1553. In a Dynamo-style store, what is hinted handoff? Understanding

A. A client hints which node it prefers to read from
B. When a replica node is down, another node accepts its write with a hint and passes it on when the node returns
C. A node hands its whole key range to a new node when it joins
D. A write is rejected and the client is told to retry

1554. Why do Cassandra-style clusters use a gossip protocol? Understanding

A. To encrypt messages between nodes
B. To elect a single master that handles all writes
C. So that each node periodically swaps state with a few others, and membership and failure news spread without a central master
D. To compress SSTables during compaction

1555. Why does consistent hashing often give each physical node many virtual nodes on the ring? Understanding

A. To store more copies of each key
B. To let keys skip the ring and go to any node
C. To avoid hashing keys at all
D. To spread load more evenly and to share a leaving node's keys among many nodes

1556. Why are writes fast in log-structured (LSM-tree) column-family stores such as Cassandra? Understanding

A. A write is appended to a commit log and an in-memory memtable, and later flushed as an immutable sorted file
B. Every write updates the row in place on disk immediately
C. Writes skip the disk entirely and are never persisted
D. Each write locks the whole table, so there is no contention

1557. What does a secondary index on the city field give a document store? Understanding

A. A second copy of every document
B. Fast lookup of documents by city without scanning the whole collection
C. A guarantee that city values are unique
D. A way to join two collections automatically

1558. What does index-free adjacency mean in a native graph database? Understanding

A. The database has no indexes of any kind
B. Nodes are stored in alphabetical order
C. Each node stores direct references to its neighbouring relationships, so a traversal follows pointers instead of index lookups
D. Relationships are stored in a separate join table

1559. Cassandra lets each query choose a consistency level such as ONE, QUORUM or ALL. What is this feature called? Understanding

A. Strong serialisability
B. Snapshot isolation
C. Two-phase commit
D. Tunable consistency

1560. What is MongoDB's aggregation pipeline? Understanding

A. A sequence of stages, such as $match, $group and $sort, that transform documents step by step
B. The process that copies writes from the primary to secondaries
C. A tool that converts SQL tables into collections
D. The queue in which shard balancing tasks wait

1561. Which consistency guarantee means a user always sees their own earlier writes, even if other users may see stale data? Understanding

A. Linearisability
B. Read-your-writes consistency
C. Eventual consistency with no session guarantees
D. Monotonic writes

1562. In Cassandra, what happens when a row is deleted? Understanding

A. The row is erased from every SSTable at once
B. The delete is ignored until a node restarts
C. A tombstone marker is written, and the data is removed later during compaction after a grace period
D. The row is moved to a recycle-bin table

1563. Column-family stores keep a Bloom filter for each SSTable. What can the filter tell a read? Understanding

A. The exact disk offset of every key
B. That a key is definitely in the SSTable, with no false positives
C. The latest value of the key without reading the file
D. That a key is definitely not in the SSTable, or that it might be, with some false positives

1564. A game needs a live leaderboard that updates scores thousands of times a second and shows the top 10 instantly. Which store fits best? Application

A. A graph database
B. A document store with one document per game
C. A relational data warehouse refreshed nightly
D. An in-memory key-value store with a sorted-set type, such as Redis

1565. A transport app must find the route between two metro stations with the fewest line changes. Which store fits best? Application

A. A graph database
B. A key-value store
C. A column-family store
D. A document store

1566. A web app wants to cache the results of slow database queries for 5 minutes, looked up by the query text. Which store fits best? Application

A. A graph database
B. A key-value store with TTL expiry
C. A column-family store
D. A document store with secondary indexes

1567. Hash ring positions 0-63, clockwise. Nodes: X at 8, Y at 24, Z at 50. A key is stored on the first node at or after its hash, wrapping past 63 to the start. Which node stores key 30? Application

A. Y
B. X
C. Z
D. No node; the key is lost

1568. A document store holds 5 authors and 12 books. Books are kept in their own collection and reference the author by ID. How many documents are stored in total? Application

A. 5
B. 12
C. 60
D. 17

1569. A cluster of 6 nodes stores 1,200 GB of unique data with replication factor 3, spread evenly. How much is stored in total, and how much per node? Application

A. 3,600 GB in total, about 600 GB per node
B. 1,200 GB in total, about 200 GB per node
C. 3,600 GB in total, about 1,200 GB per node
D. 7,200 GB in total, about 1,200 GB per node

1570. Ring positions 0-99, clockwise. Physical node A has virtual nodes at 20 and 70; B has virtual nodes at 45 and 95. Each key goes to the first virtual node at or after its hash, wrapping past 99. Which physical nodes own keys 50, 80 and 97? Application

A. B, B, A
B. A, B, A
C. A, B, B
D. B, A, A

1571. Ring positions 0-99, clockwise. Nodes: P at 5, Q at 30, R at 70. A key goes to the first node at or after its hash, wrapping past 99. If keys are spread evenly over 0-99, which node owns the largest share? Application

A. P, with 35% of the positions
B. Q, with 25% of the positions
C. R, with 40% of the positions
D. All three own an equal third

1572. Ring positions 0-99 with nodes A at 10, B at 30, C at 55 and D at 80. Each key is stored on its owner (first node at or after its hash) and the next 2 nodes clockwise. Which nodes store key 60? Application

A. C, D and A
B. D, A and C
C. B, C and D
D. D, A and B

1573. A store keeps N = 5 replicas and writes wait for W = 4 acknowledgements. What is the smallest R that guarantees fresh reads, and how many replicas can be down while writes still succeed? Application

A. R = 2; writes tolerate 1 down replica
B. R = 1; writes tolerate 1 down replica
C. R = 2; writes tolerate 2 down replicas
D. R = 3; writes tolerate 4 down replicas

1574. Ring positions 0-99. Physical node A has virtual nodes at 10 and 60, B is at 35, and C is at 85. Keys are 5, 30, 50 and 80. Node A leaves the cluster. Which keys move, and where? Application

A. Keys 5 and 50 both move to B
B. Key 5 moves to B and key 50 moves to C
C. Keys 5 and 50 both move to C
D. All four keys are reassigned

1575. A graph has directed FOLLOWS edges A→B, A→C, B→D, C→D, C→E, D→F and E→A. Which users are exactly two hops from A and not followed directly by A? Application

A. D, E and F
B. B and C
C. D and E
D. D only

1576. A users collection holds the documents shown. How many documents does the query return? Application

```
// documents
{ name: "Ali",  age: 30, city: "Lahore" }
{ name: "Sara", age: 22, city: "Lahore" }
{ name: "Omar", age: 40, city: "Karachi" }
{ name: "Hina", age: 27, city: "Lahore" }
{ name: "Zain",          city: "Lahore" }

db.users.find({ city: "Lahore", age: { $gt: 25 } })
```

A. 3
B. 4
C. 1
D. 2

1577. A write-heavy app shards its users collection. User IDs are random, and almost every query looks up one user by ID. Which shard key spreads writes most evenly while keeping lookups targeted? Application

A. A hashed shard key on user_id
B. A ranged shard key on account creation time
C. A ranged shard key on country
D. No shard key; let every shard store every user

1578. What does GET x return after these Redis commands? Application

```
SET x 5
INCR x
INCRBY x 3
DECR x
GET x
```

A. 9
B. 8
C. 6
D. 5

1579. Given this Cassandra table, which query can be served efficiently without scanning every partition? Application

```
CREATE TABLE readings (
  sensor_id text,
  ts        timestamp,
  value     double,
  PRIMARY KEY ((sensor_id), ts)
);
```

A. SELECT * FROM readings WHERE ts > '2026-01-01';
B. SELECT * FROM readings WHERE value > 40;
C. SELECT * FROM readings WHERE sensor_id = 's7' AND ts > '2026-01-01';
D. SELECT * FROM readings WHERE ts = '2026-01-01 10:00';

1580. A customer's name is copied into each of their 50 order documents, as well as into their own customer document. The customer changes their name. How many documents must be updated to keep every copy current? Application

A. 1
B. 50
C. 2
D. 51

1581. Three replicas return values for the same key with these write timestamps: 'Lahore' at 105, 'Karachi' at 110 and 'Multan' at 98. The store resolves conflicts with last-writer-wins. Which value is kept? Application

A. Karachi
B. Lahore
C. Multan
D. All three are kept as siblings

1582. Four versions carry vector clocks over nodes (A, B, C): V1 = [2, 1, 0], V2 = [1, 2, 0], V3 = [2, 2, 0], V4 = [3, 2, 1]. Which pair was written concurrently? Application

A. V1 and V3
B. V1 and V2
C. V2 and V4
D. V3 and V4

1583. Ring positions 0-99 with nodes A at 10, B at 40 and C at 75. Each key is stored on its owner and the next node clockwise (2 replicas). Node C is down, and reads use consistency ALL. Which of the keys 20, 50 and 80 can still be read? Application

A. Only key 20
B. Keys 50 and 80
C. Only key 80
D. All three

1584. A Cassandra keyspace has replication factor 3. Two of the three replicas for a partition are down. Which operations on that partition succeed? Application

A. Reads and writes at QUORUM succeed; ONE fails
B. All operations fail until both nodes return
C. Only reads at ALL succeed
D. Reads and writes at ONE succeed; QUORUM reads and writes fail

1585. In a document store, students take many courses and each course has many students. Course details such as the title and the instructor change each term. How should enrolments be modelled? Analysis

A. Keep students and courses as separate documents and store course IDs in each student document
B. Embed full course details in every student document
C. Embed every student in the course document and every course in the student document
D. Put all students and courses in one document

1586. Management wants monthly sales totals per region over five years of orders. Which store suits this query least well? Analysis

A. A columnar relational warehouse
B. A graph database
C. A relational database with GROUP BY
D. A column-family store with a table designed for this query

1587. Sessions are stored as opaque blobs in a key-value store, keyed by session ID. A new feature needs 'all active sessions for users in Lahore'. Why is this hard, and what is a sensible change? Analysis

A. Key-value stores cannot hold more than one session
B. The query fails because key-value stores cannot be sharded
C. The store cannot query inside values, so it would scan every key; store sessions in a document store with an index on city
D. Key-value stores lose data on restart

1588. In a 'schemaless' collection, some documents use email and others use emailAddress. A report counting users with an email finds too few. What is the root cause and the best fix? Analysis

A. The store lost documents; restore from backup
B. Document stores cannot count documents
C. The index on email is corrupt; rebuild it
D. Schema-on-read let inconsistent field names in; add schema validation and migrate old documents

1589. A payments company needs SQL, multi-row ACID transactions and automatic horizontal scaling across regions. Which choice fits best? Analysis

A. A NewSQL database
B. A key-value cache
C. An eventually consistent document store
D. A single-server relational database

1590. In a social app, a celebrity's posts are stored under partition key user_id. Her partition takes most of the cluster's write and read load. What is the cause and a standard fix? Analysis

A. The ring has too many virtual nodes; remove some
B. One hot key maps to one partition; split it by adding a bucket suffix such as user_id:bucket
C. The replication factor is too low; raise it to 10
D. Writes are too small; batch them across users

1591. Replica X holds a cart with vector clock [A:2, B:1], and replica Y holds a cart with [A:2, B:2]. Read repair runs. What should it do? Analysis

A. Keep both versions as conflicting siblings
B. Keep X's version, because node A has the same count
C. Keep Y's version and overwrite X, because X's clock is an ancestor of Y's
D. Merge the carts by taking the union of their items

1592. A Dynamo-style store uses N = 3, R = 2, W = 2 with sloppy quorums. During a partition, writes go to fallback nodes outside the key's usual replica set. After the partition heals, a read returns stale data. Why? Analysis

A. R + W = 4 is not greater than N = 3
B. Vector clocks are disabled whenever R = 2
C. The read was served from a cache with a long TTL
D. Sloppy quorums count fallback nodes, so the write and read sets need not overlap until hinted handoff finishes

1593. A Cassandra table has PRIMARY KEY (user_id, ts). A new dashboard needs the latest 10 events across all users, and the query is slow. What is the best fix? Analysis

A. Create a second table partitioned for this query, such as by day with ts as the clustering column, and write each event to both
B. Add ALLOW FILTERING to the query
C. Increase the replication factor
D. Switch the consistency level to ONE

1594. A cache tier grows from 4 to 5 servers. Using consistent hashing with many virtual nodes per server, about what fraction of keys moves? Analysis

A. About 80%, as with key mod N
B. About 20%, the share taken by the new server
C. About 50%
D. None, because consistent hashing never moves keys

1595. Two replicas each read a like-counter of 10 during a partition, add 1 locally and write 11. Conflicts are resolved by last-writer-wins. What is the final value, and what design avoids the problem? Analysis

A. 12, because both increments are applied
B. 10, because conflicting writes cancel out
C. 11, losing one like; use a counter CRDT or a server-side atomic increment
D. 11, which is correct because only one like happened

1596. A MongoDB app writes with write concern w: 1. The primary acknowledges a payment, then fails before any secondary copies it. A secondary becomes primary. What happens to the payment, and what prevents this? Analysis

A. It is safe, because w: 1 means it is on every member
B. The election waits until the old primary returns
C. The payment is duplicated on both primaries
D. It is rolled back and lost from the new primary; use write concern 'majority'

1597. A store uses N = 3, writes at ONE (W = 1) and reads at ALL (R = 3). Which statement best describes this setting? Analysis

A. Reads are always fresh, but any single replica outage makes reads fail
B. Reads can be stale, because W = 1
C. Writes fail whenever one replica is down
D. It is the most available setting for both reads and writes

1598. An e-commerce document store embeds a copy of the product name and price in each order line. Later, a product's price changes, and old orders still show the old price. Is this a bug? Analysis

A. Yes: all embedded copies should be updated whenever the product changes
B. No: the order should keep the price at the time of sale, so this copy is a correct snapshot
C. Yes: orders should only reference the product ID and read the current price
D. No: document stores cannot update embedded fields

1599. An eventually consistent store sells concert tickets. Under load, more tickets are sold than exist. Which fix keeps the store but stops overselling? Analysis

A. Raise the replication factor
B. Lower the read consistency to ONE to speed up checks
C. Use a conditional (compare-and-set) or lightweight-transaction write for the stock decrement only
D. Add a TTL to every ticket record

Data Warehousing & Data Mining (Introductory) ...

1600. Which workload is typical of OLAP rather than OLTP? Understanding

A. Many short transactions that insert or update a few rows each
B. Enforcing row-level locks during an ATM withdrawal
C. Complex, read-mostly queries that summarise large volumes of historical data
D. Keeping the current balance of one account up to date

1601. Inmon defines a data warehouse as subject-oriented, integrated, non-volatile and which fourth property? Understanding

A. Time-variant
B. Normalised to BCNF
C. Real-time updated
D. Distributed

1602. In ETL, what is the usual order of the three stages? Understanding

A. Extract, Load, Transform only
B. Transform, Extract, Load
C. Load, Extract, Transform
D. Extract, Transform, Load

1603. What is the defining shape of a star schema? Understanding

A. Dimension tables normalised into several levels of sub-tables
B. One central fact table joined directly to denormalised dimension tables
C. Several fact tables with no dimension tables
D. A single wide table with no keys

1604. How does a snowflake schema differ from a star schema? Understanding

A. It has no fact table
B. Its dimension tables are normalised into further related tables
C. Its fact table is split into one table per measure
D. It stores dimensions inside the fact table

1605. A warehouse has a sales fact table and a shipping fact table that share the date, product and store dimensions. What is this schema called? Understanding

A. Snowflake schema
B. Fact constellation (galaxy schema)
C. Star schema with a degenerate dimension
D. Data lake

1606. What does a fact table mainly contain? Understanding

A. Numeric measures plus foreign keys to the dimension tables
B. Descriptive text attributes such as customer name and city
C. Only the primary keys of the source systems
D. The ETL job schedule

1607. What is a data mart? Understanding

A. A smaller, subject-focused store serving one department, such as sales or finance
B. The staging area where raw data waits before transformation
C. A backup copy of the OLTP database
D. A tool that mines association rules

1608. Which OLAP operation moves from city-level sales to country-level sales? Understanding

A. Drill-down
B. Slice
C. Roll-up
D. Pivot

1609. An account_balance measure can be summed across customers on one date but not across dates. What kind of measure is it? Understanding

A. Fully additive
B. Semi-additive
C. Non-additive
D. Derived only

1610. In the knowledge discovery in databases (KDD) process described by Han and Kamber, where does the data mining step sit? Understanding

A. It is the first step, before any cleaning
B. It is the last step, after knowledge presentation
C. After data cleaning, integration, selection and transformation, and before pattern evaluation
D. It replaces the whole KDD process

1611. What is the key difference between classification and clustering? Understanding

A. Classification needs no training data; clustering does
B. Classification learns from records with known class labels; clustering groups records that have no labels
C. Clustering always produces exactly two groups
D. They are two names for the same technique

1612. In association rule mining, what does the support of an itemset measure? Understanding

A. The fraction of transactions with the left-hand side that also contain the right-hand side
B. The fraction of all transactions that contain the itemset
C. How much more likely items appear together than by chance
D. The number of distinct items in the itemset

1613. Why do data warehouses usually use surrogate keys in dimension tables? Understanding

A. They are warehouse-generated keys independent of source systems, so data from several sources can be integrated and history versions kept
B. They are copied directly from the OLTP primary keys to save storage
C. They make the fact table human-readable
D. They replace the need for a fact table

1614. A customer moves from Lahore to Karachi. Which slowly changing dimension type adds a new dimension row with a new surrogate key and validity dates, keeping the old row? Understanding

A. Type 1
B. Type 3
C. Type 2
D. Type 0

1615. How does ROLAP differ from MOLAP? Understanding

A. ROLAP stores data in arrays; MOLAP stores data in relational tables
B. ROLAP is used for OLTP and MOLAP for OLAP
C. MOLAP cannot support roll-up or drill-down
D. ROLAP runs OLAP queries over relational tables such as a star schema; MOLAP stores pre-computed aggregates in multidimensional arrays

1616. Using this fact table, roll up from (region, quarter) to region. What is South's total? Application

```
CREATE TABLE sales (region TEXT, quarter TEXT, product TEXT, amount INT);
INSERT INTO sales VALUES
  ('North','Q1','Pens',40), ('North','Q1','Books',60),
  ('North','Q2','Pens',30), ('South','Q1','Pens',50),
  ('South','Q2','Books',70), ('South','Q2','Pens',20),
  ('East','Q1','Books',25), ('East','Q2','Books',35);
```

A. 140
B. 90
C. 50
D. 330

1617. Using this fact table, slice on quarter = 'Q2' and then total by product. What is the result? Application

```
CREATE TABLE sales (region TEXT, quarter TEXT, product TEXT, amount INT);
INSERT INTO sales VALUES
  ('North','Q1','Pens',40), ('North','Q1','Books',60),
  ('North','Q2','Pens',30), ('South','Q1','Pens',50),
  ('South','Q2','Books',70), ('South','Q2','Pens',20),
  ('East','Q1','Books',25), ('East','Q2','Books',35);
```

A. Books 105, Pens 50
B. Books 190, Pens 140
C. Books 70, Pens 50
D. Books 35, Pens 30

1618. Using this fact table, dice the cube for region IN ('North', 'East') and quarter = 'Q1'. What is the total amount in the sub-cube? Application

```
CREATE TABLE sales (region TEXT, quarter TEXT, product TEXT, amount INT);
INSERT INTO sales VALUES
  ('North','Q1','Pens',40), ('North','Q1','Books',60),
  ('North','Q2','Pens',30), ('South','Q1','Pens',50),
  ('South','Q2','Books',70), ('South','Q2','Pens',20),
  ('East','Q1','Books',25), ('East','Q2','Books',35);
```

A. 100
B. 125
C. 175
D. 190

1619. An analyst is viewing yearly sales and asks to see the same figures broken down by quarter. Which OLAP operation is this? Application

A. Roll-up
B. Slice
C. Dice
D. Drill-down

1620. What does this star-schema query return? Application

```
CREATE TABLE dim_product (product_key INT, name TEXT, category TEXT);
CREATE TABLE dim_store (store_key INT, city TEXT);
CREATE TABLE fact_sales (product_key INT, store_key INT, units INT);
INSERT INTO dim_product VALUES (1,'Pen','Stationery'), (2,'Notebook','Stationery'),
(3,'Mouse','Electronics');
INSERT INTO dim_store VALUES (10,'Lahore'), (20,'Karachi');
INSERT INTO fact_sales VALUES (1,10,5), (2,10,3), (3,10,2), (1,20,4), (3,20,6), (3,10,1);
SELECT p.category, SUM(f.units) AS total
FROM fact_sales f
JOIN dim_product p ON f.product_key = p.product_key
JOIN dim_store s ON f.store_key = s.store_key
WHERE s.city = 'Lahore'
GROUP BY p.category
ORDER BY p.category;
```

A. Electronics 9, Stationery 12
B. Electronics 2, Stationery 8
C. Electronics 3, Stationery 8
D. Stationery 8 only

1621. A data cube has 3 dimensions and no concept hierarchies. How many cuboids does the full cube contain, counting the base and apex cuboids? Application

A. 6
B. 3
C. 9
D. 8

1622. A cube has 3 dimensions. Time has 3 hierarchy levels (day, month, year), location has 2 (city, country) and product has 1. How many cuboids are there, counting the 'all' level of each dimension? Application

A. 8
B. 6
C. 24
D. 12

1623. Given these 10 transactions, what is the support of {bread, milk}? Application

```
T1: bread, milk
T2: bread, butter, milk
T3: bread, butter
T4: milk, eggs
T5: bread, milk, eggs
T6: butter, milk
T7: bread, butter, milk, eggs
T8: bread
T9: milk
T10: bread, milk, butter
```

A. 0.7
B. 0.625
C. 0.8
D. 0.5

1624. Given these 10 transactions, what is the confidence of the rule butter → bread? Application

```
T1: bread, milk
T2: bread, butter, milk
T3: bread, butter
T4: milk, eggs
T5: bread, milk, eggs
T6: butter, milk
T7: bread, butter, milk, eggs
T8: bread
T9: milk
T10: bread, milk, butter
```

A. 0.8
B. 0.4
C. 0.571
D. 0.5

1625. Given these 10 transactions, what are the confidence and lift of eggs → milk? Application

```
T1: bread, milk
T2: bread, butter, milk
T3: bread, butter
T4: milk, eggs
T5: bread, milk, eggs
T6: butter, milk
T7: bread, butter, milk, eggs
T8: bread
T9: milk
T10: bread, milk, butter
```

A. Confidence 0.375, lift 1.25
B. Confidence 1.0, lift 0.8
C. Confidence 0.3, lift 1.0
D. Confidence 1.0, lift 1.25

1626. During preprocessing, income values range from 200 to 1,000. Using min-max normalisation to [0, 1], what does 500 become? Application

A. 0.375
B. 0.5
C. 0.625
D. 0.3

1627. Income has mean 54,000 and standard deviation 16,000. What is the z-score of 73,600? Application

A. 4.6
B. 0.817
C. 1.225
D. 19.6

1628. Sorted prices 4, 8, 15, 21, 21, 24, 25, 28, 34 are split into equal-frequency bins of 3. With smoothing by bin means, what value replaces 8? Application

A. 4
B. 8
C. 15
D. 9

1629. A bank's system debits an account at an ATM and must update the balance within the transaction. Which kind of system handles this? Application

A. OLAP
B. Data mart
C. OLTP
D. ETL

1630. For these transactions, the rule bread → milk has confidence 0.714. A manager wants to place milk next to bread because 'most bread buyers buy milk'. What does the lift show? Analysis

```
T1: bread, milk
T2: bread, butter, milk
T3: bread, butter
T4: milk, eggs
T5: bread, milk, eggs
T6: butter, milk
T7: bread, butter, milk, eggs
T8: bread
T9: milk
T10: bread, milk, butter
```

A. Lift is about 1.43, so bread strongly increases milk sales
B. Lift equals the confidence, 0.714, so the rule is moderately strong
C. Lift cannot be calculated without a minimum support threshold
D. Lift is about 0.89, so bread buyers are slightly less likely than average to buy milk and the rule is misleading

1631. With a minimum support count of 2, {butter, eggs} appears in only 1 transaction. Why can Apriori skip counting {bread, butter, eggs}? Analysis

```
T1: bread, milk
T2: bread, butter, milk
T3: bread, butter
T4: milk, eggs
T5: bread, milk, eggs
T6: butter, milk
T7: bread, butter, milk, eggs
T8: bread
T9: milk
T10: bread, milk, butter
```

A. Every superset of an infrequent itemset is also infrequent, so the candidate is pruned
B. Itemsets with three items are never frequent
C. Bread is the most common item, so it is excluded
D. Apriori counts only 2-itemsets

1632. Run Apriori on these transactions with a minimum support count of 4. Which 3-itemsets are frequent? Analysis

```
T1: bread, milk
T2: bread, butter, milk
T3: bread, butter
T4: milk, eggs
T5: bread, milk, eggs
T6: butter, milk
T7: bread, butter, milk, eggs
T8: bread
T9: milk
T10: bread, milk, butter
```

A. {bread, butter, milk}, with count 4
B. {bread, milk, eggs}, with count 5
C. No 3-itemset is frequent; the only candidate, {bread, butter, milk}, appears in 3 transactions
D. {bread, butter, milk} and {bread, milk, eggs}

1633. Analysts complain that dashboard queries on a snowflake schema are slow, and storage is not a concern. Which change is most likely to help? Analysis

A. Normalise the fact table into third normal form
B. Denormalise the dimensions into a star schema so each query needs fewer joins
C. Move the dashboards onto the OLTP database
D. Add more levels of sub-dimension tables

1634. A report sums each customer's month-end account balance over 12 months and shows a yearly 'total balance' twelve times larger than expected. What is the error? Analysis

A. Balance is semi-additive and must not be summed over time; use the last or average balance for the year
B. The fact table needs a surrogate key
C. The data should have been loaded with ELT instead of ETL
D. Balances are non-additive and cannot be summed across customers either

1635. Monthly management reports run directly on the order-entry database, and order processing slows badly while they run. What is the best long-term fix? Analysis

A. Run the reports more often so each one is smaller
B. Remove indexes from the order tables
C. Increase the isolation level of the reports to SERIALIZABLE
D. Copy the data through ETL into a separate data warehouse and run the reports there

1636. After two source systems were loaded into the warehouse, revenue per customer looks doubled for some customers, who appear twice under different IDs. Which ETL step was missing? Analysis

A. Extraction, because only one source should ever be read
B. Loading, because the warehouse should be reloaded twice
C. Indexing the fact table on the date key
D. Transformation that integrates and de-duplicates records, matching the same customer across sources

1637. A developer accidentally inserts store_key 10 (Lahore) twice into dim_store. What does this query now return? Analysis

```
CREATE TABLE dim_store (store_key INT, city TEXT);
CREATE TABLE fact_sales (product_key INT, store_key INT, units INT);
INSERT INTO dim_store VALUES (10,'Lahore'), (20,'Karachi'), (10,'Lahore');
INSERT INTO fact_sales VALUES (1,10,5), (2,10,3), (3,10,2), (1,20,4), (3,20,6), (3,10,1);
SELECT s.city, SUM(f.units) AS total
FROM fact_sales f
JOIN dim_store s ON f.store_key = s.store_key
WHERE s.city = 'Lahore'
GROUP BY s.city;
```

A. Lahore 11
B. Lahore 22
C. Lahore 12
D. Lahore 33

1638. After a customer moved from Lahore to Karachi, a report shows all of her past Lahore purchases under Karachi. What caused this, and what fixes it? Analysis

A. The fact table is semi-additive; use averages instead of sums
B. The customer dimension uses Type 1 (overwrite); switch to Type 2 so each address gets its own row
C. The cube was rolled up instead of drilled down
D. The ETL used Type 2; switch to Type 1 to keep history

1639. A telecom firm has no labels, but wants to find natural groups of customers with similar usage to design new packages. Which data mining task fits best? Analysis

A. Classification
B. Association rule mining
C. Clustering
D. Regression

1640. An analyst swaps the rows and columns of a sales report so that products run across the top and regions run down the side. Which OLAP operation is this? Understanding

A. Drill-across
B. Pivot (rotate)
C. Slice
D. Roll-up

1641. In a star schema, what does a dimension table such as dim_product mainly hold? Understanding

A. Only the foreign keys of the fact table
B. Descriptive attributes, such as product name, brand and category, used to filter and group the facts
C. Transaction logs copied unchanged from the source system
D. Numeric measures such as units sold and revenue

1642. In Kimball's dimensional modelling, what does the 'grain' of a fact table declare? Understanding

A. The compression level used to store the measures
B. How often the ETL job refreshes the warehouse
C. Exactly what one row of the fact table represents, such as one product on one receipt line
D. The number of dimension tables joined to the fact table

1643. Two data marts, Sales and Inventory, use the identical date and product dimension tables, with the same keys and attribute meanings. What are these shared dimensions called? Understanding

A. Role-playing dimensions
B. Conformed dimensions
C. Junk dimensions
D. Degenerate dimensions

1644. An invoice number is stored directly in the sales fact table as a key column, with no dimension table of its own because it has no other attributes. What kind of dimension is it? Understanding

A. A conformed dimension
B. A slowly changing dimension
C. A degenerate dimension
D. An outrigger dimension

1645. A university warehouse records which student attended which lecture on which date. The table has only foreign keys to student, lecture and date, and no numeric measure. What is it called? Understanding

A. A bridge-free snowflake
B. A factless fact table
C. A dimension table
D. An aggregate table

1646. How does Inmon's approach to building a warehouse differ from Kimball's? Understanding

A. Inmon builds a normalised enterprise warehouse first and derives data marts from it (top-down); Kimball builds dimensional marts on conformed dimensions that together form the warehouse (bottom-up)
B. Inmon's warehouse is volatile and updated in place; Kimball's is non-volatile
C. Inmon uses star schemas for everything; Kimball forbids star schemas
D. Inmon loads data without any transformation; Kimball transforms it

1647. What is the purpose of the staging area in an ETL architecture? Understanding

A. A backup copy of the operational database used for disaster recovery
B. The place where OLAP cubes are displayed to managers
C. A temporary workspace where extracted data is cleaned and transformed before it is loaded into the warehouse
D. The final presentation layer that business users query directly

1648. In a data warehouse, what is metadata? Understanding

A. Encrypted copies of the dimension tables
B. Data about the data, such as table definitions, source mappings, transformation rules and refresh times
C. Historical data older than five years
D. The largest fact table in the warehouse

1649. In a data cube lattice, what is the apex cuboid? Understanding

A. The base cuboid at the lowest level of every dimension
B. The cuboid with the most cells
C. Any cuboid that has been materialised
D. The 0-D cuboid holding a single grand total over all dimensions

1650. A warehouse pre-computes and stores only a chosen subset of cuboids, and computes the others on demand from those. What is this strategy called? Understanding

A. No materialisation
B. Partial materialisation
C. Full materialisation
D. Denormalisation

1651. What is HOLAP? Understanding

A. OLAP run only on a single desktop machine
B. Hierarchical OLAP that allows only roll-up and never drill-down
C. An OLAP server that stores everything in a NoSQL document store
D. A hybrid that keeps detailed data in relational tables (as ROLAP does) and aggregates in multidimensional storage (as MOLAP does)

1652. Why are bitmap indexes popular in data warehouses? Understanding

A. They are compact and fast for low-cardinality columns such as gender or region, and bit operations combine several conditions quickly
B. They make frequent single-row updates cheaper than a B+ tree
C. They remove the need for a fact table
D. They are the best index for columns where almost every value is unique

1653. Which slowly changing dimension type keeps the current value and the previous value of an attribute in two separate columns of the same row? Understanding

A. Type 3
B. Type 2
C. Type 0
D. Type 1

1654. A bank wants to flag card transactions that are very unlike a customer's normal behaviour. Which data mining task is this? Understanding

A. Outlier (anomaly) detection
B. Association rule mining
C. Data cube aggregation
D. Regression

1655. What is the difference between classification and regression? Understanding

A. Classification needs no training data; regression does
B. Regression is used only for time series; classification only for text
C. Classification predicts a categorical label; regression predicts a continuous numeric value
D. Classification groups unlabelled records; regression labels them

1656. What does the Apriori property state? Understanding

A. All non-empty subsets of a frequent itemset must also be frequent
B. An itemset is frequent only if its confidence exceeds its support
C. All supersets of a frequent itemset must also be frequent
D. Every 1-itemset in the data is frequent

1657. What is the main advantage of FP-growth over Apriori? Understanding

A. It finds clusters as well as itemsets
B. It works on unlabelled data while Apriori needs class labels
C. It produces association rules with higher confidence
D. It mines frequent itemsets without generating candidate itemsets, using a compressed FP-tree and usually only two database scans

1658. How does a closed frequent itemset differ from a maximal frequent itemset? Understanding

A. Closed itemsets meet minimum confidence; maximal itemsets meet minimum support
B. A closed itemset has no superset with the same support; a maximal itemset has no frequent superset at all
C. A closed itemset contains exactly two items; a maximal itemset contains the most items in the data
D. A closed itemset has no frequent superset; a maximal itemset has no superset with the same support

1659. Which statement about k-means clustering is correct? Understanding

A. The number of clusters k must be chosen in advance, and each cluster is represented by the mean of its points
B. It needs class labels for every training record
C. It represents each cluster by an actual data point called the medoid
D. It always finds the globally optimal clustering

1660. When ID3 builds a decision tree, how does it choose the attribute to split on at each node? Understanding

A. The attribute with the highest information gain
B. The attribute that appears first in the table
C. The attribute with the lowest support
D. The attribute with the most distinct values, always

1661. What is overfitting in classification? Understanding

A. The training set contains more records than the test set
B. The model is too simple to fit even the training data
C. The model learns noise and quirks of the training data, so it scores well on training data but poorly on unseen data
D. The model's accuracy is the same on training and test data

1662. Principal component analysis (PCA) applied during preprocessing is an example of which step? Understanding

A. Data integration
B. Pattern evaluation
C. Data reduction (dimensionality reduction)
D. Data cleaning

1663. A sales cube has 4 dimensions (time, item, branch, supplier) and no concept hierarchies. How many cuboids does the full cube contain, counting the base and apex cuboids? Application

A. 8
B. 16
C. 4
D. 24

1664. Time has 4 hierarchy levels (day, month, quarter, year), location has 3 (city, province, country) and product has 2 (item, brand). Using Han and Kamber's formula, how many cuboids are there, counting the 'all' level of each dimension? Application

A. 60
B. 36
C. 24
D. 8

1665. A cube has 4 dimensions and no hierarchies. How many of its cuboids are 2-D cuboids? Application

A. 4
B. 16
C. 6
D. 8

1666. For a rule A → B, confidence is 0.6, support of A is 0.5 and support of B is 0.4. What is the lift of the rule? Application

A. 0.3
B. 0.24
C. 1.2
D. 1.5

1667. This query turns quarters into columns. What row does it return for South? Application

```
-- sales(region, quarter, amount)
-- ('North','Q1',40) ('North','Q2',25) ('North','Q1',15)
-- ('South','Q1',30) ('South','Q2',45) ('South','Q2',10)
-- ('West','Q2',60)
SELECT region,
       SUM(CASE WHEN quarter = 'Q1' THEN amount ELSE 0 END) AS q1,
       SUM(CASE WHEN quarter = 'Q2' THEN amount ELSE 0 END) AS q2
FROM sales
GROUP BY region
ORDER BY region;
```

A. South, 85, 0
B. South, 30, 45
C. South, 55, 30
D. South, 30, 55

1668. Given this star schema, what does the query return? Application

```
CREATE TABLE dim_store (store_key INT PRIMARY KEY, city TEXT);
CREATE TABLE dim_date (date_key INT PRIMARY KEY, year INT, quarter TEXT);
CREATE TABLE fact_sales (date_key INT, store_key INT, units INT);
INSERT INTO dim_store VALUES (1,'Lahore'),(2,'Karachi'),(3,'Karachi');
INSERT INTO dim_date VALUES (101,2024,'Q4'),(201,2025,'Q1'),(202,2025,'Q2');
INSERT INTO fact_sales VALUES (101,1,10),(101,2,7),(201,2,5),(201,3,8),
  (202,1,4),(202,3,6),(202,2,9);

SELECT d.year, SUM(f.units) AS units
FROM fact_sales f
JOIN dim_date  d ON f.date_key  = d.date_key
JOIN dim_store s ON f.store_key = s.store_key
WHERE s.city = 'Karachi'
GROUP BY d.year
ORDER BY d.year;
```

A. 2024: 17, 2025: 32
B. 2024: 7, 2025: 14
C. 2025: 13
D. 2024: 7, 2025: 28

1669. Prices range from 200 to 1,000. Using min-max normalisation to the new range [-1, 1], what does 700 become? Application

A. 0.625
B. 0.25
C. 0.5
D. -0.25

1670. Values of an attribute range from -986 to 917. Using normalisation by decimal scaling, what does -452 become? Application

A. -4.52
B. -0.452
C. -0.0452
D. -0.493

1671. Sorted values 5, 10, 11, 13, 15, 35, 50, 55, 72 are split into equal-frequency bins of 3. With smoothing by bin boundaries, what value replaces 15? Application

A. 35
B. 21
C. 13
D. 15

1672. Ages from 10 to 90 are split into 4 equal-width bins. Which bin does the age 63 fall into? Application

A. The fourth bin, [70, 90]
B. The second bin, [30, 50)
C. The third bin, [50, 70)
D. The first bin, [10, 30)

1673. A training set has 14 records: 9 labelled 'yes' and 5 labelled 'no'. What is its entropy (information), to three decimal places? Application

A. 1.000
B. 0.940
C. 0.459
D. 0.643

1674. A node holds 9 'yes' and 5 'no' records. What is its Gini index, as used by CART? Application

A. 0.459
B. 0.357
C. 0.541
D. 0.940

1675. What is the Euclidean distance between the points (2, 3) and (5, 7)? Application

A. 5
B. 7
C. 12
D. 25

1676. What is the Manhattan distance between the points (1, 2, 3) and (4, 0, 5)? Application

A. 3
B. 4.12
C. 17
D. 7

1677. One-dimensional points 2, 3, 4, 10, 12, 20 are clustered with k-means, k = 2, starting centroids 3 and 12. After the first assignment and update step, what are the new centroids? Application

A. 3 and 12
B. 3 and 11
C. 4.75 and 16
D. 3 and 14

1678. In the first pass, Apriori finds 5 frequent 1-itemsets. How many candidate 2-itemsets does the join step produce? Application

A. 5
B. 25
C. 10
D. 20

1679. {bread, milk, eggs} is a frequent itemset. How many association rules with a non-empty left side and a non-empty right side can be formed from it? Application

A. 8
B. 9
C. 6
D. 3

1680. The frequent 2-itemsets are {A,B}, {A,C}, {B,C} and {B,D}. After Apriori's join and prune steps, which candidate 3-itemsets remain? Application

A. {A,B,C}, {A,B,D} and {B,C,D}
B. No candidates remain
C. {A,B,C} and {B,C,D}
D. Only {A,B,C}

1681. A classifier is tested on 100 records: TP = 40, FN = 10, FP = 5, TN = 45. What is its precision for the positive class? Application

A. 0.889
B. 0.900
C. 0.800
D. 0.850

1682. A manager viewing national sales asks to see only Lahore and Karachi, only the Books and Toys categories, and only 2025. Which OLAP operation produces this sub-cube? Application

A. Slice
B. Roll-up
C. Pivot
D. Dice

1683. An estate agency wants to estimate the selling price, in rupees, of each new house from its area, location and age. Which data mining task fits? Application

A. Clustering
B. Association rule mining
C. Regression
D. Outlier detection

1684. A fact table stores order-header rows (with order_total) and order-line rows (with line_amount) in the same table. A report sums both amount columns per month and shows revenue twice the real value. What is the design fault? Analysis

A. The report should use a snowflake schema instead
B. The date dimension needs a surrogate key
C. The ETL should have loaded the data with ELT instead
D. The fact table mixes two grains; each fact table should have one declared grain, such as one row per order line

1685. A supermarket mines its baskets with minimum support 5%. Expensive items such as imported cheese never appear in any rule, although the manager believes they sell in pairs with wine. What is the likely reason, and the fix? Analysis

A. Cheese and wine should be clustered with k-means instead
B. The confidence threshold is too low
C. Apriori cannot handle items that cost more than average
D. Rare but valuable items fall below a single high support threshold; use a lower or item-specific (multiple) minimum support

1686. Fraud is 1% of 10,000 transactions. A model labels every transaction 'not fraud' and reports 99% accuracy. What is the right assessment? Analysis

A. Precision for fraud is 99%, so the model is acceptable
B. 99% accuracy proves the model is excellent
C. The model overfits because its accuracy is above 95%
D. The model is useless: its recall for fraud is 0, and accuracy misleads on imbalanced classes

1687. A decision tree scores 100% accuracy on its training data but only 62% on a held-out test set. What should the analyst do first? Analysis

A. Prune the tree (or limit its depth) because it is overfitting the training data
B. Switch to clustering because the data has labels
C. Train and test on the same data to get a fair estimate
D. Add more branches so it fits the test data too

1688. Customers are clustered with k-means on age (20 to 70) and annual income (200,000 to 5,000,000 rupees). The clusters split almost purely by income. Why, and what fixes it? Analysis

A. k-means always ignores the first attribute
B. Income's much larger range dominates the Euclidean distance; normalise both attributes first (for example min-max or z-score)
C. k is too large; use k = 1
D. Age is categorical, so it must be removed

1689. A k-means run on store sales places one centroid far from every store because a single head-office record has sales 50 times higher than the rest. Which change best addresses this? Analysis

A. Increase the number of iterations of k-means
B. Use k-medoids (such as PAM), which represents clusters by actual objects and is less sensitive to outliers
C. Apply association rule mining instead
D. Use the mean of all records as every starting centroid

1690. Store A has sales of 1,000 with a 10% profit margin; store B has sales of 100 with a 30% margin. A dashboard shows the combined margin as 20%. What is the correct combined margin, and why was 20% wrong? Analysis

A. About 11.8%; margin is non-additive, so it must be recomputed as total profit / total sales (130 / 1,100)
B. 20%; averaging the two margins is correct
C. 30%; the higher margin always applies
D. 40%; margins should be summed across stores

1691. After one reorganisation of sales territories, managers want every report to show each customer's current territory and the territory just before it, side by side. Full history is not needed. Which SCD type fits best? Analysis

    A. Type 3
    B. Type 0
    C. Type 2
    D. Type 1

1692. A nightly ETL job reloads the entire 2-billion-row orders table, although only about 1% of rows change each day, and it now runs past the morning deadline. What is the best fix? Analysis

    A. Remove the transformation step to save time
    B. Switch to an incremental load that extracts only new or changed rows (change data capture)
    C. Move the reports back onto the OLTP system
    D. Run the full reload twice a night

1693. A telecom firm must run ad hoc OLAP queries over 20 TB of detailed call records, with new dimensions added often. Which OLAP storage approach suits this best? Analysis

    A. ROLAP, because relational storage scales to very large detailed data and flexible queries
    B. No OLAP; use the OLTP system directly
    C. A spreadsheet pivot table on one analyst's laptop
    D. MOLAP, because dense pre-computed arrays scale best to 20 TB of detail

1694. Finance and Marketing built separate data marts. Each defines 'active customer' differently and uses its own customer keys, so their customer counts never agree. What is the best architectural fix? Analysis

    A. Build a third independent mart for the CEO
    B. Adopt conformed dimensions (a shared customer dimension and agreed definitions) across marts, as in Kimball's bus architecture
    C. Switch both marts from star to snowflake schemas
    D. Delete the older mart's data each month

1695. A cube has 10 dimensions, each with 4 hierarchy levels. An architect proposes materialising every cuboid. Why is this a poor plan? Analysis

    A. Cuboids cannot be materialised when dimensions have hierarchies
    B. There are 2^10 = 1,024 cuboids, which is trivially small to store
    C. There are only 40 cuboids, so materialising them gives no speed-up
    D. There are 5^10, about 9.8 million, cuboids, so full materialisation is impractical; partial materialisation of useful cuboids is needed

1696. SELECT SUM(amount) FROM fact_sales returns 100, but this category report adds up to only 85. What explains the gap, and what is the usual warehouse fix? Analysis

```
CREATE TABLE dim_product (product_key INT PRIMARY KEY, category TEXT);
CREATE TABLE fact_sales (product_key INT, amount INT);
INSERT INTO dim_product VALUES (1,'Books'),(2,'Toys');
INSERT INTO fact_sales VALUES (1,30),(2,25),(1,20),(9,15),(2,10);

SELECT p.category, SUM(f.amount)
FROM fact_sales f
JOIN dim_product p ON f.product_key = p.product_key
GROUP BY p.category;
```

A. SUM ignores the last row of each group
B. GROUP BY removes duplicate amounts such as the two 25s
C. A fact row with product_key 9 has no matching dimension row, so the inner join drops it; ETL should map unknown keys to an 'Unknown' dimension member
D. The primary key on dim_product deletes fact rows with repeated keys

1697. A strong rule diapers → beer (lift 2.1) is found. A manager concludes that buying diapers causes people to want beer and plans an advert on that basis. What is the sound interpretation? Analysis

A. Confidence, not lift, would prove causation
B. Lift above 1 proves diapers cause beer purchases
C. The rule is wrong because diapers and beer are in different categories
D. The rule shows co-occurrence, not causation; both may share a cause, such as parents shopping late, and the idea should be tested before acting

1698. A telecom firm has three years of customer records, each labelled 'left' or 'stayed'. It wants to predict which current customers will leave. A junior analyst suggests clustering. What is the better choice? Analysis

A. Data cube roll-up, because it summarises churn
B. Association rule mining, because it predicts individual behaviour
C. Classification, because labelled past outcomes exist and the goal is to predict a class
D. Clustering, because customers should be grouped

SQL Joins Practice for the NSCT

1699. Which join returns only the rows that have a matching value in both tables? Understanding

A. INNER JOIN
B. LEFT OUTER JOIN
C. FULL OUTER JOIN
D. CROSS JOIN

1700. In a LEFT JOIN, a left-table row has no matching row in the right table. What appears in the right table's columns for that row? Understanding

A. The row is dropped from the result
B. Zero or an empty string, depending on the column type
C. The values from the first row of the right table
D. NULL in every right-table column

1701. Both tables have a row whose join column is NULL. Why does an INNER JOIN ON a.k = b.k not pair those two rows? Understanding

A. SQL removes all rows containing NULL before any join runs
B. The two NULLs are treated as different data types
C. INNER JOIN only works on primary key columns, which cannot hold NULL
D. NULL = NULL evaluates to UNKNOWN, and a join keeps only pairs where the condition is TRUE

1702. Which query returns the same rows as A RIGHT JOIN B ON A.id = B.id (ignoring column order)? Understanding

A. A LEFT JOIN B ON A.id = B.id
B. B LEFT JOIN A ON A.id = B.id
C. A INNER JOIN B ON A.id = B.id
D. A FULL OUTER JOIN B ON A.id = B.id

1703. What must you use to join a table to itself, for example to list each employee beside their manager? Understanding

A. A UNION of the table with itself
B. A CROSS JOIN with no condition
C. Two different table aliases, such as emp e and emp m
D. A separate copy of the table created with CREATE TABLE

1704. What does the pattern LEFT JOIN B ON A.k = B.k WHERE B.k IS NULL return? Understanding

A. Rows of B that have no matching row in A
B. Rows of A that have no matching row in B
C. Rows of A whose k value is NULL only
D. Pairs of rows that match in both tables

1705. How many rows does the INNER JOIN return? Application

```
CREATE TABLE course (cid INT, title TEXT);
INSERT INTO course VALUES (1,'DB'), (2,'OS'), (3,'AI');
CREATE TABLE enrol (sid INT, cid INT);
INSERT INTO enrol VALUES (10,1), (11,1), (12,2), (13,4);
SELECT e.sid, c.title
FROM enrol e INNER JOIN course c ON e.cid = c.cid;
```

A. 4
B. 3
C. 2
D. 12

1706. How many rows does the LEFT JOIN return? Application

```
CREATE TABLE cust (id INT, name TEXT);
INSERT INTO cust VALUES (1,'Ali'), (2,'Sara'), (3,'Omar');
CREATE TABLE ord (oid INT, cust_id INT);
INSERT INTO ord VALUES (100,1), (101,1), (102,1), (103,3), (104,9);
SELECT c.name, o.oid
FROM cust c LEFT JOIN ord o ON c.id = o.cust_id;
```

A. 3
B. 4
C. 6
D. 5

1707. How many rows does the FULL OUTER JOIN return? (SQLite supports FULL OUTER JOIN from version 3.39.) Application

```
CREATE TABLE a (k INT); INSERT INTO a VALUES (1), (2), (3), (3);
CREATE TABLE b (k INT); INSERT INTO b VALUES (3), (4), (5);
SELECT a.k, b.k FROM a FULL OUTER JOIN b ON a.k = b.k;
```

A. 6
B. 5
C. 7
D. 12

1708. What does this self join return? Application

```
CREATE TABLE emp (eid INT, name TEXT, mgr INT);
INSERT INTO emp VALUES (1,'Nadia',NULL), (2,'Faraz',1),
  (3,'Kiran',1), (4,'Imran',3), (5,'Saba',3);
SELECT m.name, COUNT(*) AS reports
FROM emp e JOIN emp m ON e.mgr = m.eid
GROUP BY m.name ORDER BY m.name;
```

A. Kiran 2, Nadia 2
B. Faraz 1, Kiran 2, Nadia 2
C. Kiran 2, Nadia 2, NULL 1
D. Nadia 4

1709. Neither table has a unique key. How many rows does the join return? Application

```
CREATE TABLE r (k INT); INSERT INTO r VALUES (1), (1), (2), (2), (2), (7);
CREATE TABLE s (k INT); INSERT INTO s VALUES (1), (1), (1), (2), (8);
SELECT * FROM r JOIN s ON r.k = s.k;
```

A. 5
B. 9
C. 6
D. 2

1710. Which department appears with a NULL employee name in the RIGHT JOIN output? Application

```
CREATE TABLE emp (name TEXT, did INT);
INSERT INTO emp VALUES ('Ali',1), ('Sara',2), ('Omar',2), ('Hina',5);
CREATE TABLE dept (did INT, dname TEXT);
INSERT INTO dept VALUES (1,'HR'), (2,'IT'), (3,'Finance');
SELECT d.dname, e.name
FROM emp e RIGHT JOIN dept d ON e.did = d.did
ORDER BY d.did, e.name;
```

A. HR
B. IT
C. No department; Hina appears with a NULL department instead
D. Finance

1711. How many rows does the CROSS JOIN produce? Application

```
CREATE TABLE size (s TEXT); INSERT INTO size VALUES ('S'), ('M'), ('L'), ('XL');
CREATE TABLE colour (c TEXT); INSERT INTO colour VALUES ('Red'), ('Blue'), ('Black');
SELECT s, c FROM size CROSS JOIN colour;
```

A. 12
B. 7
C. 4
D. 3

1712. How many rows does the query return? Application

```
CREATE TABLE x (a INT); INSERT INTO x VALUES (1), (2), (3), (4);
CREATE TABLE y (b INT); INSERT INTO y VALUES (2), (3), (5);
SELECT a, b FROM x CROSS JOIN y WHERE a < b;
```

A. 12
B. 5
C. 7
D. 4

1713. The two queries differ only in where the condition o.amount > 100 is placed. How many rows does each return? Analysis

```
CREATE TABLE cust (id INT, name TEXT);
INSERT INTO cust VALUES (1,'Ali'), (2,'Sara'), (3,'Omar');
CREATE TABLE ord (cust_id INT, amount INT);
INSERT INTO ord VALUES (1,50), (1,200), (2,80);
-- Q1
SELECT c.name, o.amount FROM cust c
LEFT JOIN ord o ON c.id = o.cust_id AND o.amount > 100;
-- Q2
SELECT c.name, o.amount FROM cust c
LEFT JOIN ord o ON c.id = o.cust_id WHERE o.amount > 100;
```

A. Q1: 3 rows, Q2: 1 row
B. Q1: 1 row, Q2: 3 rows
C. Both return 3 rows
D. Q1: 4 rows, Q2: 1 row

1714. The report should show 0 for customers with no orders. What does it show for Omar, and why? Analysis

```
CREATE TABLE cust (id INT, name TEXT);
INSERT INTO cust VALUES (1,'Ali'), (2,'Omar');
CREATE TABLE ord (oid INT, cust_id INT);
INSERT INTO ord VALUES (10,1), (11,1);
SELECT c.name, COUNT(*) AS orders
FROM cust c LEFT JOIN ord o ON c.id = o.cust_id
GROUP BY c.name ORDER BY c.name;
```

A. 0, because Omar has no orders
B. NULL, because there is nothing to count
C. 1, because COUNT(*) counts Omar's NULL-padded row; COUNT(o.oid) would give 0
D. Omar does not appear, because GROUP BY drops NULL rows

1715. The query is meant to find students without a valid department. Which names does it return? Analysis

```
CREATE TABLE student (name TEXT, did INT);
INSERT INTO student VALUES ('Ali',1), ('Sara',NULL), ('Omar',3), ('Hina',2);
CREATE TABLE dept (did INT, dname TEXT);
INSERT INTO dept VALUES (1,'CS'), (2,'EE');
SELECT s.name FROM student s
LEFT JOIN dept d ON s.did = d.did
WHERE d.did IS NULL ORDER BY s.name;
```

A. Omar only
B. Sara only
C. No rows
D. Omar and Sara

1716. Ali placed orders totalling 300 and made payments totalling 250. Why are the totals wrong, and what does the query show? Analysis

```
CREATE TABLE ord (cust TEXT, amount INT);
INSERT INTO ord VALUES ('Ali',100), ('Ali',200);
CREATE TABLE pay (cust TEXT, paid INT);
INSERT INTO pay VALUES ('Ali',50), ('Ali',100), ('Ali',100);
SELECT o.cust, SUM(o.amount), SUM(p.paid)
FROM ord o JOIN pay p ON o.cust = p.cust
GROUP BY o.cust;
```

A. Ali, 300, 250: the join does not change the sums
B. Ali, 900, 500: each order row is repeated for all 3 payments and each payment for both orders
C. Ali, 600, 750: each table is counted twice
D. Ali, 300, 500: only the payments are duplicated

1717. Both tables have a column called name. How many rows does the NATURAL JOIN return? Analysis

```
CREATE TABLE emp (eid INT, name TEXT, did INT);
INSERT INTO emp VALUES (1,'Ali',10), (2,'Sara',20), (3,'Omar',10);
CREATE TABLE dept (did INT, name TEXT);
INSERT INTO dept VALUES (10,'CS'), (20,'EE');
SELECT * FROM emp NATURAL JOIN dept;
```

A. 3
B. 2
C. 0
D. 6

1718. The query should list every student, even those without a course. Which students actually appear? Analysis

```
CREATE TABLE student (sid INT, name TEXT);
INSERT INTO student VALUES (1,'Ali'), (2,'Sara'), (3,'Omar');
CREATE TABLE enrol (sid INT, cid INT);
INSERT INTO enrol VALUES (1,100), (2,200);
CREATE TABLE course (cid INT, title TEXT);
INSERT INTO course VALUES (100,'DB');
SELECT s.name, c.title
FROM student s
LEFT JOIN enrol e ON s.sid = e.sid
JOIN course c ON e.cid = c.cid;
```

A. Ali, Sara and Omar
B. Ali and Sara
C. Ali only
D. Ali and Omar

Normalization Practice for the NSCT

1719. In R with functional dependency set F, attribute E appears on the right-hand side of some FDs but never on any left-hand side. What can you say about E? Understanding

A. E is not part of any candidate key
B. E is part of every candidate key
C. E must be a candidate key by itself
D. E is a prime attribute

1720. For relation R, the closure X^+ of an attribute set X contains every attribute of R. What is X? Understanding

A. Always a candidate key of R
B. A prime attribute set that must be minimal
C. A superkey of R, though not necessarily a candidate key
D. A trivial functional dependency

1721. In 3NF, a non-trivial FD X → A is still allowed when X is not a superkey, provided which condition holds? Understanding

A. A is a prime attribute (part of some candidate key)
B. X is a single attribute
C. A is a non-prime attribute
D. X is part of the primary key

1722. Which statement about the standard decomposition algorithms is correct? Understanding

A. Both BCNF decomposition and 3NF synthesis always preserve every dependency
B. BCNF decomposition is always lossless but may lose dependencies; 3NF synthesis is lossless and dependency-preserving
C. 3NF synthesis may be lossy, but BCNF decomposition never is
D. BCNF decomposition always preserves dependencies but may be lossy

1723. What is a partial dependency, the thing 2NF removes? Understanding

A. A non-prime attribute that depends on another non-prime attribute
B. A prime attribute that depends on a non-key attribute
C. A dependency whose right-hand side is a subset of its left-hand side
D. A non-prime attribute that depends on a proper subset of a candidate key

1724. Attribute A appears on the left-hand side of some FDs but never on any right-hand side. What follows? Understanding

A. A is in no candidate key
B. A is a non-prime attribute
C. A can be dropped from R without losing information
D. A is in every candidate key

1725. R(A, B, C, D, E, F) has F = {A → C, B → D, CD → E, E → F}. What is A^+? Application

A. {A, C, E, F}
B. {A, C}
C. {A, B, C, D, E, F}
D. {A}

1726. R(A, B, C, D, E, F) has F = {A → B, C → D, BD → E, E → F}. What is $(BC)^+$, and is BC a superkey? Application

A. {A, B, C, D, E, F}; a superkey
B. {B, C, D, E, F}; not a superkey
C. {B, C, D}; not a superkey
D. {B, C, D, E}; not a superkey

1727. R(A, B, C, D, E) has F = {AB → C, C → D, D → B, D → E}. What are all the candidate keys? Application

A. AB only
B. A only
C. AB and AD
D. AB, AC and AD

1728. R(A, B, C, D) has F = {A → B, B → C, C → D, D → A}. How many candidate keys does R have? Application

A. 1
B. 2
C. 4
D. 0

1729. R(A, B, C, D, E, F) has F = {AB → C, C → D, D → EF}. What is the highest normal form of R? Application

A. 1NF
B. 3NF
C. BCNF
D. 2NF

1730. R(A, B, C, D, E) has F = {A → BC, CD → E, B → D, E → A}. Which list gives exactly the candidate keys? Application

A. A and E
B. A, E, BC and CD
C. A, E and BC
D. A, BC and CD

1731. R(A, B, C, D) has F = {A → B, C → D}. Which decomposition is lossless? Application

A. R1(A, B), R2(A, C, D)
B. R1(A, B), R2(C, D)
C. R1(A, B, D), R2(B, C)
D. R1(A, D), R2(B, C)

1732. What is a minimal (canonical) cover of F = {AB → C, A → B, B → C, AC → D}? Application

A. {A → B, B → C, AC → D}
B. {AB → C, A → B, AC → D}
C. {A → B, B → C, A → D}
D. {A → B, A → C, A → D}

1733. R(A, B, C, D, E) has F = {AB → C, C → D, D → B, D → E}. A student says R is in 3NF because every left-hand side contains a key attribute. What is the highest normal form? Analysis

A. 2NF
B. 3NF
C. 1NF
D. BCNF

1734. R(A, B, C, D) has F = {AB → CD, D → A}. Splitting on D → A gives R1(D, A) and R2(B, C, D). Which statement is correct? Analysis

A. Lossless, both parts in BCNF, but AB → CD is not preserved
B. Lossy, because the parts share only D
C. Lossless and every original FD is preserved
D. R was already in BCNF, so the split was unnecessary

1735. R(A, B, C, D) has F = {AB → C, C → A}. Which statement about R is correct? Analysis

A. R is in BCNF, because every FD has a key attribute on its left
B. R is only in 1NF, because C → A is a partial dependency
C. R is in 3NF but not BCNF, because C → A has a non-superkey left side and A is prime
D. R is in 2NF but not 3NF, because C → A is transitive

1736. Rows of Lab(sid, course, teacher, room): (S1, DB, Saad, R1), (S2, DB, Saad, R1), (S1, OS, Amna, R2), (S3, AI, Saad, R3), (S2, OS, Amna, R2). Which FD is definitely violated by this data? Analysis

A. course → teacher
B. teacher → room
C. course → room
D. room → teacher

1737. R(A, B, C, D) has exactly two candidate keys, A and BC. How many superkeys does R have? Analysis

A. 12
B. 8
C. 4
D. 10

1738. Given F = {A → B, B → C, C → D}, which functional dependency is NOT implied by F? Analysis

A. D → A
B. A → D
C. AC → BD
D. B → CD

ACID Properties Practice for the NSCT

1739. A booking transaction marks seat 14C as taken and then fails while inserting the payment row. Afterwards the seat stays marked as taken with no payment. Which ACID property was violated? Understanding

A. Isolation
B. Durability
C. Consistency only, because the data is wrong
D. Atomicity

1740. Two booking clerks run at the same moment. Each reads seat 14C as free and each books it, so two passengers hold the same seat. Each transaction on its own was correct. Which property failed? Understanding

A. Atomicity
B. Isolation
C. Durability
D. Consistency of each individual transaction

1741. A table declares CHECK (balance >= 0). An UPDATE inside a transaction would make one balance −200, so the DBMS rejects the statement. Which ACID property is this protecting most directly? Understanding

A. Durability
B. Isolation
C. Atomicity of concurrent transactions
D. Consistency

1742. Which pairing of ACID properties with the DBMS mechanisms that mainly enforce them is correct? Understanding

A. Atomicity: concurrency control. Isolation: the undo log. Durability: integrity constraints
B. Atomicity: undo using the log. Isolation: concurrency control. Durability: redo using the log on stable storage
C. Atomicity: redo using the log. Isolation: integrity constraints. Durability: locking
D. Atomicity: checkpoints only. Isolation: backups. Durability: the query optimiser

1743. Under the SQL-92 standard, which is the only isolation level that rules out phantom reads? Understanding

A. SERIALIZABLE
B. REPEATABLE READ
C. READ COMMITTED
D. READ UNCOMMITTED

1744. How does the C in ACID differ from the C in the CAP theorem? Understanding

A. They mean the same thing: all replicas hold identical data
B. ACID consistency is about replicas agreeing; CAP consistency is about CHECK constraints
C. ACID consistency means integrity rules hold after each transaction; CAP consistency means every node returns the latest write
D. ACID consistency is enforced by the network; CAP consistency by the lock manager

1745. Account 1 starts at 1000 and account 2 at 500. What does the final SELECT return? Application

```
CREATE TABLE acct (id INT PRIMARY KEY, bal INT CHECK (bal >= 0));
INSERT INTO acct VALUES (1, 1000), (2, 500);
BEGIN;
UPDATE acct SET bal = bal - 100 WHERE id = 1;
SAVEPOINT s1;
UPDATE acct SET bal = bal - 400 WHERE id = 1;
UPDATE acct SET bal = bal + 400 WHERE id = 2;
ROLLBACK TO s1;
UPDATE acct SET bal = bal + 100 WHERE id = 2;
COMMIT;
SELECT id, bal FROM acct ORDER BY id;
```

A. (1, 500) and (2, 1000)
B. (1, 1000) and (2, 500)
C. (1, 900) and (2, 600)
D. (1, 900) and (2, 1000)

1746. The UPDATE fails because account 2 would go negative. Assuming the program continues to the SELECT, what does it show? Application

```
CREATE TABLE acct (id INT PRIMARY KEY, bal INT CHECK (bal >= 0));
INSERT INTO acct VALUES (1, 500), (2, 150), (3, 900);
UPDATE acct SET bal = bal - 200;   -- fails with a CHECK error
SELECT id, bal FROM acct ORDER BY id;
```

A. 500, 150 and 900, unchanged
B. 300, 150 and 700
C. 300, −50 and 700
D. 300, 150 and 900

1747. X = 500 and Y = 0. T1 sets X to 800 but has not committed. T2 reads X, sets Y = X / 10 and commits. T1 then aborts. What are X and Y now, and what should Y be? Application

A. X = 800, Y = 80; Y is correct
B. X = 500, Y = 50; Y is correct
C. X = 500, Y = 0; T2 is rolled back too
D. X = 500, Y = 80; Y should be 50

1748. A = 2000 and B = 1000. T2 computes A + B. It reads A, then T1 moves 500 from A to B and commits, then T2 reads B. What total does T2 report? Application

A. 3000
B. 2500
C. 3500
D. 4000

1749. The log holds <T1 start>, <T1, A, 1000, 900>, <T1, B, 2000, 2100> and no commit record for T1. On disk, A = 900 and B = 2000 at the crash. What are A and B after recovery? Application

A. A = 1000, B = 2000
B. A = 900, B = 2100
C. A = 900, B = 2000
D. A = 1000, B = 2100

1750. Under READ COMMITTED, T1 reads X (50). T2 then updates X to 70 and commits. T1 reads X again. What does T1's second read return? Application

A. 70
B. 50
C. An error, because X changed
D. NULL until T1 commits

1751. Before a crash the log on disk holds <T1 start>, <T1, A, 100, 150>, <T1 commit>, <T2 start>, <T2, B, 300, 200>. The data pages on disk still show A = 100 and B = 200. With UNDO/REDO recovery, what are A and B afterwards? Application

A. A = 100, B = 300
B. A = 150, B = 300
C. A = 150, B = 200
D. A = 100, B = 200

1752. Accounts A = 1200, B = 800 and C = 0 must always total 2000. Which committed transaction keeps the invariant? Application

A. A := A − 300; B := B + 200
B. B := B + 300; C := C + 300
C. A := A − 300
D. A := A − 300; C := C + 300

1753. X = 100. T1 doubles X and T2 adds 50 to X; each reads X and then writes it. Which set lists exactly the final values a serialisable execution can produce? Analysis

    A. 250 only
    B. 250 or 300
    C. 150, 200, 250 or 300
    D. 300 only

1754. An app runs the debit and the credit of a transfer as two separate statements in autocommit mode. The server crashes between them. Which statement is correct? Analysis

    A. The debit is permanent and the credit never happened, because each statement was its own transaction; wrapping both in BEGIN ... COMMIT fixes it
    B. Recovery undoes the debit, because the DBMS knows the two statements belong together
    C. Durability is violated, because the credit was lost
    D. Isolation is violated, because another transaction could see the debit

1755. To speed up writes, a DBA lets COMMIT return before the commit log record is flushed to disk (asynchronous commit). After a power cut, the last few acknowledged transactions are missing. Which property was traded away? Analysis

    A. Atomicity, because some transactions were half-applied
    B. Isolation, because transactions overlapped
    C. Durability, but atomicity and consistency still hold for the surviving data
    D. Consistency, because constraints were not checked

1756. In SQLite, the application ignores the error on the second INSERT and carries on. What does the final SELECT return? Analysis

```
CREATE TABLE t (id INT PRIMARY KEY);
BEGIN;
INSERT INTO t VALUES (1);
INSERT INTO t VALUES (1);   -- fails: duplicate primary key
INSERT INTO t VALUES (3);
COMMIT;
SELECT id FROM t ORDER BY id;
```

    A. Nothing, because the error rolled back the whole transaction
    B. 1 only
    C. 1 and 3
    D. 1, 1 and 3

1757. A buggy transfer debits A by 100 and credits B by only 10, then commits successfully. Nothing crashed and no one else was running. Which property is broken? Analysis

    A. Atomicity, because only part of the money moved
    B. Isolation, because the amounts differ
    C. Durability, because 90 disappeared
    D. Consistency, because the transaction itself breaks the money-conservation rule

1758. Balance = 1000. Two app sessions each SELECT the balance, compute a new value in code (−200 and +500) and write it back with UPDATE ... SET bal = <computed value>. Both read before either writes, and the result is 1500 instead of 1300. Which change fixes this? Analysis

    A. Add CHECK (bal >= 0) to the table
    B. Let the database do the arithmetic, as in UPDATE acct SET bal = bal + 500, so each change applies to the current value under a row lock
    C. Switch both sessions to READ UNCOMMITTED so they see each other's changes
    D. Add an index on the id column

SQL Query Output Questions for the NSCT

1759. How many rows (groups) does this query return? Understanding

```
CREATE TABLE emp (name TEXT, dept TEXT);
INSERT INTO emp VALUES ('A','IT'), ('B',NULL), ('C','HR'), ('D',NULL), ('E','IT'), ('F','Ops');
SELECT dept, COUNT(*) FROM emp GROUP BY dept;
```

A. 4
B. 3
C. 5
D. 6

1760. The scalar subquery finds no salary above 100. What does the query show in the top column? Understanding

```
CREATE TABLE emp (name TEXT, salary INT);
INSERT INTO emp VALUES ('A', 40), ('B', 60);
SELECT name, (SELECT MAX(salary) FROM emp WHERE salary > 100) AS top FROM emp WHERE name = 'A';
```

A. NULL
B. 0
C. An error: the subquery returned no rows
D. 40

1761. Every value in column x is NULL. What do COUNT(x) and SUM(x) return? Understanding

```
CREATE TABLE t (x INT);
INSERT INTO t VALUES (NULL), (NULL);
SELECT COUNT(x), SUM(x) FROM t;
```

A. 0 and 0
B. NULL and NULL
C. 0 and NULL
D. 2 and NULL

1762. The table is empty and there is no GROUP BY. What does the query return? Understanding

```
CREATE TABLE t (x INT);
SELECT COUNT(*), MAX(x) FROM t;
```

A. No rows
B. One row: 0 and 0
C. One row: NULL and NULL
D. One row: 0 and NULL

1763. What does ORDER BY 2 DESC sort by, and which name comes first? Understanding

```
CREATE TABLE p (name TEXT, city TEXT, age INT);
INSERT INTO p VALUES ('Ali','LHR',22), ('Sara','KHI',25), ('Omar','ISB',21);
SELECT name, age FROM p ORDER BY 2 DESC;
```

A. The second column of the table (city); Omar first
B. The constant 2, so the order is unchanged; Ali first
C. The second row; Sara first
D. The second column in the SELECT list (age); Sara first

1764. There is HAVING but no GROUP BY. What does the query return? Understanding

```
CREATE TABLE t (x INT);
INSERT INTO t VALUES (5), (10), (15);
SELECT SUM(x) FROM t HAVING SUM(x) > 100;
```

A. One row: 30
B. No rows
C. One row: NULL
D. An error: HAVING needs GROUP BY

1765. What does the query return? Application

```
CREATE TABLE orders (oid INT, cust TEXT, city TEXT, amount INT);
INSERT INTO orders VALUES
 (1,'Ali','LHR',300), (2,'Sara','KHI',150), (3,'Ali','LHR',200), (4,'Omar','ISB',NULL),
 (5,'Hina','KHI',400), (6,'Sara','KHI',250), (7,'Zain','LHR',100), (8,'Omar','ISB',50);
SELECT city, COUNT(*) FROM orders GROUP BY city ORDER BY city;
```

A. ISB 1, KHI 3, LHR 3
B. ISB 2, KHI 2, LHR 3
C. KHI 3, LHR 3
D. ISB 2, KHI 3, LHR 3

1766. What does the query return? Application

```
CREATE TABLE marks (sid TEXT, course TEXT, score INT);
INSERT INTO marks VALUES ('S1','DB',72), ('S1','OS',45), ('S2','DB',88),
 ('S2','OS',91), ('S3','DB',39), ('S3','OS',58), ('S3','AI',64);
SELECT sid, AVG(score) FROM marks
WHERE score >= 50
GROUP BY sid HAVING COUNT(*) >= 2
ORDER BY sid;
```

A. S1 58.5, S2 89.5 and S3 53.67
B. S2 89.5 and S3 61.0
C. S2 89.5 only
D. S1 72.0, S2 89.5 and S3 61.0

1767. Which order ids are returned? Application

```
CREATE TABLE orders (oid INT, cust TEXT, city TEXT, amount INT);
INSERT INTO orders VALUES
 (1,'Ali','LHR',300), (2,'Sara','KHI',150), (3,'Ali','LHR',200), (4,'Omar','ISB',NULL),
 (5,'Hina','KHI',400), (6,'Sara','KHI',250), (7,'Zain','LHR',100), (8,'Omar','ISB',50);
SELECT oid FROM orders WHERE amount > (SELECT AVG(amount) FROM orders) ORDER BY oid;
```

A. 1 and 5
B. 1, 3, 5 and 6
C. 1, 5 and 6
D. 5 only

1768. Which names does the correlated subquery return? Application

```
CREATE TABLE emp (name TEXT, dept TEXT, salary INT);
INSERT INTO emp VALUES ('Ali','IT',90), ('Sara','IT',120), ('Omar','HR',70),
 ('Hina','HR',70), ('Zain','Ops',55);
SELECT e.name FROM emp e
WHERE e.salary = (SELECT MAX(salary) FROM emp i WHERE i.dept = e.dept)
ORDER BY e.name;
```

A. Omar, Sara and Zain
B. Sara only
C. Hina, Omar, Sara and Zain
D. Hina, Sara and Zain

1769. Which customers are returned? Application

```
CREATE TABLE orders (oid INT, cust TEXT, city TEXT, amount INT);
INSERT INTO orders VALUES
 (1,'Ali','LHR',300), (2,'Sara','KHI',150), (3,'Ali','LHR',200), (4,'Omar','ISB',NULL),
 (5,'Hina','KHI',400), (6,'Sara','KHI',250), (7,'Zain','LHR',100), (8,'Omar','ISB',50);
SELECT DISTINCT cust FROM orders WHERE city IN (SELECT city FROM orders GROUP BY city HAVING
SUM(amount) > 650) ORDER BY cust;
```

A. Ali, Hina, Sara and Zain
B. Ali and Zain
C. Hina, Sara and Sara
D. Hina and Sara

1770. What does the query return? Application

```
CREATE TABLE emp (name TEXT, salary INT);
INSERT INTO emp VALUES ('A',50), ('B',80), ('C',80), ('D',65), ('E',40);
SELECT MAX(salary) FROM emp WHERE salary < (SELECT MAX(salary) FROM emp);
```

A. 80
B. 65
C. 50
D. NULL

1771. What does the query return? Application

```
CREATE TABLE score (name TEXT, pts INT);
INSERT INTO score VALUES ('A',70), ('B',85), ('C',60), ('D',85), ('E',90), ('F',75);
SELECT name FROM score ORDER BY pts DESC, name LIMIT 2 OFFSET 2;
```

A. B and D
B. D and F
C. E and B
D. F and A

1772. Which students are returned? (Using a SELECT alias in HAVING is allowed in SQLite and MySQL; standard SQL would repeat the expression.) Application

```
CREATE TABLE att (sid TEXT, status TEXT);
INSERT INTO att VALUES ('S1','P'), ('S1','A'), ('S1','P'), ('S2','P'),
 ('S2','L'), ('S2','A'), ('S2','A'), ('S3','P');
SELECT sid,
       SUM(CASE WHEN status = 'P' THEN 1 ELSE 0 END) AS present,
       COUNT(*) AS total
FROM att GROUP BY sid HAVING present * 2 > total
ORDER BY sid;
```

A. S1, S2 and S3
B. S1 and S3
C. S3 only
D. S1 only

1773. A student expects both expressions to give the same average. What do they return? Analysis

```
CREATE TABLE t (x INT);
INSERT INTO t VALUES (10), (NULL), (20), (NULL), (30);
SELECT AVG(x), SUM(x) * 1.0 / COUNT(*) FROM t;
```

A. 12.0 and 12.0
B. 20.0 and 20.0
C. 20.0 and 12.0
D. 12.0 and 20.0

1774. Q1 and Q2 are both meant to find 'repeat customers with big orders'. What does each return? Analysis

```
CREATE TABLE orders (cust TEXT, amount INT);
INSERT INTO orders VALUES ('Ali',50), ('Ali',300), ('Sara',150), ('Sara',120),
 ('Omar',500), ('Hina',80), ('Hina',90);
-- Q1
SELECT cust FROM orders WHERE amount > 100 GROUP BY cust HAVING COUNT(*) > 1;
-- Q2
SELECT cust FROM orders GROUP BY cust HAVING COUNT(*) > 1 AND MAX(amount) > 100;
```

A. Q1: Ali and Sara. Q2: Sara
B. Both return Ali and Sara
C. Q1: Sara and Omar. Q2: Ali and Sara
D. Q1: Sara. Q2: Ali and Sara

1775. The query is meant to return the 'top 2'. What does it actually return? Analysis

```
CREATE TABLE s (name TEXT, pts INT);
INSERT INTO s VALUES ('A',90), ('B',85), ('C',85), ('D',70), ('E',95);
SELECT a.name FROM s a
WHERE (SELECT COUNT(*) FROM s b WHERE b.pts > a.pts) < 2
ORDER BY a.name;
```

A. A and E
B. A, B, C and E
C. E only
D. A, B and E

1776. What three values does the query return? Analysis

```
CREATE TABLE t (a INT, b INT);
INSERT INTO t VALUES (1,1), (1,1), (1,2), (2,NULL), (2,NULL), (NULL,NULL);
SELECT COUNT(*), COUNT(DISTINCT a), (SELECT COUNT(*) FROM (SELECT DISTINCT a, b FROM t))
FROM t;
```

A. 6, 2 and 4
B. 6, 3 and 4
C. 6, 2 and 3
D. 5, 2 and 6

1777. The manager wants the average sales per rep in each region. What does the query return? Analysis

```
CREATE TABLE sales (rep TEXT, region TEXT, amt INT);
INSERT INTO sales VALUES ('A','N',100), ('A','N',300), ('B','N',200),
 ('C','S',600), ('D','S',50), ('D','S',50);
SELECT region, AVG(total) FROM (
  SELECT region, rep, SUM(amt) AS total FROM sales GROUP BY region, rep
) GROUP BY region ORDER BY region;
```

A. N 300.0, S 350.0
B. N 200.0, S 233.33
C. N 300.0, S 233.33
D. N 200.0, S 350.0

1778. The query is meant to report the smallest department. What is wrong with it? Analysis

```
CREATE TABLE emp (name TEXT, dept TEXT);
INSERT INTO emp VALUES ('A','IT'), ('B','IT'), ('C','HR'), ('D','IT'), ('E','Ops');
SELECT dept, COUNT(*) FROM emp GROUP BY dept ORDER BY COUNT(*) LIMIT 1;
```

A. It returns IT, because ORDER BY sorts in descending order by default
B. It fails, because an aggregate cannot appear in ORDER BY
C. HR and Ops tie with 1 employee each, but LIMIT 1 returns only one of them, and which one is not guaranteed
D. It returns both HR and Ops, so nothing is wrong

Relational Algebra Practice for the NSCT

1779. Which relational algebra operator can reduce the number of attributes (the degree) of a relation? Understanding

A. Selection (σ)
B. Projection (π)
C. Union (∪)
D. Cartesian product (×)

1780. Which set operation is NOT commutative? Understanding

A. Union (R ∪ S)
B. Difference (R − S)
C. Intersection (R ∩ S)
D. Natural join (R ⋈ S), ignoring column order

1781. R has attributes (sid, cid) and S has the single attribute (cid). What are the attributes of R ÷ S? Understanding

A. (cid)
B. (sid, cid)
C. (sid)
D. No attributes; division returns TRUE or FALSE

1782. R and S have exactly the same set of attributes. What does the natural join R ⋈ S produce? Understanding

A. R ∪ S
B. R × S
C. An empty relation
D. R ∩ S

1783. R has 12 tuples. What are the possible numbers of tuples in σ_p(R), for any predicate p? Understanding

A. Exactly 12
B. From 1 to 12
C. From 0 to 144
D. Anywhere from 0 to 12

1784. For union-compatible R and S, which equivalence always holds? Understanding

A. π_A(R ∪ S) = π_A(R) ∪ π_A(S)
B. π_A(R − S) = π_A(R) − π_A(S)
C. π_A(R ∩ S) = π_A(R) ∩ π_A(S)
D. σ_p(R) − S = σ_p(R − S) ∪ S

1785. Using the relations shown, how many tuples does σ_dept='IT' ∧ sal≥80(Emp) return? Application

```
Emp(eid, name, dept, sal):
  (E1, Asad, IT, 70)  (E2, Bina, HR, 50)  (E3, Cyrus, IT, 90)
  (E4, Dua, Ops, 50)  (E5, Ehsan, IT, 90)
Works(eid, pid):
  (E1, P1) (E1, P2) (E2, P1) (E3, P1) (E3, P2) (E3, P3) (E5, P2)
Proj(pid, budget):
  (P1, 100) (P2, 250) (P3, 400)
```

A. 3
B. 2
C. 1
D. 4

1786. Using the relations shown, how many tuples does π_sal(Emp) return? Application

```
Emp(eid, name, dept, sal):
  (E1, Asad, IT, 70)  (E2, Bina, HR, 50)  (E3, Cyrus, IT, 90)
  (E4, Dua, Ops, 50)  (E5, Ehsan, IT, 90)
Works(eid, pid):
  (E1, P1) (E1, P2) (E2, P1) (E3, P1) (E3, P2) (E3, P3) (E5, P2)
Proj(pid, budget):
  (P1, 100) (P2, 250) (P3, 400)
```

A. 5
B. 2
C. 3
D. 4

1787. Using the relations shown, how many tuples does Emp ⋈ Works contain? Application

```
Emp(eid, name, dept, sal):
  (E1, Asad, IT, 70)  (E2, Bina, HR, 50)  (E3, Cyrus, IT, 90)
  (E4, Dua, Ops, 50)  (E5, Ehsan, IT, 90)
Works(eid, pid):
  (E1, P1) (E1, P2) (E2, P1) (E3, P1) (E3, P2) (E3, P3) (E5, P2)
Proj(pid, budget):
  (P1, 100) (P2, 250) (P3, 400)
```

A. 7
B. 5
C. 35
D. 6

1788. Using the relations shown, E2 is a renamed copy of Emp. How many tuples are in Emp ⋈_Emp.sal<E2.sal E2? Application

```
Emp(eid, name, dept, sal):
  (E1, Asad, IT, 70)  (E2, Bina, HR, 50)  (E3, Cyrus, IT, 90)
  (E4, Dua, Ops, 50)  (E5, Ehsan, IT, 90)
Works(eid, pid):
  (E1, P1) (E1, P2) (E2, P1) (E3, P1) (E3, P2) (E3, P3) (E5, P2)
Proj(pid, budget):
  (P1, 100) (P2, 250) (P3, 400)
```

A. 10
B. 8
C. 6
D. 25

1789. Using the relations shown, what is π_name(Emp ⋈ (Works ÷ π_pid(Proj)))? Application

```
Emp(eid, name, dept, sal):
  (E1, Asad, IT, 70)  (E2, Bina, HR, 50)  (E3, Cyrus, IT, 90)
  (E4, Dua, Ops, 50)  (E5, Ehsan, IT, 90)
Works(eid, pid):
  (E1, P1) (E1, P2) (E2, P1) (E3, P1) (E3, P2) (E3, P3) (E5, P2)
Proj(pid, budget):
  (P1, 100) (P2, 250) (P3, 400)
```

A. {Asad, Cyrus}
B. {Cyrus, Ehsan}
C. {Cyrus}
D. {} (empty)

1790. Using the relations shown, what does π_eid(Emp) − π_eid(Works) return? Application

```
Emp(eid, name, dept, sal):
  (E1, Asad, IT, 70)  (E2, Bina, HR, 50)  (E3, Cyrus, IT, 90)
  (E4, Dua, Ops, 50)  (E5, Ehsan, IT, 90)
Works(eid, pid):
  (E1, P1) (E1, P2) (E2, P1) (E3, P1) (E3, P2) (E3, P3) (E5, P2)
Proj(pid, budget):
  (P1, 100) (P2, 250) (P3, 400)
```

A. {E4}
B. {E4, E5}
C. {E2, E4}
D. {} (empty)

1791. Using the relations shown, how many tuples does the left outer join Emp ⟕ Works contain, and how many of them have a NULL pid? Application

```
Emp(eid, name, dept, sal):
  (E1, Asad, IT, 70)  (E2, Bina, HR, 50)  (E3, Cyrus, IT, 90)
  (E4, Dua, Ops, 50)  (E5, Ehsan, IT, 90)
Works(eid, pid):
  (E1, P1) (E1, P2) (E2, P1) (E3, P1) (E3, P2) (E3, P3) (E5, P2)
Proj(pid, budget):
  (P1, 100) (P2, 250) (P3, 400)
```

A. 7 tuples, 0 with NULL pid
B. 8 tuples, 2 with NULL pid
C. 12 tuples, 5 with NULL pid
D. 8 tuples, 1 with NULL pid

1792. Using the relations shown, what is π_name(σ_budget>200(Emp ⋈ Works ⋈ Proj))? Application

```
Emp(eid, name, dept, sal):
  (E1, Asad, IT, 70)  (E2, Bina, HR, 50)  (E3, Cyrus, IT, 90)
  (E4, Dua, Ops, 50)  (E5, Ehsan, IT, 90)
Works(eid, pid):
  (E1, P1) (E1, P2) (E2, P1) (E3, P1) (E3, P2) (E3, P3) (E5, P2)
Proj(pid, budget):
  (P1, 100) (P2, 250) (P3, 400)
```

A. {Cyrus}
B. {Asad, Bina, Cyrus, Ehsan}
C. {Cyrus, Ehsan}
D. {Asad, Cyrus, Ehsan}

1793. Using the relations shown, which expression returns the eids of employees who work on BOTH P1 and P2? Analysis

```
Emp(eid, name, dept, sal):
  (E1, Asad, IT, 70)  (E2, Bina, HR, 50)  (E3, Cyrus, IT, 90)
  (E4, Dua, Ops, 50)  (E5, Ehsan, IT, 90)
Works(eid, pid):
  (E1, P1) (E1, P2) (E2, P1) (E3, P1) (E3, P2) (E3, P3) (E5, P2)
Proj(pid, budget):
  (P1, 100) (P2, 250) (P3, 400)
```

A. π_eid(σ_pid='P1' ∧ pid='P2'(Works))
B. π_eid(σ_pid='P1' ∨ pid='P2'(Works))
C. π_eid(σ_pid='P1'(Works)) ∪ π_eid(σ_pid='P2'(Works))
D. π_eid(σ_pid='P1'(Works)) ∩ π_eid(σ_pid='P2'(Works))

1794. Using the relations shown, Works ÷ π_pid(Proj) is computed as π_eid(Works) − π_eid((π_eid(Works) × π_pid(Proj)) − Works). How many tuples does the inner difference (π_eid(Works) × π_pid(Proj)) − Works contain? Analysis

```
Emp(eid, name, dept, sal):
  (E1, Asad, IT, 70)  (E2, Bina, HR, 50)  (E3, Cyrus, IT, 90)
  (E4, Dua, Ops, 50)  (E5, Ehsan, IT, 90)
Works(eid, pid):
  (E1, P1) (E1, P2) (E2, P1) (E3, P1) (E3, P2) (E3, P3) (E5, P2)
Proj(pid, budget):
  (P1, 100) (P2, 250) (P3, 400)
```

A. 7
B. 12
C. 5
D. 3

1795. Which expression returns the students ALL of whose grades are A, and what is its result? Analysis

```
Grade(sid, course, grade):
  (S1, DB, A) (S1, OS, A) (S2, DB, A) (S2, OS, B) (S3, AI, A) (S4, DB, C)
```

A. π_sid(σ_grade='A'(Grade)), giving {S1, S2, S3}
B. π_sid(σ_grade='A'(Grade)) ∩ π_sid(Grade), giving {S1, S2, S3}
C. π_sid(Grade) − π_sid(σ_grade≠'A'(Grade)), giving {S1, S3}
D. π_sid(Grade) − π_sid(σ_grade='A'(Grade)), giving {S4}

1796. Relational algebra has no MAX. Using the relations shown, with E2 a renamed copy of Emp, what does π_name(Emp) − π_Emp.name(Emp ⋈_Emp.sal<E2.sal E2) return? Analysis

```
Emp(eid, name, dept, sal):
  (E1, Asad, IT, 70)  (E2, Bina, HR, 50)  (E3, Cyrus, IT, 90)
  (E4, Dua, Ops, 50)  (E5, Ehsan, IT, 90)
Works(eid, pid):
  (E1, P1) (E1, P2) (E2, P1) (E3, P1) (E3, P2) (E3, P3) (E5, P2)
Proj(pid, budget):
  (P1, 100) (P2, 250) (P3, 400)
```

A. {Bina, Dua}
B. {Cyrus, Ehsan}
C. {Cyrus}
D. {Asad, Bina, Dua}

1797. R(A, B) has 4 tuples and S(B, C) has 5 tuples, with no keys declared. What are the smallest and largest possible sizes of R ⋈ S? Analysis

A. 0 and 20
B. 4 and 5
C. 0 and 9
D. 1 and 20

1798. A student writes π_eid(σ_Emp.eid≠Works.eid(Emp × Works)) to find employees who work on no project. Using the relations shown, what does it actually return? Analysis

```
Emp(eid, name, dept, sal):
  (E1, Asad, IT, 70)  (E2, Bina, HR, 50)  (E3, Cyrus, IT, 90)
  (E4, Dua, Ops, 50)  (E5, Ehsan, IT, 90)
Works(eid, pid):
  (E1, P1) (E1, P2) (E2, P1) (E3, P1) (E3, P2) (E3, P3) (E5, P2)
Proj(pid, budget):
  (P1, 100) (P2, 250) (P3, 400)
```

A. All five eids, E1 to E5
B. {E4}
C. {} (empty)
D. {E1, E2, E3, E5}

Answer key and explanations

1. **D** — Elmasri-Navathe define a DBMS as a general-purpose software system for defining, constructing, manipulating and sharing databases among users and applications. The collection of related data is the database itself, and database plus DBMS software is the database system.
2. **B** — A database system is the database together with the DBMS software, and Elmasri-Navathe often include the application programs too. The DBMS on its own is just the software, and the data on its own is the database.
3. **A** — Elmasri-Navathe call it the self-describing nature: the database holds not only data but a complete description of its own structure (metadata) in the catalog. This lets general-purpose DBMS software work with any database.
4. **D** — Metadata is data about data: descriptions of structure such as column names, data types, lengths and constraints, stored in the catalog. A CGPA value, a name value and a student's enrolments are ordinary data stored in the tables.
5. **C** — When the same data is duplicated in several files (redundancy), an update that reaches only some copies leaves them disagreeing. Silberschatz lists this as 'data redundancy and inconsistency', the first drawback of file-processing systems.
6. **D** — Silberschatz uses 'data isolation' for data spread over several files that may have different formats, which makes it hard to write new programs that combine them. It is not the 'I' of ACID, which is about concurrent transactions.
7. **B** — Elmasri-Navathe give these duties to the DBA, who is also accountable for security breaches and poor response time. Designers decide the structure before implementation; implementers build the DBMS software itself.
8. **C** — In Elmasri-Navathe, system analysts gather end-user requirements and write specifications for standard canned transactions; application programmers (software engineers) then implement, test and maintain them.
9. **D** — Database designers talk to prospective user groups, build a view for each group and integrate them into one design, mostly before any data is loaded. The DBA administers the database once it exists.
10. **D** — Program-data independence (insulation between programs and data) comes from the database approach. Data isolation, integrity problems and atomicity problems are all in Silberschatz's list of drawbacks of file-processing systems.
11. **B** — In file processing the record layout is coded inside each program, so changing it breaks the programs. A DBMS stores the structure in the catalog, separately from the programs. Elmasri-Navathe's example is adding a field to STUDENT without changing programs that do not use it.
12. **C** — Standalone users maintain personal databases with ready-made program packages that offer easy menu-based or graphics-based interfaces; a tax package is Elmasri-Navathe's own example. Naive users run canned transactions on a shared database.
13. **A** — These are among Elmasri-Navathe's 'workers behind the scene'. They implement the DBMS software as a product. Tool developers build optional packages such as design or performance tools; application programmers write programs that use a DBMS.
14. **D** — Elmasri-Navathe say a DBMS may be unnecessary for simple, well-defined applications that are not expected to change, for embedded systems with limited storage, for strict real-time needs and when there is no multiuser access. The other three need concurrency, recovery or sharing.
15. **C** — Ideally each logical data item is stored once, but Elmasri-Navathe note that some redundancy is sometimes kept to improve query performance. The point is that the DBMS controls it, for example by keeping a duplicated StudentName consistent.
16. **C** — In object and object-relational databases an operation has an interface (name and arguments) defined separately from its implementation (method). User programs depend only on the interface, so the method can be rewritten freely.
17. **C** — A view is a subset of the database, or virtual data derived from it, defined for one user group. A multiuser DBMS lets each group have its own view of one shared database instead of separate files.

18. **B** — Naive or parametric end users repeatedly run standard, pre-programmed (canned) transactions through simple interfaces. Bank tellers and reservation clerks are the textbook examples, and the billing clerk fits the same pattern.
19. **A** — ROLLBACK undoes every change made since BEGIN, so the debit from account 1 is cancelled and both balances are as they started. This is how a DBMS solves the atomicity problem of file systems, where a crash mid-transfer can leave money debited but never credited.
20. **D** — 4.5 is outside BETWEEN 0 AND 4, so the CHECK constraint fails. 4.0 and 0 are inside the inclusive range. A NULL makes the CHECK condition UNKNOWN, and SQL rejects a row only when a CHECK is FALSE, so NULL is accepted (add NOT NULL to forbid it).
21. **B** — sqlite_master holds one row per schema object, with type 'table', 'view', 'index' or 'trigger'. Only course and enrol are tables; the view and the two indexes (the explicit one and course's automatic primary-key index) have other types. Standard SQL exposes the catalog as INFORMATION_SCHEMA instead, where views appear in TABLES with TABLE_TYPE 'VIEW'.
22. **A** — Each student needs one copy; the other three are redundant. 3 × 30 bytes × 10,000 students = 900,000 bytes. 1,200,000 counts all four copies, including the necessary one.
23. **D** — A writes 10,000 − 1,000 = 9,000, then B overwrites it with 10,000 − 2,000 = 8,000, computed from the stale read. The correct result of both withdrawals is 7,000. This lost update is the concurrent-access anomaly that DBMS concurrency control prevents.
24. **A** — Silberschatz lists storage structure and access-method definition, and schema and physical-organisation modification, among the DBA's duties. Tuning indexes and storage is routine DBA work; it changes performance, not the logical design.
25. **B** — Casual end users access the database occasionally and need different information each time, using a query interface; Elmasri-Navathe's typical example is a middle- or high-level manager. Naive users repeat the same canned transactions.
26. **B** — The data is scattered across files with different formats, which is exactly Silberschatz's 'data isolation'. A DBMS stores it under one schema so a single query can combine it.
27. **D** — Integrity rules buried in application code are Silberschatz's 'integrity problems': hard to add or change and easy to miss. Declaring the constraint in the schema (for example a CHECK) makes the DBMS enforce it for all programs, now and future.
28. **A** — Restricting unauthorised access is a DBMS advantage in Elmasri-Navathe. The DBA creates accounts and grants privileges, often on views that show only the permitted columns. File systems make such fine-grained rules hard to enforce.
29. **A** — Each student has three copies; one is updated and two are left stale. 120 × 2 = 240 stale copies. 360 counts all three copies, and 120 counts one stale copy per student.
30. **A** — The transaction had not committed, so atomicity requires all-or-nothing. The recovery subsystem uses the log to undo the 400 changes, leaving the pre-transaction state; the program can then be rerun. Restoring a backup is only needed for media failure such as a disk crash.
31. **C** — Concurrent-access anomalies such as lost updates are handled by concurrency control (locking, timestamps), not by backup and recovery, which handles failures. The other three pairings are correct.
32. **C** — Relational designs repeat key values in foreign keys on purpose, and Elmasri-Navathe allow controlled redundancy for performance as long as the DBMS keeps it consistent. So 'never stores any value twice' is false; the goal is to avoid uncontrolled redundancy.
33. **B** — Two users reading and updating the same data at the same time, each unaware of the other, is a concurrent-access anomaly. A crash mid-transfer is an atomicity problem, an unsynchronised copy is redundancy-driven inconsistency, and a broken rule is an integrity problem.
34. **A** — This mirrors Elmasri-Navathe's own example. A DBMS keeps the structure in the catalog and maps programs onto it, so unaffected programs need no change. A file program embeds the layout, so adding a field shifts the byte offsets it relies on.
35. **C** — Writing application code is the application programmer's job, choosing structures before implementation is the designer's, granting privileges and taking backups belong to the DBA, and someone using fixed forms without writing queries is a naive (parametric) user.
36. **B** — (b) repeats primary key 1, (c) puts NULL into a NOT NULL column and (d) breaks the CHECK with 4.2. (a) is accepted: a NULL cgpa makes the CHECK UNKNOWN rather than FALSE, and cgpa has no NOT NULL.

37. **B** — Each clerk writes its read value plus its deposit, so the last write wins: 1,000 + 300 = 1,300. The correct balance is 1,000 + 600 = 1,600, so 300 is lost (the 100 and 200 deposits vanish).
38. **D** — Elmasri-Navathe stress that general-purpose DBMS software must work with any database, so it relies on the catalog to know file structures, types and constraints. A file-based program has this knowledge hard-coded instead.
39. **C** — What changed is that many users now update the same data at once, which risks lost updates and overselling. Concurrency control is the capability a spreadsheet lacks; persistence already existed.
40. **A** — In file processing the rule lives in every program, so all eight must change and be retested. With a declared CHECK constraint the rule lives once in the schema, so one change applies to every program.
41. **A** — Constructing is the process of storing the data itself on some storage medium controlled by the DBMS. Specifying types and constraints is defining, querying and updating is manipulating, and simultaneous access is sharing.
42. **B** — Defining a database means specifying the data types, structures and constraints for the data. The definition is stored in the catalog as metadata. Loading records is constructing; reports and privileges come later.
43. **C** — Elmasri-Navathe call the part of the real world a database models the miniworld or universe of discourse. Changes in the miniworld are reflected in the database. The schema and catalog describe the database, not the real world.
44. **D** — Workers behind the scene are DBMS system designers and implementers, tool developers, and operators and maintenance personnel. They do not use the database's content for their own work. The DBA, designers and end users are actors on the scene.
45. **A** — Elmasri-Navathe list 'providing multiple user interfaces' as an advantage: users with different technical knowledge get different interfaces to the same database. The other options are separate advantages.
46. **B** — Codd's 1970 paper introduced the relational model, which represents data as relations (tables). Hierarchical and network models came earlier, and object models became popular in the 1980s and 1990s.
47. **C** — IBM's Information Management System (IMS) is the classic hierarchical DBMS from the 1960s. Oracle and PostgreSQL are relational, and MongoDB is a document (NoSQL) database.
48. **D** — Operators and maintenance personnel (system administration staff) run and maintain the hardware and software environment. They are workers behind the scene. Designing tables, coding transactions and granting privileges belong to the designer, programmer and DBA.
49. **A** — Silberschatz's example is a clerk who needs a list of students in one postcode: if no program exists for it, someone must write one. A DBMS solves this with ad-hoc query languages such as SQL.
50. **B** — Elmasri-Navathe list persistent storage for program objects as an advantage, mainly of object and object-relational DBMSs. Data that outlives the program that created it is persistent.
51. **C** — Sophisticated end users, such as engineers, scientists and business analysts, know the DBMS facilities well and meet complex requirements with them. The teller is naive, the manager is casual and the shopkeeper is standalone.
52. **D** — A trigger is a rule that fires automatically when a specified event, such as an UPDATE on a table, occurs. Elmasri-Navathe list rules and triggers as an advantage. Views, indexes and buffering do not react to events.
53. **A** — Data abstraction means hiding details of how data is stored and presenting a conceptual view through a data model such as the relational model. Compression is a storage technique, and row restriction is security.
54. **B** — Canned transactions are standard types of queries and updates that have been carefully programmed and tested. Naive (parametric) users such as bank tellers invoke them through a fixed screen.
55. **C** — Elmasri-Navathe note that after the database is up and running, developing a new application, such as a new report, usually takes much less time than with files. The DBMS already provides definition, storage, querying and integrity.
56. **D** — Silberschatz lists schema definition, storage structure and access-method definition, schema and physical-organisation modification, granting authorisation and routine maintenance. Users write their own ad-hoc queries.
57. **A** — Integrity problems arise because rules such as 'balance must not fall below zero' are coded into each program. Adding or changing a rule means editing many programs. Simultaneous access, scattered formats and half-done operations are other problems on the list.

58. **B** — Elmasri-Navathe say a DBMS provides storage structures such as indexes to speed up disk search, and buffering or caching to keep frequently used data in memory. The query optimiser then chooses an efficient plan.
59. **C** — Sharing lets multiple users and programs access the database simultaneously. The DBMS uses concurrency control so that simultaneous access stays correct. Copying the database to each user would create redundancy.
60. **D** — Elmasri-Navathe say a database represents part of the miniworld, is a logically coherent collection with inherent meaning, and is designed for a specific purpose and group of users. A random assortment of data is not a database, and no particular model or machine is required.
61. **A** — Elmasri-Navathe note that central control lets the DBA define and enforce standards, such as names and formats of data elements and terminology. This eases communication between departments and projects.
62. **B** — Elmasri-Navathe list stringent real-time requirements as a case where the DBMS's general-purpose overhead may be unacceptable. A DBMS can store numbers and serve many users; the issue is the processing overhead.
63. **C** — Casual users, often middle or senior managers, access the database occasionally and need different information each time. Sophisticated users, such as analysts and engineers, learn most DBMS facilities. Canned transactions belong to naive users and personal databases to standalone users.
64. **D** — Both are self-describing, but differently. A relational DBMS stores the schema once in its catalog, separate from the data. In XML and many NoSQL stores, each record carries the names of its data items, so the description travels with the data.
65. **A** — Someone who knows the DBMS well and uses it directly for complex analysis is a sophisticated end user. Naive users only run canned transactions, standalone users keep personal databases, and designers plan the structure.
66. **B** — The transfer must happen completely or not at all. Leaving it half done is an atomicity problem. A DBMS solves it with transactions that are rolled back after a failure.
67. **C** — When every new information need requires a new program, the file system has difficulty in accessing data. A DBMS lets the registrar's request be written as one ad-hoc SQL query.
68. **D** — Elmasri-Navathe's backup and recovery subsystem lets the database be restored after hardware or software failures. A nightly backup plus recovery limits the loss. Views, triggers and the optimiser do not protect against disk failure.
69. **A** — ROLLBACK undoes every change made since BEGIN, so the debit of 1,500 is cancelled and the balance is back to 5,000. This is how a DBMS gives atomicity.
70. **B** — The view selects only rows where dept is 'Exams', and two of the three employees are in Exams. The view hides salary, but COUNT(*) does not need it. This is one stored table serving a different view to one user group.
71. **B** — With three copies, each change must be made in three files: 400 × 3 = 1,200 updates. An integrated database stores the number once, so 400 updates are enough and no copy can become stale.
72. **B** — A is now 5,000 and B is still 2,000, so the total is 7,000 instead of 10,000. The 3,000 left one account and never reached the other. A DBMS transaction would roll back the debit during recovery.
73. **A** — student was created with 3 columns, and ALTER TABLE added email, so the catalog now records 4. The query reads metadata, not data, which is why it works even though the table has no rows. This is the self-describing nature of a DBMS.
74. **C** — Silberschatz lists schema and physical-organisation modification for changes that reflect new needs or improve performance. Initial schema definition happens when the database is first created, and routine maintenance covers backups, disk space and job monitoring.
75. **D** — In object and object-relational databases, an operation has an interface and an implementation. Changing the implementation without changing the interface leaves calling programs unaffected: program-operation independence.
76. **A** — Elmasri-Navathe list the ability to represent a variety of complex relationships, such as many-to-many links, and to retrieve and update related data efficiently. The scenario is about relationships, not interfaces or security.
77. **B** — Tool developers design and implement tools: optional packages for database design, performance monitoring, graphical interfaces and so on. DBMS implementers build the DBMS modules themselves, and the DBA uses such tools.

78. **A** — The trigger fires once for each updated row. The first UPDATE changes one row and logs 1 entry. The second UPDATE has no WHERE, so it changes both rows and logs 2 more. That makes 3.
79. **C** — Elmasri-Navathe note that as soon as one user's update is applied, all other users can see it, because everyone shares one database. Separate files would each need their own update.
80. **D** — Elmasri-Navathe describe economies of scale: consolidating data and applications reduces overlapping work and lets the organisation invest in more powerful hardware instead of many low-end systems.
81. **A** — The UPDATE would make pen 5 and ink -2. Ink breaks the CHECK, so the whole statement fails and is undone, including the change to pen. The sum stays 10 + 3 = 13. A DBMS makes each statement atomic.
82. **B** — Every agent sold a ticket, so 5 were sold. Each wrote 3 - 1 = 2, so the file shows 2 seats left even though the flight is overbooked by 2. Four updates were lost: a concurrent-access anomaly.
83. **B** — Files: 4 × 200 × 2,500 = 2,000,000 bytes. Integrated: 200 × 2,500 = 500,000 bytes for profiles plus 4 × 8 × 2,500 = 80,000 bytes of references, total 580,000. Saving: 2,000,000 − 580,000 = 1,420,000 bytes.
84. **A** — The first transaction commits: account 1 becomes 300 and account 2 becomes 1,200. In the second, the credit makes account 2 1,900, but the debit would make account 1 -400 and is rejected. ROLLBACK undoes the credit as well, leaving 300 and 1,200.
85. **A** — (a) refers to dept 30, which does not exist. (b) would delete dept 20 while employee 2 still refers to it. (c) inserts a NULL foreign key, which is allowed, and (d) points employee 3 to dept 20, which exists. So (a) and (b) are rejected.
86. **A** — The DBMS's recovery subsystem relies on backups and logs to restore the database after a failure such as a disk crash. Moving to a DBMS gives better tools for backup, not freedom from it.
87. **B** — Only one program was running, so this is not a concurrency problem. The disagreement comes from a crash in the middle of a two-step operation. A transaction makes the two writes atomic, and recovery undoes the first if the second never happens.
88. **C** — In Silberschatz's file-system list, data isolation is about scattered files in different formats that make new programs hard to write. ACID isolation is about concurrent transactions. The shared word is a common exam trap.
89. **D** — Elmasri-Navathe stress that a DBMS has a high initial cost for hardware, software and training, plus overhead. For simple applications, files may be cheaper. The other three are on their list of advantages.
90. **C** — Controlled redundancy means duplication that is deliberate and kept consistent by the DBMS. A foreign key repeats dept_id by design, and the DBMS enforces that it refers to a real department. Repeated free-text names have no such check.
91. **D** — A view stores only its definition. Its rows come from the base table when queried, so every office sees the same current data. Separate files are separate copies, which leads to redundancy and inconsistency.
92. **C** — An integrity problem is data that breaks a consistency rule, such as an impossible CGPA. The other options are a security problem, a concurrent-access anomaly and an atomicity problem.
93. **D** — A view stores its query, not data. When hostel_view is queried, it reads the current student table, where roll 1's city is now Multan. This is how one stored copy serves many views consistently.
94. **C** — T1 overwrote T2's write using a value read before T2 wrote, so T2's withdrawal disappeared. The correct result of both is 4,000 + 1,000 − 500 = 4,500. Concurrency control would make one transaction wait.
95. **D** — Private copies are uncontrolled redundancy. When a customer's details change at 10 a.m., each department works with stale data until the next night. Using the shared database directly, through views, would avoid this.
96. **C** — In autocommit mode each statement is its own transaction. The credit commits at once; the failed debit is undone on its own. Without BEGIN ... COMMIT around both, the DBMS cannot treat the transfer as one atomic unit.
97. **D** — Elmasri-Navathe say privileged commands for creating accounts, setting parameters, granting authorisation and changing storage structures are interfaces for the DBA staff. Naive users get simple forms and keys.
98. **C** — Elmasri-Navathe suggest files may suit simple stable applications, real-time systems, embedded systems with little storage and cases with no multiuser access. Many concurrent updaters is exactly where a DBMS's concurrency control is needed.

99. **B** — Only the 15 students updated by all three offices are fully consistent. So 60 − 15 = 45 students have at least one stale copy. 65 counts stale copies (20 in hostel plus 45 in exam), not students.
100. **D** — Change (1) alters how data is stored without affecting programs, which is program-data independence. Change (2) alters an operation's implementation but not its interface, which is program-operation independence.
101. **A** — The internal schema describes how the database is physically stored, using a physical data model: record layouts, file organisation and access paths. The conceptual level hides these details, and the external (view) level describes user views.
102. **C** — In Elmasri-Navathe the conceptual schema describes the whole database for a community of users. It hides physical storage and concentrates on entities, types, relationships, user operations and constraints. It is usually written in a representational model such as the relational model.
103. **B** — The external level includes a number of external schemas, or user views. Each describes the part of the database a particular user group is interested in and hides the rest. There is only one conceptual and one internal schema.
104. **C** — The three-schema architecture is often called the ANSI/SPARC architecture, after the American National Standards Institute's Standards Planning and Requirements Committee study group of the 1970s. OSI is the networking reference model, and IEEE 802 covers LAN standards.
105. **C** — Silberschatz names the levels physical, logical and view. They line up with Elmasri-Navathe's internal, conceptual and external levels. The logical level describes what data is stored and the relationships among it.
106. **C** — Physical data independence means storage-level changes, such as new indexes, a different file organisation or new devices, do not require changes to the conceptual schema, and so do not affect external schemas or programs.
107. **A** — Logical data independence lets the conceptual schema grow or be restructured, for example by adding an attribute or splitting a table, while external schemas and the programs written against them stay unchanged. Only the view definitions and mappings change.
108. **D** — Each level is only a description; the real data exists only at the physical level. The DBMS transforms a request on an external schema into one on the conceptual schema and then into one on the internal schema. These transformations are called mappings.
109. **C** — The schema is the description of the database, specified at design time and rarely changed. The data at a given moment is the database state, current set of occurrences, instance or snapshot. Every insert, update or delete produces a new state.
110. **C** — The DBMS stores the schema descriptions and the mappings among them in its catalog, so it can refer to them when it processes requests. The log records changes for recovery, and the buffer pool caches disk pages.
111. **A** — Elmasri-Navathe separate the DDL (conceptual schema), SDL (internal schema) and VDL (user views). They note that most current relational DBMSs have no separate SDL: physical details are set through commands and parameters, and SQL's DDL covers views too.
112. **B** — A high-level, nonprocedural DML states what data to retrieve, and it can return many records with one statement (set-at-a-time). A low-level, procedural DML, as in older network and hierarchical systems, retrieves one record at a time inside program loops.
113. **B** — The buffer manager moves pages between disk and main memory and decides which ones to cache; it lets the database be much larger than memory. The file manager allocates disk space and data structures, and the transaction manager handles consistency and concurrency.
114. **B** — A query can be evaluated in many equivalent ways. The optimiser, part of the query processor (Silberschatz places it with the DML compiler), estimates their costs and chooses the cheapest. The query evaluation engine then executes that plan.
115. **D** — In a three-tier architecture the client is only a front end. The application server holds the business rules and talks to the database server. In a two-tier architecture the client application itself contains the logic and connects to the database directly, for example through ODBC or JDBC.
116. **D** — Elmasri-Navathe and Silberschatz both say logical data independence is harder. Programs are written against the logical structure (table and column names), so restructuring it is more likely to affect them. Storage changes are hidden more easily behind the conceptual schema.

117. **C** — An index is an access path in the internal schema. Adding one changes the internal schema and the conceptual/internal mapping but not the conceptual schema, so programs are untouched: physical data independence.
118. **C** — Adding an attribute changes the conceptual schema. Because the external schemas do not refer to the new attribute, they and their programs are unaffected. Elmasri-Navathe give exactly this kind of change as an example of logical data independence.
119. **D** — Storage devices and file organisations are internal-schema details. The relation's attributes and constraints are untouched, so the conceptual schema is unchanged and physical data independence hides the change from programs.
120. **B** — The conceptual schema changed, so the external schemas must be mapped onto the new structure. Redefining the views changes the external/conceptual mapping while the external schemas stay the same, which is logical data independence in action.
121. **D** — An index is a physical access path. It can change how the DBMS finds rows (SQLite's EXPLAIN QUERY PLAN shows idx_score being used afterwards), but never which rows a query returns. Both runs return 2 and 4 in roll order, as ORDER BY requires.
122. **A** — The view is an external schema defined as SELECT roll, name. Adding email to the base table changes the conceptual schema, but the view's definition still names only roll and name, so it returns the same two columns.
123. **D** — Each external schema needs its own mapping to the single conceptual schema, giving 5. There is one conceptual schema and one internal schema, so there is one conceptual/internal mapping.
124. **D** — A view describes the part of the database one user group sees, so it belongs to the external level. In Elmasri-Navathe's terms it is written in a view definition language; in SQL that is part of the DDL. It stores no new data, so the database state is unchanged.
125. **A** — The client application holds the user interface and logic and talks directly to the database server through an API such as ODBC or JDBC. With no application server in between, this is the classic two-tier architecture.
126. **C** — In a three-tier design the browser only sends HTTP requests. The application server runs the business logic and issues SQL to the database server, then formats the results for the client. This keeps database credentials off the client.
127. **B** — INSERT, UPDATE and DELETE are DML: they produce a new database state while the schema stays the same. ALTER TABLE is DDL and changes the schema (schema evolution). CREATE TABLE also changes the schema, but it is not among the options.
128. **B** — Silberschatz describes three steps. The query is parsed and translated into an internal form such as relational algebra. The optimiser then chooses an evaluation plan, and finally the evaluation engine executes it. Optimisation must come before execution.
129. **B** — Views and mappings can hide where and how data is stored, but they cannot supply data that no longer exists. Removing an attribute a program uses forces a change to that program. The other three are standard examples of physical or logical data independence.
130. **D** — Browsers never connect to the database; only application servers do. 4 servers × 25 pooled connections = 100. In a two-tier design each of the 200 clients would hold its own connection.
131. **A** — SELECT * expands to every current column, so after ADD COLUMN program A receives roll, name, cgpa and email and its 3-value unpacking fails. B lists its columns, so its result shape is unchanged. Naming columns (or using views) preserves logical data independence.
132. **A** — Splitting emp_old into emp and dept changes the conceptual schema. Redefining emp_view as a join keeps the external schema identical, so programs using the view see the same rows: logical data independence through an updated external/conceptual mapping.
133. **A** — Indexes are access paths: they change speed, never results. The other statements are true. Elmasri-Navathe note that the upper levels are only descriptions, that external schemas hide parts of the database, and that mappings cost time, which is why few DBMSs implement the architecture fully.
134. **A** — Mapping requests through external, conceptual and internal levels costs time. Most DBMSs therefore support the architecture only partly, and some older ones mix physical details into the conceptual schema. Views still provide an external level in SQL systems.
135. **B** — The request is first rewritten from the external schema onto the conceptual schema, then from the conceptual schema onto the internal schema to reach stored records. Results travel back the same way, reformatted to match the user's view.

136. **B** — Indexes and block compression are internal-schema changes, hidden by physical data independence. Adding an attribute and merging tables change the conceptual schema, so hiding them from programs needs logical data independence.
137. **D** — An application server centralises the business logic, holds the only database credentials and pools connections for thousands of clients. Two-tier designs put credentials and rules on every client device.
138. **D** — Elmasri-Navathe: defining a new database specifies only its schema, and the database is then in the empty state with no data. The initial state appears when data is first loaded, and every later update produces another state.
139. **A** — Silberschatz's storage manager includes an authorisation and integrity manager, which tests integrity constraints and checks users' authority. The buffer and file managers move and allocate data. The DDL interpreter records the constraint in the catalog when it is defined but does not check each update.
140. **C** — In two-tier, every client connects to the database directly, so every client's settings change. In three-tier, only the application servers connect to the database, so only those 2 change; the clients keep talking to the same application servers.
141. **A** — The schema is also called the intension, and the database state is called the extension. Snapshot and instance are other names for the state, not the schema.
142. **B** — A schema diagram displays the schema: record types, their data items and sometimes constraints. It does not show the database state (the rows), and it is not a storage map.
143. **C** — Each object within the schema, such as a record type, is called a schema construct. The state is the data at one moment, and a mapping links two levels of the architecture.
144. **D** — The VDL defines external schemas (views) and their mappings to the conceptual schema. The SDL defines the internal schema, and the DML retrieves and updates data. In SQL, CREATE VIEW plays the VDL role.
145. **A** — The general-purpose language is the host language, and the embedded DML is the data sublanguage. SDL and VDL are schema-definition languages.
146. **B** — A low-level or procedural DML is record-at-a-time: the program fetches and processes individual records, typically in a loop. A high-level DML such as SQL is set-at-a-time and declarative.
147. **C** — The file manager handles disk-space allocation and the on-disk data structures. The buffer manager moves data between disk and memory, and the transaction manager handles concurrency and recovery.
148. **D** — The transaction manager handles recovery from failures and concurrency control. The file manager deals with disk space, the DDL interpreter processes schema definitions, and the authorisation and integrity manager tests constraints and privileges.
149. **A** — Elmasri-Navathe describe a loading utility that loads existing data files into the database, converting from the source format to the database's storage format. The other options are not used for bulk loading.
150. **B** — Classification by the number of sites gives centralised DBMSs (data at one site) and distributed DBMSs (data and software at many sites). Single-user and special-purpose are other classification criteria.
151. **C** — ODBC is a standard application programming interface that lets client programs access a DBMS through its driver. JDBC plays the same role for Java. HTTP, SMTP and FTP are web, mail and file-transfer protocols.
152. **D** — Elmasri-Navathe group data models into high-level conceptual models (such as ER), representational models (such as relational) and low-level physical models. Conceptual models use entities, attributes and relationships.
153. **A** — Representational data models sit between conceptual and physical models. Commercial DBMSs implement them directly. Storage details belong to physical models.
154. **B** — The schema is not supposed to change often, but requirements change. Elmasri-Navathe call this schema evolution. A state transition is a change to the data, and a mapping links levels.
155. **C** — The stored data manager is a higher-level module that controls access to DBMS information on disk, using basic OS services to transfer data. The compilers translate statements; they do not manage disk access.
156. **D** — Elmasri-Navathe list file reorganisation among DBMS utilities. Loading brings data in, backup copies the database, and the DDL compiler processes schema definitions.
157. **A** — The runtime database processor executes compiled queries and canned transactions. It works with the system catalog and the stored data manager. The precompiler extracts SQL, and the DDL compiler records schemas in the catalog.

158. **B** — A thin client, such as a web browser, handles presentation while application and database servers do the processing. A fat client carries much of the application logic itself.
159. **C** — Silberschatz lists data files (the data), the data dictionary (metadata, including the schema) and indices (fast access). The buffer pool is in main memory, not a disk structure.
160. **D** — In a federated DBMS, the participating databases are autonomous and may use different DBMS software. They are loosely coupled, with a high degree of local autonomy.
161. **A** — The DDL compiler processes schema definitions in the DDL and stores the descriptions in the system catalog. Executing queries, optimising them and writing pages are other modules' jobs.
162. **B** — Elmasri-Navathe note that most relational DBMSs have no specific SDL. The internal schema is set through storage-related parameters and statements such as CREATE INDEX, while SQL combines DDL, VDL and DML.
163. **C** — Elmasri-Navathe say the main criterion is the data model, such as relational, object, document or graph. Number of users, number of sites, cost and general versus special purpose are further criteria.
164. **D** — The distinction is about the DBMS software at each site. Heterogeneous systems need middleware or gateways to translate between products. Replication of rows is a separate design choice.
165. **A** — Forms-based interfaces show a form that the user fills in to insert data or to state a query. They suit naive users doing canned transactions. Natural language, DBA and embedded SQL interfaces serve other users.
166. **B** — A natural language interface accepts requests in a language such as English and interprets them using the schema and a dictionary of words. Forms, parametric screens and menus do not accept free-text requests.
167. **C** — Parametric users such as tellers repeat a small set of operations. Their interface abbreviates each operation into few keystrokes, often function keys. The DBA interface is for privileged commands.
168. **D** — A view stores its definition, not rows. When it is queried, it reads the current course table, where two codes start with CS. The external schema is defined over the conceptual schema, not over a copy of the data.
169. **A** — In three-tier architecture, the client tier presents the interface, the middle tier runs business logic, and the database server stores and manages data. The browser displays pages and sends requests.
170. **B** — Before the index, SQLite must scan every row. After it, the plan searches idx_city for city = 'Lahore'. The access path is part of the internal level; the query text and its result rows stay the same.
171. **A** — INSERT, UPDATE and DELETE (2, 4 and 6) change the data. CREATE TABLE and ALTER TABLE (1 and 3) change the conceptual schema, and CREATE INDEX (5) changes the internal schema. DELETE is a state change like the others.
172. **C** — Each view has its own external/conceptual mapping, so 2 new views add 2 mappings. Changing file organisation alters the internal schema, so the single conceptual/internal mapping is updated. The 4 old views are untouched.
173. **D** — The redefined view maps mobile back to phone, so the external schema the program uses is unchanged. Only the external/conceptual mapping was updated. This is logical data independence.
174. **A** — When the client program holds the business rules and talks straight to the DBMS, the design is two-tier with fat clients. Changing a rule means updating all 50 PCs. A three-tier design moves the logic to an application server.
175. **B** — The DDL interpreter interprets DDL statements and records the definitions in the data dictionary. The evaluation engine runs query plans, the buffer manager caches pages, and the transaction manager handles concurrency and recovery.
176. **C** — Elmasri-Navathe's precompiler extracts DML commands from a host-language program. They go to the DML compiler, and the rest goes to the host-language compiler. The two parts are then linked.
177. **D** — Elmasri-Navathe describe a performance-monitoring utility that monitors database usage and gives statistics to the DBA. The DBA uses them to decide on reorganisation or new indexes.
178. **A** — The data is spread across sites, so the DBMS is distributed. Every site uses the same DBMS software, so it is homogeneous. Federated systems link autonomous, often different, pre-existing databases.
179. **B** — Requests go down from external to internal. Results come back up: the stored data is extracted, reformatted to the conceptual schema, then to the user's external view. Elmasri-Navathe describe both directions as mappings.
180. **C** — CREATE TABLE defines a relation in the conceptual schema. CREATE INDEX defines an access path, part of the internal schema. CREATE VIEW defines a user view, part of an external schema.

181. **D** — SQLite checks dependent views and refuses the DROP COLUMN. The general point is that data independence has limits: removing an attribute that an external schema uses cannot be hidden by any mapping.
182. **A** — The conceptual schema changes, so the views that refer to the moved columns need new external/conceptual mappings: 4. The new relation is stored in a new file, so the conceptual/internal mapping changes as well. The other 11 views are untouched.
183. **B** — Two-tier: 200 × 10 × 3 = 6,000 SQL round trips over the WAN. Three-tier: 200 × 10 = 2,000 requests cross the WAN, and the 6,000 SQL calls stay inside the data centre. This is one reason three-tier designs suit remote clients.
184. **B** — The DML compiler translates the query into a plan, with the optimiser picking the cheapest one. The evaluation engine runs it and asks the buffer manager for pages. On a miss, the buffer manager has the file manager read the block from disk.
185. **A** — Roll 1 (born 10 March 2004) is 21 on 1 January 2026, and roll 2 (born 30 November 2006) is 19. Only roll 1 meets age >= 20. The external schema shows a derived attribute that the conceptual schema does not store.
186. **C** — There is one internal schema, describing how the whole database is stored. User groups differ at the external level, where each can have its own view.
187. **D** — Physical data independence holds only for programs that work through the conceptual or external levels. A program that stores physical addresses depends on the internal schema, so internal changes reach it.
188. **A** — Every state should be a valid state that satisfies the schema's structure and constraints. The DBMS enforces this as data changes. The schema changes rarely, and an empty database is in the empty state.
189. **B** — Views sit at the external level and let the conceptual schema change behind them. Physical data independence exists whether or not views are used, because programs written against the conceptual schema are shielded from storage changes.
190. **C** — A record-at-a-time program decides how to reach the data, so it cannot benefit from a new index unless it is rewritten. A declarative SQL query states only what is wanted, and the optimiser picks the access path.
191. **D** — With logic on the application server, clients become thin and changes are made in one place. The application server, not the clients, sends SQL to the database.
192. **B** — An index is an internal-level access path, so only the conceptual/internal mapping is affected. Views and their external/conceptual mappings do not change when an index is added.
193. **A** — v2 is defined over v1, which is defined over s. After the update, roll 3 has 3.1, so v1 holds rolls 1, 2 and 3. v2 keeps those below 3.5: rolls 1 (3.2) and 3 (3.1). Views over views still read the current state.
194. **C** — SQL results are sets unless ORDER BY is given. The covering index on (city, name) returns rows in name order, so the order changed but the rows did not. A program that needs an order must say ORDER BY.
195. **D** — Physical independence comes from the conceptual/internal mapping, which still exists. Without views, there is no external/conceptual mapping to redefine, so restructuring tables reaches programs directly.
196. **B** — Data independence promises correct results without program changes, not equal speed. Hash files suit equality searches but not ranges, so the plan must scan. The DBA should keep or add an ordered index.
197. **C** — An aggregate row has no single base row behind it, so the DBMS cannot translate the update. The external/conceptual mapping works for queries but is not always invertible for updates.
198. **D** — With many advisors per student, the view must return several rows per roll or drop advisors. Either way the program's one-advisor assumption breaks. Adding attributes and lossless splits can be hidden by redefining the view, and moving disks is physical.
199. **C** — ER is a high-level conceptual model. Relational table definitions are a representational (implementation) model. Index structures and record layouts belong to the physical model.
200. **D** — Cost-based optimisers estimate plan costs from catalog statistics, such as table sizes and value distributions. Without them, the estimates are poor and a slow plan may be chosen. The results stay correct.
201. **D** — A data model provides the concepts used to describe a database's structure: data types, relationships and constraints. Most models also include basic operations for retrieval and update. The data at a moment is the database state, not the model.
202. **C** — Elmasri-Navathe group data models as high-level or conceptual (such as the ER model), representational or implementation (relational, and the older network and hierarchical models) and low-level or physical (record formats, access paths). The relational model is the main representational model.

203. **D** — A double oval marks a multivalued attribute such as Colours or PhoneNumbers. A dashed oval is derived, an underlined name is a key, and a weak entity type is a double rectangle.
204. **C** — A derived attribute's value can be computed from other attributes, so it is drawn with a dashed oval. BirthDate is the stored attribute it is derived from.
205. **B** — A weak entity type, which has no key of its own, is drawn as a double rectangle. The identifying relationship linking it to its owner entity type is a double diamond.
206. **C** — A double line shows total participation, also called existence dependency: every entity in that entity type must be related to at least one entity through the relationship. A single line means partial participation.
207. **C** — The key attribute, whose values are distinct for every entity, has its name underlined. A weak entity's partial key is underlined with a dashed line instead.
208. **D** — Formal terms: a relation is a table, a tuple is a row, an attribute is a column header and a domain is the set of atomic values an attribute may take. The relation schema is the relation's name with its list of attributes.
209. **C** — Degree or arity is the number of attributes. The number of tuples is the relation's cardinality. For example, STUDENT(Roll, Name, Cgpa) has degree 3 whatever its row count.
210. **D** — The hierarchical model arranges record types in trees, so each child has exactly one parent. The network (CODASYL) model allows graphs, with record types linked through owner-member sets. Both were in use before relational systems.
211. **A** — A weak entity type has no key of its own. Each weak entity is identified by being related to an owner entity through the identifying relationship, plus its partial key (discriminator), such as a dependent's name within one employee.
212. **B** — Elmasri-Navathe call the cardinality ratio (1:1, 1:N, M:N) and the participation constraint (total or partial) the structural constraints of a relationship type. The (min, max) notation expresses both at once.
213. **A** — (min, max) gives, for each entity, the least and most relationship instances it takes part in. min = 0 means it may take part in none (partial participation); min ≥ 1 means total participation.
214. **A** — A relation is a set of tuples, so its tuples have no inherent order and no two are identical. In the flat relational model each value is atomic. Order appears only when a query asks for it with ORDER BY.
215. **B** — When the same entity type plays two roles in one relationship type, the relationship is recursive (self-referencing). Role names make each participation clear, such as supervisor and supervisee in Elmasri-Navathe's SUPERVISION.
216. **B** — Silberschatz describes the semi-structured model, used by XML and JSON, as one where items of the same type may have different sets of attributes. The flat relational model needs one fixed schema for every row.
217. **D** — An attribute that can hold a set of values for one entity is multivalued and drawn as a double oval. When mapped to relations, it becomes a separate relation of (Roll, Phone).
218. **C** — A composite attribute can be divided into smaller parts with their own meanings. When mapped to a relation, only the simple component attributes are stored as columns.
219. **D** — An M:N relationship becomes its own relation holding both participants' keys as foreign keys, plus any relationship attributes. The combination (Roll, Code) identifies each enrolment. Grade depends on that pair, so it is not part of the key.
220. **A** — For 1:N, put the key of the '1' side into the relation on the 'N' side. Each employee has one department, so EMPLOYEE gets Dno. Putting employee keys into DEPARTMENT would need many values in one field.
221. **A** — A weak entity's relation includes the owner's key as a foreign key. Its primary key combines that foreign key with the partial key: (Essn, DependentName). Two employees can each have a dependent called Ali, so the name alone is not unique.
222. **C** — A multivalued attribute gets its own relation holding the owner's key and the attribute, with both as the key. A list in one column breaks the atomic-value rule, and fixed repeating columns cap the number of values.
223. **C** — 4 entity relations, plus 1 for the M:N ENROLS, plus 1 for the multivalued Phones, gives 6. The three 1:N relationships become foreign keys in COURSE and TEACHER, so they need no relations of their own.
224. **D** — Each (min, max) pair sits next to the entity it constrains. Each department takes part in at least 4 WORKS_FOR instances (at least 4 employees), and each employee in exactly one. Departments cannot share employees, so at least 12 × 4 = 48 are needed.
225. **C** — 1:N means each employee is related to at most one department, so there are at most 40 instances, one per employee. 200 (5 × 40) would be the maximum only for an M:N relationship.

226. **A** — Elmasri-Navathe advise putting the foreign key in the relation with total participation. Every department has a manager, so Mgr_ssn is never NULL. In EMPLOYEE, most rows would hold NULL because most employees manage nothing.
227. **B** — Degree counts attributes: code, title, credits and dept make 4. Cardinality counts tuples: six rows were inserted. The primary key is one of the 4 attributes, not an extra one.
228. **B** — An n-ary relationship becomes its own relation containing all participants' keys as foreign keys. When every side is 'many', the key is the combination of all three. Quantity is a relationship attribute, not part of the key.
229. **B** — 4 entity relations (including the weak DEPENDENT) + 1 for M:N WORKS_ON + 2 for the multivalued Skills and Locations = 7. The 1:N, 1:1, recursive and identifying relationships become foreign keys. Composite Name flattens into columns and derived Age is not stored.
230. **D** — Each section needs at least 5 enrolments: 10 × 5 = 50. The upper limit is the tighter of 10 sections × 40 = 400 and 70 students × 6 = 420, so 400. Students' min of 0 adds no lower bound.
231. **A** — (min, max) beside an entity type counts the relationship instances each of its entities takes part in. An employee takes part in exactly one, so (1,1) goes beside EMPLOYEE. This is the reverse of Chen's placement, where the N sits beside EMPLOYEE.
232. **D** — With total participation on both sides of a 1:1, Elmasri-Navathe's merged-relation option combines the two entity types and the relationship into one relation, because every person has exactly one record and vice versa. The foreign-key approach would use 2.
233. **A** — E1 and E2 each need a relation. R1 (1:N) becomes a foreign key a1 in E2. R2 (M:N) needs its own relation with key (a1, b1). Total: 3.
234. **A** — Hours describes an employee-project pair, not an employee alone. For 1:1 or 1:N relationships an attribute can migrate to one side, but for M:N it must stay with the relationship and becomes a column of the WORKS_ON relation.
235. **D** — Joining SP, SJ and PJ returns the 3 real tuples plus (s1, p1, j1), because s1 supplies p1, s1 supplies j1 and p1 goes to j1, though never all three together. A ternary relationship generally cannot be replaced by three binaries without losing information.
236. **A** — An SQL table is a multiset, so the duplicate ('Ali', 'W1') is stored twice: COUNT(*) is 3, and the DISTINCT count is 2. A formal relation is a set of distinct tuples, so its cardinality is 2. Declaring a key makes the SQL table behave like a relation.
237. **B** — In a hierarchy each child record has exactly one parent, so a course shared by many students must be repeated under each of them, or linked with virtual (pointer) records. The network model's graphs and the relational model's foreign keys handle M:N directly.
238. **B** — PaymentNo is unique only within one loan, so PAYMENT cannot be identified alone. It is a weak entity type: its identity is (LoanNo, PaymentNo) through the identifying relationship, and every payment must belong to a loan (total participation).
239. **B** — In ER modelling, cardinality ratio (1:1, 1:N, M:N) limits how many entities can be related. In the relational model, a relation's cardinality is its number of tuples. Read MCQs carefully to see which sense is meant.
240. **C** — Each department needs at least 4 employees and no employee can be shared. floor(30 / 4) = 7 departments use 28 employees, and the other 2 join any of them. An 8th department would need 32 employees.
241. **D** — The schema (intension) is specified at design time and changes rarely. The state, or instance (extension), is the actual data at a given moment and changes with every insert, delete or update.
242. **D** — Elmasri-Navathe's physical data models describe storage details such as record formats, orderings and access paths like indexes. Conceptual models such as ER describe how users see data, and representational models such as the relational model sit between the two.
243. **B** — The network model stores record types linked by set types. Each set type has one owner record type and one member record type and represents a 1:N relationship. Foreign keys belong to the relational model and parent-child trees to the hierarchical model.
244. **B** — The degree of a relationship type is the number of participating entity types. Three entity types take part in SUPPLY, so its degree is 3 and it is ternary. The number of instances is not the degree.
245. **B** — In this notation, braces { } enclose a multivalued attribute and parentheses ( ) list the components of a composite attribute. The two can nest, so a complex attribute can be both multivalued and composite.

246. **A** — An entity type describes the attributes shared by its entities. The entity set is the collection of entities of that type in the database at a particular time, which is the type's extension.
247. **D** — Each N-side entity takes part in at most one relationship instance, so a relationship attribute has one value per N-side entity and can be stored there. A 1-side entity relates to many instances, so one value there could not hold all of them.
248. **D** — A weak entity is identified by its owner's key plus its partial key. Without an owner it has no identity, so each weak entity must take part in the identifying relationship. This is called existence dependency.
249. **A** — In Elmasri-Navathe's EER notation, 'd' marks the disjointness constraint and 'o' marks overlapping. Totality is a separate constraint, shown by a double line from the superclass to the circle.
250. **D** — Specialization starts from one entity type, such as EMPLOYEE, and defines subclasses such as ENGINEER. Generalization starts from several entity types, such as CAR and TRUCK, and abstracts their common features into a superclass, VEHICLE. Both give the same superclass/subclass structure.
251. **D** — A subclass entity is the same real-world entity as its superclass entity, in a specific role. So it inherits all the superclass's attributes and the relationships the superclass takes part in, and it may add its own.
252. **B** — A category (union type) is a subclass of the union of several different superclasses. An OWNER is one person, one bank or one company. Specializing OWNER would wrongly make every person, bank and company an owner and would assume one shared key.
253. **A** — In the semi-structured model the names that describe the data travel with it, for example "name": "Ali" in JSON. A separate schema is optional, so documents of the same kind can have different fields.
254. **D** — R(A1, ..., An) is a relation schema. It names the relation and lists its attributes, so its degree is n. A relation state r(R) is the set of tuples stored for that schema at a moment.
255. **C** — Codd introduced the relational model in his 1970 paper 'A Relational Model of Data for Large Shared Data Banks'. Chen proposed the ER model in 1976, and Bachman led early network-model work (IDS).
256. **D** — A cardinality ratio gives the maximum number of relationship instances an entity can take part in. 1:1 means at most one on each side. Whether participation is required is a separate participation constraint.
257. **A** — The weak relation's key includes the owner's key as a foreign key. Since the weak entity is existence-dependent, deleting the owner should delete its weak entities, so CASCADE is the usual choice. SET NULL would put NULL into the primary key.
258. **C** — The three 1:1 approaches are: foreign key (put one key in the other relation), merged relation (combine both into one relation) and cross-reference or relationship relation (a third relation holding both keys).
259. **D** — Option 8A (multiple relations, superclass and subclasses) works for any specialization. The other options describe 8B (subclass relations only), 8C (single relation with one type attribute) and 8D (single relation with several type flags).
260. **D** — The flat relational model needs atomic values. So the mapping keeps the simple component attributes of a composite attribute as columns of the entity's relation. Only multivalued attributes need a separate relation.
261. **B** — A derived attribute's value comes from other data, so storing it would risk inconsistency. It is normally computed in a query or a view from the stored attribute.
262. **D** — A partial key gets a dashed underline to show it is unique only among the weak entities of one owner. A solid underline marks a full key, a double oval a multivalued attribute and a dashed oval a derived attribute.
263. **A** — A member record has at most one owner in each set type. So M:N is broken into two 1:N set types, STUDENT-to-LINK and COURSE-to-LINK, through a link record type. This is like the separate relation used for M:N in relational mapping.
264. **A** — Elmasri-Navathe define the value set (domain) of an attribute as the set of values it may take, such as integers from 16 to 70 for Age. It is a definition, not the values present at a moment.
265. **B** — NULL can mean 'not applicable', 'value exists but is missing' or 'not known whether it exists'. A single-family house has no apartment, so the attribute does not apply to it. NULL is never the number zero.
266. **D** — Elmasri-Navathe use exactly this CAR example. An entity type can have several keys, and each is underlined. Combining two independent keys would not be minimal.
267. **C** — A recursive 1:N relationship maps like any 1:N relationship. The N-side (the supervisee) gets a foreign key to the 1-side, which here is the same relation. This gives EMPLOYEE.SuperSsn → EMPLOYEE.Ssn.
268. **B** — Any M:N relationship becomes its own relation keyed by both participants' keys. Here both participants are COURSE, playing two roles. A single PrereqNo column could store only one prerequisite per course.

269. **B** — The relation keeps Ssn, Fname, Lname, Salary and BirthDate. Age is derived and not stored, and Skills goes to its own relation EMP_SKILLS(Ssn, Skill). So the degree is 5.
270. **B** — A weak entity's key is its owner's key plus its own partial key. ROOM's key is (Bno, RoomNo), so SEAT's key is (Bno, RoomNo, SeatNo). Elmasri-Navathe note that an owner may itself be weak, and keys are then built level by level.
271. **A** — There is one relation per entity type (3), one for the M:N WRITES (1) and one for the multivalued Phones (1). BOOK N:1 PUBLISHER becomes a foreign key in BOOK. Total: 5.
272. **C** — The three entity types, including the weak one, give 3. MENTORS, being M:N, adds 1 and Locations adds 1, for 5. MANAGES and WORKS_FOR become foreign keys, the identifying relationship is covered by DEPENDENT's key, and the composite Name becomes columns.
273. **A** — Each course is in exactly one TEACHES instance, so there are 25 instances. Each professor teaches 1 to 3 courses. The fewest professors is ceil(25/3) = 9, and the most is 25, with one course each.
274. **B** — From the student side the count is 0 to 80. From the club side it is 6×3 = 18 to 6×10 = 60. Both must hold, so the range is the overlap: 18 to 60.
275. **A** — In UML a multiplicity is written at the far end of the association. The 1 beside DEPARTMENT says how many departments one employee has, and 1..* beside EMPLOYEE says how many employees one department has. Elmasri-Navathe's (min, max) pairs are placed the opposite way.
276. **A** — An attribute that refers to another entity type is really a relationship. In the conceptual design, Elmasri-Navathe turn such attributes into relationship types, so the link is not stored twice.
277. **D** — Each (department, location) pair becomes one tuple: 1 + 2 + 3 = 6. The key is (Dnumber, Location), so Lahore can appear once for each department.
278. **B** — The key (essn, pno) allows one row per employee-project pair. The first two INSERTs add 4 rows. The last one repeats (1, 10) and is rejected by the key, even though hours differ. Verified in SQLite.
279. **C** — Option 8C keeps Ssn and Name, adds all three subclass attributes and adds one type attribute (JobType) that says which subclass a row belongs to: 2 + 3 + 1 = 6. Option 8D would use three Boolean flags instead, giving 8.
280. **C** — Option 8B creates one relation per subclass, each holding the superclass attributes plus its own, and no separate EMPLOYEE relation. That gives 3. It is safe only when the specialization is total, so every employee is in some subclass.
281. **C** — One passenger relates to many tickets, but each ticket relates to only one passenger. So PASSENGER : TICKET is 1:N.
282. **C** — Participation asks whether every entity must take part. Every ticket must (total, drawn with a double line), but a passenger may not (partial, a single line).
283. **A** — Each (min, max) pair sits beside the entity it describes and counts that entity's own participations. A passenger takes part 0 to N times and a ticket exactly once. Writing them the other way round copies the Chen 1:N label placement by mistake.
284. **C** — Four entity types, including the weak one, give 4. DEPOSITOR (M:N) adds 1 and Emails (multivalued) adds 1, for 6. ACCOUNT N:1 BRANCH becomes a foreign key in ACCOUNT, and the composite Address becomes columns.
285. **C** — PERSON and PASSPORT can merge into one relation because of total participation on both sides, giving 1. COUNTRY and AIRPORT add 2, with N:1 as a foreign key, and the ternary TRAVELS always needs its own relation, adding 1. Total: 4.
286. **C** — Once a concept has its own attributes and is shared by many students, it should be an entity type. Each university's data is then stored once, and Programmes can be a multivalued attribute of UNIVERSITY.
287. **A** — An entity type is weak only if it has no key of its own. With CNIC as a key, DEPENDENT is strong. Its dependence on EMPLOYEE can still be shown by total participation in a regular relationship.
288. **B** — The students allow 10×2 = 20 to 10×3 = 30 instances. The labs need exactly 7×5 = 35. No number lies in both ranges, so these constraints cannot all hold with these entity counts.
289. **B** — The binaries record only pairs. If Ali and Sara both taught CS201, in Fall and Spring, the pairs cannot show which of them taught in which term. The ternary stores the full triple.
290. **D** — In 8C every row carries all 18 subclass columns. An employee in no subclass has all 18 NULL, and one in a subclass still has 12 NULL. Elmasri-Navathe recommend 8C only when subclasses have few specific attributes.

291. **C** — 8B drops the superclass relation, so it only works when every superclass entity is in some subclass (total). With a partial specialization, a motorbike's VEHICLE attributes have no table. 8A or 8C would keep it.
292. **D** — Network and hierarchical programs are navigational: they move through stored links record by record. Relational queries are declarative. They relate tables through matching key values, and the DBMS chooses the access paths.
293. **A** — Merging suits a 1:1 with total participation on both sides. Here only 50 of 500 employees have a spot, so 450 merged rows have NULL parking attributes. A foreign key SpotNo in EMPLOYEE, or Ssn in PARKING_SPOT, avoids this.
294. **C** — Elmasri-Navathe note that if one participating entity type has max = 1, each of its entities appears in at most one SUPPLY instance. That foreign key then identifies the instance by itself and is the minimal key.
295. **C** — In a 1:N relationship each section has one teacher, so storing TeacherId in SECTION is enough. If a separate relation is used, SectionId already determines the row, so (TeacherId, SectionId) is not minimal.
296. **B** — A constraint must hold in every valid state. One counterexample, a department with no employees, rules out total participation for DEPARTMENT. A sample in which all employees have a department cannot prove that this must always be so.
297. **C** — Each section takes part exactly once, so the number of sections equals the number of TEACHES instances. The teachers allow 12×2 = 24 to 12×4 = 48 instances, so there are 24 to 48 sections.
298. **B** — Fixed repeating columns cap the number of values, leave unused ones NULL, and make queries such as 'who has this number?' check every column. One row per phone in a separate relation has none of these problems.
299. **A** — A stored derived value repeats information held elsewhere. If an enrolment changes and the column is not updated, the database holds two conflicting answers. Storing it may still be chosen for speed, but it must then be kept in step, for example with a trigger.
300. **A** — ON DELETE CASCADE removes the dependents whose essn = 1, which are Zara and Omar of employee 1. Sara's Zara has essn = 2, so it stays. The partial key dep_name is unique only within one owner. Verified in SQLite with PRAGMA foreign_keys = ON.
301. **C** — A relation is a table, each row is a tuple and each column is an attribute. A domain is the set of allowed values for an attribute, and degree is the number of attributes.
302. **D** — Degree (or arity) counts attributes, so STUDENT(roll_no, name, dept) has degree 3. The number of tuples is the cardinality, which changes as rows are inserted or deleted.
303. **D** — A domain is a set of atomic (indivisible) values, such as the integers 0 to 100 for marks. Each attribute of a relation takes its values from one domain, and the domain constraint enforces this.
304. **A** — A superkey uniquely identifies every tuple. A candidate key (Elmasri-Navathe simply call it a key) is a superkey with no redundant attribute. A set that merely contains the primary key is a superkey, not necessarily minimal.
305. **C** — Entity integrity says no attribute of the primary key may be NULL, because the primary key identifies each tuple and a NULL could not identify anything. The foreign key rule is referential integrity.
306. **B** — In Elmasri-Navathe a foreign key value must either equal the primary key of some existing tuple in the referenced relation or be NULL (when the FK is not part of the referencing relation's primary key and NULLs are allowed). Sharing a domain is necessary but not enough.
307. **C** — A composite key uses several attributes together, for example (student_id, course_id) in an enrolment table. Neither attribute alone is unique, but the pair is. A DBMS-generated key is a surrogate key.
308. **C** — Cardinality is the number of tuples in a relation's current state, so it changes with every insert and delete. Degree and arity both mean the number of attributes, and domain size is the number of values a domain allows.
309. **A** — A candidate key is defined as a minimal superkey, so it is always a superkey. A superkey can carry extra attributes, such as {roll_no, name}, which is unique but not minimal. A relation usually has many superkeys.
310. **C** — A relation is a set of tuples, so tuple order carries no meaning and duplicate tuples cannot exist. Values are atomic (the flat relational model), so a multivalued attribute such as several phone numbers must go into a separate relation.

311. **B** — A foreign key may reference the relation it belongs to; Elmasri-Navathe call this a recursive relationship. super_ssn must match an existing ssn or be NULL, for example for the top manager. It is not unique, so it is not a key.
312. **D** — An attribute is prime if it is a member of some candidate key, not only the chosen primary key. The definition matters in normalisation, where 2NF and 3NF treat prime and non-prime attributes differently.
313. **B** — Candidate keys not chosen as the primary key are alternate keys; Elmasri-Navathe use the term unique keys. In SQL they are usually declared with UNIQUE. They are still minimal, so 'superkeys that are not minimal' is wrong.
314. **A** — Deleting a tuple cannot break domain, key or entity integrity, but it can leave foreign keys elsewhere pointing at a tuple that no longer exists. The DBMS then restricts the delete, cascades it or sets the referencing values to NULL or a default.
315. **B** — Constraints that cannot be stated directly in the schema's key, domain or referential definitions are application-based or semantic constraints. They are enforced by application code, triggers or assertions.
316. **A** — An inserted tuple can have a value outside its domain, a duplicate key, a NULL primary key, or a foreign key that matches nothing. So insertion can break all four. Deletion, by contrast, can break only referential integrity.
317. **D** — Degree counts attributes (5) and cardinality counts tuples (120). Multiplying them gives the number of cells, which is neither measure.
318. **A** — The key constraint says no two tuples can share a key value. Entity integrity would be violated by a NULL roll_no, and referential integrity concerns foreign keys.
319. **C** — A domain constraint requires every attribute value to be an atomic value from the attribute's domain. 'twenty' is not an integer in the range, so the insert is rejected.
320. **B** — A relation state is a subset of the Cartesian product of its domains, which has 2 × 3 × 4 = 24 combinations. Duplicate tuples are not allowed, so 24 is the maximum. Adding the sizes gives 9.
321. **D** — Two tuples share name = 'Ali', so {name} cannot be a key. The other sets have unique values in this instance, but an instance can only rule a key out; whether they are keys depends on the meaning of the data.
322. **B** — Every superkey must contain A; the other three attributes can each be in or out, giving 2^3 = 8 superkeys. 15 counts every non-empty subset, including ones without A.
323. **C** — A superkey must contain both A and B, and C and D can each be added or not: 2^2 = 4. They are {A,B}, {A,B,C}, {A,B,D} and {A,B,C,D}.
324. **A** — A foreign key value must match an existing primary key in the referenced relation. Nothing creates the parent row automatically. Entity integrity would only apply if a primary key attribute were NULL.
325. **B** — Entity integrity forbids NULL in any primary key attribute. The key constraint is about duplicate key values, and a NULL is not a duplicate of anything.
326. **C** — ON DELETE CASCADE deletes the referencing rows together with the referenced row. Ali, Omar and Hina (dno 10) are removed, leaving Sara and Bilal. Verified in SQLite with PRAGMA foreign_keys = ON, which SQLite needs before it enforces foreign keys.
327. **D** — ON DELETE SET NULL keeps the referencing rows but sets their foreign key to NULL. Ali and Omar pointed at department 10, so their dno becomes NULL. Verified in SQLite with foreign keys enabled.
328. **B** — A+ = {A, B, C}, so A alone determines every attribute and is a key. B+ = {B, C} misses A. {A, B} and {A, C} are superkeys but not minimal because A alone suffices.
329. **B** — Supersets of A: 2^3 = 8. Supersets of B: 8. Sets containing both A and B are counted twice: 2^2 = 4. By inclusion-exclusion, 8 + 8 − 4 = 12.
330. **D** — A foreign key may be NULL only if it is not part of the referencing relation's primary key. Here student_id is in the primary key, so entity integrity applies. Note that SQLite accepts NULL in a non-INTEGER primary key unless NOT NULL is declared; standard SQL does not.
331. **A** — Supersets of {A, B}: 2^2 = 4. Supersets of {C}: 2^3 = 8. Sets containing A, B and C are counted twice: 2^1 = 2. Total 4 + 8 − 2 = 10.
332. **C** — D appears on no right-hand side, so every key must contain D. A, B and C determine each other in a cycle, so any one of them plus D gives the closure {A, B, C, D}. Neither D alone nor A, B or C alone reaches D.

333. **C** — Keys are properties of the schema, true for every valid state, not just today's. One state can disprove a key (two tuples share A) but cannot prove one, because a future tuple might repeat a B value. The designer decides from the meaning of B.
334. **D** — Every single attribute repeats a value (A = 1 twice, B = x twice, C = p twice). Every pair is unique: (1,x), (2,x), (1,y), (3,y) for AB, and likewise for AC and BC. So the three pairs are the minimal unique sets in this instance; {A, B, C} is not minimal.
335. **A** — Elmasri-Navathe note that every relation has at least one superkey, the set of all its attributes, because no two tuples are identical. SQL tables are multisets and can hold duplicate rows when no key is declared, which is one way SQL departs from the formal model.
336. **A** — Deleting 20 succeeds because no employee references it. Deleting 10 fails under RESTRICT because Ali and Sara reference it. The UPDATE moves Omar to 10, so 30 is no longer referenced and its delete succeeds. Only 10 remains (checked in SQLite).
337. **D** — The delete removes dept 1 (1 row), cascades to employees 11 and 12 (2 rows), and cascades again to their dependants Amna, Hamza and Zoya (3 rows): 6 in total. Employee 13 and Ahmed belong to dept 2 and stay. Verified in SQLite.
338. **B** — A department needs an existing manager and an employee needs an existing department, so the first insert into either table fails. Making one FK nullable, or deferring the constraint check to commit time, breaks the cycle. Circular references are legal; Elmasri-Navathe's COMPANY schema has one.
339. **D** — Changing a referenced primary key value would leave employees pointing at department 5, which no longer exists, so it violates referential integrity. Updating non-key attributes cannot break any foreign key, and changing an unreferenced key is safe.
340. **A** — The flat relational model needs atomic values, so a list in one cell breaks first normal form. A separate relation with one phone per tuple, keyed on (ssn, phone) and with ssn as a foreign key, stores any number of phones correctly.
341. **D** — A relational database schema is S = {R1, …, Rm} plus a set of integrity constraints IC. The data stored at a moment is the database state, not the schema.
342. **B** — Inherent constraints come from the relational model itself. A relation is a set, so it cannot hold duplicate tuples, whatever the DDL says. Schema-based constraints are declared in the DDL, and application-based ones come from business rules.
343. **B** — Schema-based constraints are written directly in the schema through the DDL. Domain, key, NOT NULL, entity integrity and referential integrity constraints are all in this group.
344. **D** — NULL can mean value unknown, value unavailable or withheld (such as an unlisted phone), or not applicable (such as a visa status for a citizen). It is not a value like 0 or ''.
345. **A** — Changing an ordinary attribute does not alter any tuple's identity or any reference. The DBMS only needs to check that the new value has the correct data type and domain. Key and referential checks matter only for key or foreign-key attributes.
346. **B** — The options are restrict (reject), cascade (propagate the delete) and set null or set default (change the referencing values). Renaming keys is not one of them. Renaming would be an update, and it would leave the reference pointing at the wrong tuple.
347. **C** — Elmasri-Navathe require a foreign key to have the same domain(s) as the referenced primary key and to take values from it. Names are free, as in EMPLOYEE.Dno → DEPARTMENT.Dnumber.
348. **B** — SQL lets a foreign key reference either the primary key or a UNIQUE (alternate) key of the parent. Elmasri-Navathe's formal definition uses the primary key, but SQL is broader. SQLite reports 'foreign key mismatch' if the parent columns are not PRIMARY KEY or UNIQUE, which we confirmed.
349. **D** — State constraints must hold in each valid state on its own. Transition constraints govern how the database may change from one state to the next, such as new salary ≥ old salary. They are usually enforced with triggers.
350. **C** — The PRIMARY KEY clause declares the chosen key. Each other candidate key is declared UNIQUE, which Elmasri-Navathe call unique keys. FOREIGN KEY declares references, CHECK restricts values and DEFAULT fills missing values.
351. **B** — Both candidate keys are minimal and unique, so either is valid. Elmasri-Navathe advise choosing one with a single attribute or a small number of attributes. It is shorter to store, index and reference as a foreign key.
352. **D** — A state that obeys all the constraints in IC is a valid state. A state that breaks any of them is invalid. The initial state is the one loaded first, and it should also be valid.

353. **B** — In the usual definition a tuple is an ordered list, so position tells you which value belongs to which attribute. If each value is paired with its attribute name, position no longer matters. Tuple order within a relation never mattered in either definition.
354. **C** — A delete can violate only referential integrity, and only when the deleted tuple is referenced by others. No relation references EMP, so deleting an employee leaves no dangling reference.
355. **A** — The new DEPT tuple can repeat an existing dno (key), have a NULL dno (entity integrity) or hold a bad value (domain). It contains no foreign key, and nothing yet refers to it, so referential integrity cannot be violated.
356. **D** — A formal relation is a set, so it has no duplicates. A SQL table is a multiset (bag), so without a key it can store the same row twice. Neither has a guaranteed row order, and SQL tables allow NULLs.
357. **B** — Elmasri-Navathe use this example. A domain is a set of values, and many attributes can draw from it. The attribute name tells you the role, such as home versus office.
358. **A** — Semantic rules, such as 'salary must not exceed the supervisor's', cannot be written as plain key or domain constraints. They are checked in application code or, more generally, with triggers and assertions.
359. **A** — Degree counts attributes, which come from the schema. Cardinality counts tuples, which come from the state. An empty state is valid, so the cardinality is 0 while the degree stays 6.
360. **B** — SET DEFAULT changes only the foreign-key column of the referencing rows, putting in its declared DEFAULT. That default must itself exist in the parent table, or the referential check fails.
361. **B** — A deferred constraint may be broken temporarily inside a transaction, but it must hold by COMMIT. This is the standard way to insert rows that reference each other in a cycle.
362. **D** — Entity integrity says no primary key value can be NULL, and for a composite key that covers each of its attributes. Alternate keys declared UNIQUE are not covered unless NOT NULL is added too.
363. **C** — The referencing relation is the one that holds the foreign key, here EMP. The referenced relation, here DEPT, holds the primary key being pointed to.
364. **A** — The schema is the relation's intension: its name and attributes, which rarely change. The state is its extension: the set of tuples at a moment, which changes constantly.
365. **A** — A surrogate key is an artificial identifier with no business meaning, created only to identify tuples. A natural key comes from real-world data, such as a CNIC.
366. **C** — ON UPDATE CASCADE copies the parent's new key into every child row. The three employees of department 10 now have dno 30. Verified in SQLite with PRAGMA foreign_keys = ON.
367. **D** — A superkey must contain {A, B, C} and may add any subset of the other 3 attributes, so there are 2^3 = 8. We checked this by enumerating all 63 non-empty attribute sets.
368. **D** — Sets containing A: 2^4 = 16. Sets containing B: 16. Sets containing both: 2^3 = 8. By inclusion-exclusion, 16 + 16 − 8 = 24. Checked by enumeration.
369. **D** — Sets containing {A, B}: 2^3 = 8. Sets containing {B, C}: 8. Sets containing {A, B, C}: 2^2 = 4. So 8 + 8 − 4 = 12. Checked by enumeration.
370. **B** — Inclusion-exclusion: 3 × 2^3 − 3 × 2^2 + 2^1 = 24 − 12 + 2 = 14. Put another way, every non-empty subset except {D} contains A, B or C: 15 − 1 = 14. Checked by enumeration.
371. **D** — Sets containing {A, B}: 2^2 = 4. Sets containing {C, D}: 4. Their only common set is {A, B, C, D}. So 4 + 4 − 1 = 7. Checked by enumeration.
372. **B** — C and D never appear on a right-hand side, so every key contains them, but {C, D}+ = {C, D}. Adding A, B or E each gives all five attributes: for example {C, D, E} → A (via ED) → B. All three are minimal. Checked with an attribute-closure program.
373. **A** — B is on no right-hand side, so it is in every key, but {B}+ = {B}. {A, B} gives C, then D. {B, C} gives D, then A. {B, D} gives A, then C. {C, D}+ = {A, C, D} misses B. Checked by closure.
374. **D** — The keys are {A}, {E}, {B, C} and {C, D}. {A} gives B and C, then D (from B), then E. {E} gives A. {B, C} gives D, then E (from CD), then A. {C, D} gives E, then A. Checked by enumeration with closures.
375. **C** — {A}+ = {A, B, C, D}, so {A} is the only candidate key. {A, C} contains it, so it is a superkey, but not a minimal one. Sets without A cannot reach A, because A is on no right-hand side.
376. **B** — Changing roll 502 to 501 would give two tuples the same primary key value, which breaks the key (uniqueness) constraint. SQLite rejects it with 'UNIQUE constraint failed: student.roll'.

377. **C** — With no action declared, deleting department 10 is rejected because Ali and Omar still reference it. Department 20 has no employees, so its delete succeeds, leaving 1 row. Verified in SQLite.
378. **A** — Ali's dno 10 exists, so that row is accepted. Sara's NULL dno is allowed, because a NULL foreign key references nothing. Omar's dno 99 matches no department, so that row is rejected. Verified in SQLite.
379. **A** — Ali (no supervisor) is accepted. Sara's first insert fails because supervisor 3 does not exist yet. Omar (supervisor 1) is accepted, and then Sara's retry succeeds. That leaves 3 rows. Verified in SQLite.
380. **A** — Deleting Sara (2) cascades to her reports Hina (4) and Bilal (5). Deleting Hina cascades again to Usman (7). Ali, Omar and Zara remain, so 3 rows. Verified in SQLite.
381. **C** — ON DELETE SET DEFAULT moves the three employees of department 10 to the column default, 0. Department 0 exists, so the result still satisfies the foreign key. Verified in SQLite.
382. **B** — A composite foreign key must match a whole (course, sec_no) pair. (CS101, 1) and (CS101, 2) exist. (CS102, 1) fails even though CS102 and section number 1 each appear somewhere, and CS103 does not exist. Verified in SQLite.
383. **B** — Omar's dno, 40, matches no department. Sara's NULL is allowed, because a foreign key may be NULL when it is not part of its own relation's primary key. Hina's 30 and Ali's 10 both exist. Checked with an anti-join query in SQLite.
384. **B** — The row with sid NULL breaks entity integrity and must go. (1, C1) appears twice, so one of those rows must go to satisfy the key constraint. That makes 2. Different grades do not make the rows distinct keys. Checked with GROUP BY in SQLite.
385. **D** — 120 + 4 successful inserts − 3 deletes = 121. A rejected insert changes nothing, and the degree does not change at all.
386. **A** — No candidate key can contain another, so the keys form an antichain of attribute sets. On 4 attributes the largest antichain is all 2-attribute sets, C(4, 2) = 6 (Sperner's theorem), and suitable FDs can make each of them a key. Confirmed by brute force over all families of subsets.
387. **A** — Candidate keys must be minimal. A set that properly contains a key is only a superkey. Separately, uniqueness in one state cannot establish a key, which must hold in all valid states.
388. **C** — Only rejecting the delete keeps both the orders and their correct customer. A customer with orders could instead be marked inactive.
389. **C** — Dependents exist only through their employee, so removing the employee should remove them too. This is the usual weak-entity choice.
390. **A** — Entity integrity forbids NULL in any primary key attribute. A standard DBMS treats primary key columns as NOT NULL, so deleting a student with enrolments would fail. The declared action cannot do what it says. CASCADE or RESTRICT is consistent.
391. **A** — Under MATCH SIMPLE, if any column of a composite foreign key is NULL, the constraint is not checked. So (CS999, NULL) is accepted even though CS999 does not exist. (CS101, 3) is fully non-NULL and matches no section. Verified in SQLite, which uses this rule.
392. **D** — Deferred checks run at COMMIT. By then department 10 has manager 1 and employee 1 is in department 10, so both references hold. Without DEFERRABLE, the first INSERT would fail at once, which we also confirmed in SQLite.
393. **A** — B must be in every key, but {B}+ = {B}. {B, D}+ adds A from D → A, then C from AB → CD, giving all attributes. Neither B nor D alone works. So the keys are {A, B} and {B, D}. Checked by closure.
394. **D** — If K's values are unique, adding attributes keeps them unique, so supersets of a superkey are superkeys. Removing attributes may lose uniqueness, and K may not be minimal.
395. **C** — {A}, {B} and {C} are all candidate keys, so every non-empty subset contains a key and is a superkey: 2^3 − 1 = 7. Checked by enumeration with closures.
396. **C** — The NULL eid breaks entity integrity, and dno 40, which is non-NULL and matches no department, breaks referential integrity. One INSERT can break several constraints at once.
397. **C** — A key must tell every tuple apart. UNIQUE ignores NULLs, which are not equal to each other, so many rows can share NULL. We verified that SQLite accepts two NULL cnic values but rejects a repeated non-NULL one. Adding NOT NULL makes it a true alternate key.

398. **D** — Deleting department 10 cascades to employees 1 and 2. Employee 1 is still referenced by assign, which has no action, so the cascaded delete breaks referential integrity. The whole statement fails and rolls back. Verified in SQLite.
399. **C** — A and D appear on no right-hand side, so both are in every key. {A, D}+ = {A, B, C, D}, so {A, D} is the only key. The prime attributes are A and D, which leaves B and C non-prime. Checked by closure.
400. **C** — A foreign key points to a unique value in the parent. In the child it repeats freely, which is how one department has many employees (1:N).
401. **C** — Selection σ_condition(R) is a horizontal filter: it keeps whole tuples that satisfy the condition. Choosing columns is projection π, pairing all tuples is the Cartesian product ×, and renaming is ρ.
402. **A** — Projection keeps only the listed attributes, and because a relation is a set, duplicate tuples are removed. SQL's SELECT dept keeps duplicates unless you write DISTINCT; this is a classic difference.
403. **D** — Union compatibility (type compatibility) means both relations have the same number of attributes and each pair of corresponding attributes has the same domain. The tuple counts can differ freely.
404. **B** — R ÷ S returns the values in R that are paired with every tuple of S, which is exactly a 'for all' query. The other operators cannot express it alone.
405. **B** — A natural join is an equijoin on every pair of same-named attributes, and the duplicate columns are dropped. Pairing everything is the Cartesian product, and keeping unmatched tuples is an outer join.
406. **B** — ρ_S(B1, …, Bn)(R) renames R to S and its attributes to B1…Bn. It is needed for self-joins, where the same relation appears twice and the attribute names would otherwise clash.
407. **A** — A theta join is a Cartesian product followed by a selection on the join condition θ. An equijoin is a theta join where θ uses only equality.
408. **A** — In tuple relational calculus, t ranges over tuples of EMP and the formula keeps those with salary > 50. It is equivalent to σ_salary>50(EMP). Calculus says what to retrieve; algebra says how.
409. **C** — Elmasri-Navathe give {σ, π, ∪, ρ, −, ×} as a complete set. Intersection, joins and division can all be rewritten with these. Without difference you cannot express negation, so the other sets fall short.
410. **B** — R − S removes the common tuples from R, so subtracting that from R leaves exactly the common tuples. This is why intersection is not in the complete set: difference expresses it.
411. **C** — Selection is commutative, and a cascade of selections equals one selection with the conditions joined by AND. Optimisers use this to split and push conditions down the query tree.
412. **B** — The left outer join keeps every left tuple. Unmatched left tuples appear once with NULLs in the right-hand attributes. The right outer join does the same for S, and the full outer join does both.
413. **B** — A safe expression is guaranteed to give a finite result whose values come from the database. All tuples not in STUDENT range over the whole universe, which is infinite. Negation itself is allowed when it is restricted to a relation.
414. **A** — Both are non-procedural and equally expressive (when safe). TRC binds variables to whole tuples, as in {t | EMP(t)}; DRC binds them to domain values, as in {<n> | ∃s (EMP(n, s))}. SQL is closer to TRC and QBE to DRC.
415. **C** — Codd called a language relationally complete if it can express any query expressible in relational algebra, which is equivalent to safe relational calculus. Basic relational algebra has no aggregation, sorting or recursion, so those are extras.
416. **D** — With no common attributes, the join condition is empty and therefore always true, so every pair of tuples qualifies. The result equals R × S, with |R| × |S| tuples.
417. **A** — A Cartesian product pairs every tuple of R with every tuple of S: 4 × 5 = 20 tuples. Each result tuple has all attributes of both: 3 + 2 = 5.
418. **A** — Ali (60), Hina (70), Bilal (55) and Usman (65) earn more than 50. Omar earns exactly 50, and > excludes him; including him gives 5.
419. **B** — The dept column holds CS, EE and ME. Projection returns a set, so duplicates are removed and 3 tuples remain. SELECT dept FROM EMP in SQL would return 7 rows.
420. **D** — Union removes duplicates: {1, 2, 3, 4, 5, 6} has 6 tuples. Adding 5 + 3 = 8 counts 4 and 5 twice.
421. **C** — R − S keeps tuples in R that are not in S: 4 and 5 are removed, leaving {1, 2, 3}. {6} is S − R, {4, 5} is the intersection, and {1, 2, 3, 6} is the symmetric difference.

422. **D** — They join on B. A = 1 and A = 3 (B = x) each match two S tuples (10, 20): 4 tuples. A = 2 (B = y) matches 30: 1 tuple. A = 4 (B = z) matches nothing. Total 5.
423. **C** — The 5 matching tuples are kept, and the unmatched left tuple (4, z) is added once with C = NULL: 6. The unmatched S tuple (w, 40) is kept only by a right or full outer join.
424. **D** — Count pairs with a < c: a = 1 pairs with 2, 3, 4 (3); a = 2 with 3, 4 (2); a = 3 with 4 (1). Total 6. 12 is the full Cartesian product before the condition.
425. **A** — Select first, then project: the condition needs dept and salary, which a prior projection onto name would remove. With the EMP data it returns Ali and Omar. ∨ would also return non-CS staff, and the last option ignores the dept condition.
426. **D** — If every tuple of S is already in R, the union has 5 tuples. If none overlap, it has 5 + 3 = 8. It can never be smaller than the larger operand.
427. **A** — At most 3 of R's tuples can also be in S, so at least 5 − 3 = 2 remain. If nothing overlaps, all 5 remain.
428. **C** — Division keeps each sid paired with every course in CORE (DB, OS, CN). S1 and S3 have all three; S3's extra AI course does not matter. S2 lacks CN and S4 has only CN. Checked with a Python implementation of ÷.
429. **D** — Each order's cid is not NULL and matches exactly one customer, because cid is CUSTOMER's key. So every order joins with exactly one customer tuple: 100 results. 2000 would be the Cartesian product.
430. **C** — The existential quantifier links each employee to a department through dno = dnum, which is a join; the dname condition is a selection, and t.name is a projection. Without the join condition (the × option) every employee pairs with Research.
431. **D** — π_A(R) × S lists every (A, B) pair that would be needed; subtracting R leaves the missing pairs; projecting gives the A values that miss something; subtracting those from π_A(R) leaves A values paired with all of S. A random test of 3,000 relations confirmed it.
432. **A** — A sid qualifies if it is paired with every course in CORE. With no courses, that holds vacuously for every sid. The formula π_A(R) − π_A((π_A(R) × S) − R) agrees: the product is empty, so nothing is subtracted.
433. **C** — π_name(EMP) keeps only name, so a following selection on salary refers to an attribute that no longer exists. Selecting before projecting is fine, and a cascade of projections is valid when the outer list is a subset of the inner one.
434. **B** — The natural join matches on every shared name, here both cid and city. Only orders 101 (Ali, Lahore) and 103 (Sara, Karachi) agree on both. Joining on cid alone would give 4. Checked with a Python natural-join implementation.
435. **A** — Relational algebra works on sets, so projection removes duplicates. SQL works on multisets (bags) and keeps duplicate rows unless you write SELECT DISTINCT. Checked in SQLite: 7 rows, and 3 with DISTINCT.
436. **D** — Projection does not distribute over difference. With R = {(1, a)} and S = {(1, b)}, π_A(R − S) = {1} but π_A(R) − π_A(S) = ∅. The other three held on 3,000 random relation pairs in a Python check.
437. **D** — Two copies of EMP are needed, renamed E (employee) and S (supervisor), joined on E.sup = S.eid. Reversing the join condition makes E the supervisor, so it returns supervisors who out-earn a subordinate instead.
438. **B** — E1 removes S1 and S3 (who took CS101) from all students, leaving S2, S4 and S5. E2 lists students with some other course: S1 (MA101) wrongly appears, and S5, who has no enrolment, is missed. Negation needs difference, not ≠.
439. **B** — The natural join gives 5 tuples. The full outer join adds the unmatched left tuple (4, z, NULL) and the unmatched right tuple (NULL, w, 40): 7. Checked with a Python outer-join implementation.
440. **C** — The safe expression ties t to STUDENT, so every result tuple comes from the database (students with no enrolment). The others let t range over tuples not in any relation, which gives an infinite or undefined result.
441. **B** — This is the closure property of relational algebra: the input and the output of every operator are relations. The result of σ_salary>50(EMP) is itself a relation, so π can be applied to it, and so on to any depth.
442. **B** — Selection (σ), projection (π) and rename (ρ) each act on one relation. Cartesian product, union, difference, join and division are binary: they combine two relations.
443. **B** — Relational algebra is procedural: an expression states the operators and their order. Relational calculus is declarative: a formula states what the result tuples must satisfy, not how to find them. Safe calculus and algebra have the same expressive power.

444. **B** — An equijoin is a theta join that uses only equality, and it keeps every attribute of both relations, so the result has two identical dept columns. A natural join equates the same-named attributes and then drops the duplicate column.
445. **C** — A semijoin keeps the R tuples that have a join partner in S and returns them with R's schema only, so it equals π_attrs(R)(R ⋈ S). It never makes a tuple wider or duplicates it.
446. **C** — R − S keeps the tuples of R that are not in S, while S − R keeps those of S that are not in R; these are usually different. Union, intersection and natural join give the same tuples whichever operand comes first.
447. **B** — σ keeps the tuples that satisfy a condition, which is exactly what WHERE does to rows. Confusingly, the SELECT list corresponds to projection π, not selection.
448. **A** — The assignment operator ← names an intermediate result so a long query can be written as a sequence of simple steps. It does not change EMP or the stored schema.
449. **B** — Selection keeps whole tuples, so the attributes (degree) do not change. It can only remove tuples, so the cardinality is between 0 and |R|. Projection is the operator that reduces the degree.
450. **A** — R ⋈_θ S is defined as σ_θ(R × S). If θ never fails, the selection removes nothing, leaving the full Cartesian product.
451. **D** — With identical schemas, the natural join must match every attribute, so a pair joins only when the two tuples are identical. The result is the set of tuples in both relations, which is R ∩ S.
452. **D** — The outer projection can only keep attributes that the inner one still has, so L1 must be contained in L2. Then projecting twice is the same as projecting once onto L1.
453. **D** — A join or product over small inputs is far cheaper than one over full tables. Applying σ first removes unwanted tuples before they are combined, which is the most important algebraic heuristic in Elmasri-Navathe.
454. **D** — R ⋉ S holds the R tuples that do have a partner in S. Subtracting it from R leaves the R tuples with no partner. R − S needs union-compatible relations and compares whole tuples, not join matches.
455. **C** — A variable is bound when a quantifier (∃ or ∀) introduces it, and free otherwise. d is bound by ∃d. t is not quantified, and in a TRC query the variable left of the bar must be the only free one.
456. **C** — 'P holds for every x' means 'there is no x for which P fails'. This rewrite lets a ∀ query be expressed with ∃ and negation, which is also how SQL writes it with NOT EXISTS.
457. **D** — The left outer join adds R's unmatched tuples and the right outer join adds S's; both contain the matched tuples. Their union therefore holds the matches plus both sets of NULL-padded tuples, which is the full outer join.
458. **D** — QBE variables stand for individual attribute values written into columns, which is the domain relational calculus. SQL's FROM-clause tuple variables are closer to the tuple relational calculus.
459. **A** — Division asks for values paired with every required item. SQL says this as 'there is no required item that this value lacks': NOT EXISTS (required item ... NOT EXISTS (matching row)). A single IN or an outer join only finds 'at least one' or 'none'.
460. **A** — Codd's algebra included selection, projection, the set operations, product, join and division. Aggregation such as COUNT and AVG per group was added later as an extension, written $\mathscr{F}$ in Elmasri-Navathe and γ in many other textbooks.
461. **C** — Each join climbs one fixed level, so an expression can only reach a fixed depth. 'Ancestors at any depth' needs a transitive closure, which needs recursion (as in SQL's WITH RECURSIVE). The other queries use one self-join, a difference or a self-join with ≠.
462. **C** — σ, ⋈, ∪, π and × are monotone: more input can only mean more (or the same) output. Difference is not, because a new tuple in S can remove a matching tuple from R − S. Division is non-monotone for the same reason, since it is built from difference.
463. **D** — With sets, a union of a relation with itself adds nothing. With bags, SQL's UNION ALL doubles every row, so idempotence fails. Commutativity of union, commuting selections and the product's size hold under both semantics.
464. **B** — σ, π, ∪, ρ and × are all monotone, so any combination of them is monotone too; difference is not, so it cannot be derived and must be a basic operator. Intersection is derivable from difference, so it is redundant in a complete set.
465. **A** — dept = 'IT' gives Danish, Fatima and Kamran. salary < 45 gives Iqra (39) and Nabeel (44). No tuple is in both, so the OR gives 5 tuples.

466. **A** — The selection keeps Danish (75), Fatima (48) and Kamran (75). Projecting onto salary gives 75, 48, 75, and because the result is a set the duplicate 75 is removed, leaving 2 tuples.

467. **B** — DEPT has HR, IT, FIN and MKT. EMP uses HR, IT, FIN and OPS. Removing the departments that appear in EMP leaves MKT, the department with no employees. OPS would be the answer to the reverse difference.

468. **A** — The variables e, n, d and s range over the four attribute values of each EMP tuple, and only n is returned. Salaries above 60 are 75 (Danish), 61 (Hamza) and 75 (Kamran). Laiba earns 58.

469. **B** — The HR side gives Ayesha and Iqra. The salary side gives Danish and Kamran, both on 75. The two sets do not overlap, so the union has 4 names.

470. **D** — If the relations share no tuples, the intersection is empty. At most, every tuple of the smaller relation S also appears in R, giving 4. The intersection can never be larger than the smaller operand.

471. **C** — floor is in DEPT, so EMP must first be joined to DEPT on dept. Then the selection keeps floor 3 (IT) and the projection returns Danish, Fatima and Kamran. Selecting on floor directly from EMP, or after floor has been projected away, is invalid.

472. **D** — The 8 (dept, salary) pairs are all different except Danish and Kamran, who are both (IT, 75). The set result keeps one copy, so there are 7 tuples. A duplicate is removed only when the whole projected tuple repeats.

473. **A** — The join matches on dept. Every employee matches one DEPT tuple except Nabeel, whose department OPS is not in DEPT, so 7 tuples remain. MKT has no employees, but an inner join simply ignores it. 32 is the Cartesian product.

474. **D** — The inner join gives 7 tuples. A right outer join also keeps every unmatched DEPT tuple, padded with NULLs: MKT has no employees, so (NULL, NULL, MKT, NULL, 4) is added. Nabeel is unmatched on the left side, so he is not kept. Total 8.

475. **B** — EMP ⋉ DEPT keeps the 7 employees whose department exists in DEPT. Removing them from EMP leaves Nabeel, whose department OPS has no DEPT tuple. The result has EMP's attributes, so it can never contain a DEPT tuple such as MKT.

476. **A** — This lists each pair of colleagues in the same department once. HR has 1 pair (1, 5), IT has 3 pairs (2, 3), (2, 6), (3, 6), FIN has 1 pair (4, 7) and OPS has none: 5 in total. Using ≠ instead of < would count each pair twice (10).

477. **A** — For b = 3, a must lie strictly between 3 and 6: only 4. For b = 5, a must lie strictly between 5 and 10: 6 and 8. That gives (4, 3), (6, 5) and (8, 5), so 3 tuples.

478. **A** — Departments above floor 1 are IT (3), FIN (2) and MKT (4). Employees in IT are Danish, Fatima and Kamran; in FIN, Hamza and Laiba; MKT has none. That is 5 names. The ∃ quantifier acts like a join followed by a projection.

479. **D** — There is one result tuple per department: HR, IT, FIN and OPS. IT has 3 employees with salaries 75, 48 and 75, so the average is 198 / 3 = 66. 75 is the IT maximum, not the average.

480. **A** — Because B is a key of S, each R tuple matches at most one S tuple, so the join cannot exceed |R| = 10. Without a foreign key, some or all R tuples may match nothing, so the minimum is 0. 60 would need every pair to match.

481. **D** — The inner selection finds every employee who earns less than someone else. Removing them leaves those nobody out-earns: the maximum. Danish and Kamran both earn 75 and neither earns less than the other, so both are returned. This is how 'highest salary' is written without aggregates.

482. **C** — The divisor is the set of IT projects, {P1, P3}. Employee 2 works on P1, P2 and P3, and employee 4 on P1 and P3, so both cover every IT project. Employee 3 has only P3 and employee 1 only P1 among IT projects.

483. **D** — Each A value in the quotient must appear in R with all 3 B values of S, so it uses at least 3 tuples of R. With 12 tuples, at most 12 / 3 = 4 A values can qualify.

484. **D** — The selection finds every employee who earns no more than some HR employee; HR's highest salary is 52. Removing them leaves those who earn more than every HR employee: Danish 75, Hamza 61, Kamran 75 and Laiba 58. Ayesha is removed because she is paired with herself (52 ≤ 52).

485. **C** — The quotient keeps the attributes not in CORE, here (sid, sem), and returns each pair that appears with both DB and OS. S1 took both in semester 1 and S3 both in semester 2. S2 took them in different semesters, and S1 in semester 2 took only DB.

486. **B** — A tuple is in σ_c(R − S) when it is in R, not in S, and satisfies c. That is exactly a tuple of σ_c(R) that is not in S. The selection only needs to filter R, because removing S's tuples does not depend on c.

487. **C** — The product pairs every tuple with itself, and salary ≤ salary is always true. So every eid appears in the subtracted set, and the difference is empty. Using < removes the self-pairs and returns {2, 6}.

488. **C** — An intersection keeps only the tuples that satisfy both conditions, so it equals σ_c1 ∧ c2(R), not the OR. The union, the De Morgan complement and the swapped condition all give the OR.

489. **C** — The condition is tested on one tuple at a time, and no tuple has two pno values, so the result is always empty. The correct query intersects two projections: π_eid(σ_pno='P1'(WORKS)) ∩ π_eid(σ_pno='P2'(WORKS)) = {1, 2}.

490. **C** — The selected pairs find employees with two different projects: 1, 2 and 4. Subtracting them from π_eid(WORKS) = {1, 2, 3, 4, 6} leaves {3, 6}. Zero projects also counts as 'at most one', so starting from π_eid(EMP) would give the complete answer {3, 5, 6, 7, 8}.

491. **D** — E1 joins first, then keeps tuples with floor 3, which drops every NULL-padded row: Danish, Fatima and Kamran, so 3. E2 filters DEPT to IT first, then the left outer join keeps all 8 employees, with NULL floor for the 5 not in IT. Pushing σ into the inner side of an outer join changes the result.

492. **B** — When S(b) holds, the formula is true for every possible value of a, including values that appear nowhere in the database. The result is therefore infinite, or depends on the domain, so the expression is unsafe. In each of the other expressions, every output variable is bound to a value in R.

493. **B** — Relational algebra uses set semantics, so R ∪ S = {1, 2, 3, 4}. SQL's UNION ALL is a bag union that keeps every row, giving 3 + 3 = 6. SQL's plain UNION would match the algebra and return 4.

494. **A** — A sid is in the quotient only if it is paired with every course in CORE. Nobody is paired with ML, so no student qualifies and the result is empty, however many other courses they took.

495. **A** — After the rename, EMP(eid, name, dept, salary) and DEPT2(d, floor) share no attribute name. A natural join with no common attributes is a Cartesian product: 8 × 4 = 32 tuples, most of them meaningless.

496. **C** — EMP ⋈ WORKS gives 9 tuples. The second natural join matches on every common name, pno and dept, so a tuple survives only when the employee's department controls the project: (1, P2) HR, (2, P1) IT, (2, P3) IT and (3, P3) IT. Total 4.

497. **C** — A Cartesian product pairs every R tuple with every S tuple, matched or not. If S is not empty, projecting it back onto R's attributes returns all of R, not just the R tuples that join with S.

498. **B** — The ∃ condition asks for at least one IT project. IT controls P1 and P3. Employee 1 works on P1, 2 on P1 and P3, 3 on P3 and 4 on P1 and P3, so the result is {1, 2, 3, 4}. Asking for every IT project would need ∀ (division) and give {2, 4}.

499. **A** — With S empty, no tuple pairs exist, so both the join and the product are empty. The left outer join still keeps each of R's 5 tuples, with NULL for C.

500. **B** — Plan A multiplies the full tables: 1,000 × 50 = 50,000. Plan B filters DEPT to 5 tuples first, so the product has 1,000 × 5 = 5,000. Both plans give the same final answer, which is why pushing σ down is a safe and useful rewrite.

501. **C** — DDL statements define or change database structure: CREATE, ALTER and DROP (and usually TRUNCATE). ALTER TABLE changes a table's definition, so it is DDL. UPDATE is DML, GRANT is DCL and SAVEPOINT is TCL.

502. **A** — GRANT gives privileges to users or roles and REVOKE takes them away. Because they control access rights rather than structure, data or transactions, they form the Data Control Language.

503. **D** — TCL commands manage transactions: COMMIT makes changes permanent, ROLLBACK undoes them, and SAVEPOINT marks a point you can roll back to. The other groups are DDL, DCL and DML respectively.

504. **B** — DML commands read or change the rows stored in tables: SELECT, INSERT, UPDATE and DELETE. UPDATE modifies existing rows, so it is DML. TRUNCATE and CREATE INDEX are treated as DDL, and REVOKE is DCL.

505. **B** — Logically, WHERE runs on each row before GROUP BY forms groups, so there is nothing to count yet. HAVING is evaluated after grouping, which is why group-level conditions like COUNT(*) > 3 belong there.

506. **B** — COUNT(*) counts rows regardless of their contents. COUNT(column) counts only the non-NULL values in that column, so the two differ whenever phone has NULLs. Counting distinct values needs COUNT(DISTINCT phone).

507. **A** — In SQL LIKE patterns, _ matches exactly one character and % matches any sequence of zero or more characters. So 'A_' matches two-character strings that start with A, such as 'Al'.

508. **A** — BETWEEN is inclusive at both ends, so rows with price exactly 100 or exactly 200 are included. It is shorthand for price >= low AND price <= high.

509. **C** — A primary key identifies each row, so its value (or combination of values) must be unique and can never be NULL. A table can declare only one primary key, though that key may span several columns. Referencing another table is the job of a FOREIGN KEY.
510. **B** — UNIQUE forbids duplicate non-NULL values but, unlike a primary key, does not imply NOT NULL, and you can declare as many UNIQUE constraints as you need. (Some products, such as SQL Server, allow only one NULL in a UNIQUE column.)
511. **C** — ON DELETE CASCADE propagates the delete: every referencing row in the child table is removed automatically, so no order is left pointing at a missing customer. Rejecting the delete is the default (NO ACTION/RESTRICT), and setting NULL needs ON DELETE SET NULL.
512. **B** — A CHECK constraint attaches a Boolean condition to a column or table, and any INSERT or UPDATE that makes the condition false is rejected. UNIQUE, NOT NULL and FOREIGN KEY enforce other specific rules, not arbitrary conditions.
513. **B** — DROP TABLE removes the table entirely: its structure, data, indexes and constraints. DELETE and TRUNCATE only remove rows and leave an empty table behind, and ALTER TABLE ... DROP COLUMN removes one column.
514. **A** — DELETE is a DML statement that can target specific rows with WHERE. TRUNCATE removes every row in one operation and cannot take WHERE, but the empty table, its columns and constraints remain. Only DROP removes the table itself.
515. **C** — The logical order is FROM, WHERE, GROUP BY, HAVING, SELECT, then ORDER BY. Because ORDER BY comes after SELECT, it can use column aliases defined in the select list, while WHERE cannot.
516. **D** — SQL uses three-valued logic. manager_id = NULL evaluates to UNKNOWN for every row, even NULL ones, and WHERE keeps only rows where the condition is TRUE. The predicate IS NULL is the correct test.
517. **A** — COUNT(*) counts all 5 rows. COUNT(phone) skips the two NULLs, giving 3. COUNT(DISTINCT phone) counts distinct non-NULL values, '0300' and '0301', giving 2.
518. **A** — AVG ignores NULLs, so it averages only 100, 200 and 300: 600/3 = 200. Getting 150 would mean dividing by all 4 rows, as if the NULL were 0.
519. **B** — Group sums are East 180, North 350 and South 320. HAVING keeps only groups whose total exceeds 300, so North and South remain, sorted by region.
520. **D** — 'A_%' needs an A, then exactly one more character, then anything. Ali, Amna and Asad match. 'A' alone fails because _ requires one more character, and Bilal and Sana do not start with A.
521. **C** — BETWEEN includes both endpoints, so marks 50, 60, 70 and 80 all qualify: 4 rows. Answering 2 treats both bounds as exclusive.
522. **D** — Rows are sorted by dept alphabetically (Admin, HR, IT), then by salary from high to low inside each dept. The order is Zain, Hina (6000), Sara (5000), Omar, Ali, so the second row is Hina.
523. **C** — DISTINCT on two columns removes rows whose whole (student, course) pair repeats. The distinct pairs are (S1, DB), (S1, OS), (S2, DB) and (S3, OS), so the count is 4. 3 counts distinct students only.
524. **D** — NOT IN ('Punjab', 'Sindh') excludes Lahore, Multan and Karachi. Quetta (Balochistan) and Peshawar (KP) remain, so the count is 2.
525. **A** — Group the rows by dept, then keep groups whose row count exceeds 2 with HAVING COUNT(*) > 2. An aggregate cannot appear in WHERE, and >= 2 would also return departments with exactly 2 employees.
526. **D** — a + b is NULL whenever either side is NULL, so only rows (1, 2) and (5, 6) contribute: 3 + 11 = 14. SUM(a) = 9 and SUM(b) = 12 each skip their own NULL, giving 21. SUM ignores NULL inputs, so the first result is not NULL.
527. **A** — A CHECK rejects a row only when its condition is FALSE. -1 >= 0 is false, so that insert fails. 0 >= 0 is true, and NULL >= 0 is UNKNOWN, which CHECK accepts; blocking NULL would need NOT NULL.
528. **C** — A foreign key value must match an existing dept.id or be NULL. 30 does not exist in dept, so that insert violates referential integrity. 10 and 20 exist, and a NULL foreign key is allowed unless the column is NOT NULL.
529. **D** — The DELETE removes the three 'pending' rows and the one 'cancelled' row, 4 rows in all. Only the two 'shipped' rows remain, so COUNT(*) returns 2.
530. **C** — With GROUP BY dept, MAX and MIN are computed separately for each department. IT's salaries are 7000, 9000 and 5500, so its row is IT, 9000, 5500. 4500 belongs to HR and is not part of the IT group.

531. **D** — id NOT IN (1, NULL) means id <> 1 AND id <> NULL. The comparison with NULL is UNKNOWN, so the whole condition is never TRUE and no row qualifies. NOT EXISTS or filtering out NULLs in the subquery avoids this trap.
532. **A** — dept is a grouping column, so testing it in HAVING is legal and removes the HR group after grouping. WHERE removes HR rows before grouping. The results match, but WHERE avoids grouping rows that will be thrown away.
533. **C** — SUM(score) = 60. AVG ignores NULLs and divides by the 3 non-NULL values, giving 20. Dividing by COUNT(*) uses all 5 rows, giving 12, while dividing by COUNT(score) uses 3, giving 20 again. So AVG(x) equals SUM(x)/COUNT(x), not SUM(x)/COUNT(*).
534. **B** — The LEFT JOIN keeps Admin with one row whose emp columns are NULL. COUNT(*) counts that row, giving 1, while COUNT(e.id) skips the NULL, giving 0. To count employees per department correctly, count a column from the right-hand table.
535. **A** — ROLLBACK TO SAVEPOINT sp1 undoes everything after sp1, both the insert of 3 and the delete of 1, but keeps the transaction open with rows 1 and 2. Row 4 is then inserted, and COMMIT makes 1, 2 and 4 permanent.
536. **D** — For grouping, all NULLs are treated as one group, so the groups are NULL, Karachi and Lahore: 3 rows. COUNT(*) counts rows, so the NULL group shows 2. COUNT(city) would show 0 for it.
537. **D** — When a query uses GROUP BY, every selected column must either be a grouping column or appear inside an aggregate. name is neither, and a department has many names, so the result would be ambiguous. Some products (MySQL in older modes, SQLite) accept it anyway and pick an arbitrary name.
538. **B** — COUNT(DISTINCT color) ignores NULLs and counts red and blue, giving 2. SELECT DISTINCT keeps NULL as a value and treats the two NULLs as duplicates, so it returns red, blue and one NULL: 3 rows.
539. **B** — '%a_' means anything, then 'a', then exactly one final character, so the second-to-last letter must be a. That holds for asad, bilal and faraz. Names ending in a (sana, amna, zara) fail because _ needs one more character after the a.
540. **C** — AND binds tighter than OR, so the condition is dept = 'IT' OR (dept = 'HR' AND salary > 50000). Every IT employee qualifies (Ali, Omar), plus Sara from HR. Parentheses are needed if the salary test should apply to both departments.
541. **B** — CHAR(n) is fixed-length: shorter values are padded with blanks to n characters. VARCHAR(n) is variable-length: it stores the actual characters up to a maximum of n. Both limit the length of one value, not the number of rows.
542. **A** — Changing a table's structure is DDL, done with ALTER TABLE ... ADD. UPDATE and INSERT are DML: they change rows, not the set of columns. Many DBMSs also accept the optional keyword COLUMN after ADD.
543. **A** — A DEFAULT value is used when the column is omitted from the INSERT (or DEFAULT is written explicitly). If the INSERT gives NULL or an empty string, that value is stored instead, because a value was supplied.
544. **C** — ASC is the default sort direction in SQL, so ORDER BY salary is the same as ORDER BY salary ASC. DESC must be written explicitly.
545. **A** — % matches any run of characters, including none, so 'A%' matches 'A' as well as 'Asim'. The underscore matches exactly one character. To match a literal % you need an ESCAPE character.
546. **D** — IN tests membership in a list, which means 'equal to any one of these values': a chain of equalities joined by OR. AND would need one value to equal both strings at once, which is impossible.
547. **B** — AS defines an alias, a name for an output column (or a table) used in the result and in ORDER BY. It does not change the table's structure or store anything.
548. **A** — A composite primary key makes the combination unique, so the same student can appear with many courses and the same course with many students, but a pair cannot repeat. Every column of a primary key is also NOT NULL.
549. **D** — COMMIT makes a transaction's changes permanent (durable). ROLLBACK only undoes the work of the current, uncommitted transaction, so the committed update is not affected.
550. **A** — Entity integrity requires every row to be identifiable, so primary key columns cannot be NULL. Referential integrity is the rule for foreign keys: each value must match a referenced key or be NULL.
551. **C** — GRANT SELECT lets ali query the table. WITH GRANT OPTION also lets ali pass that same privilege on to others. It does not add UPDATE, DELETE or DROP rights.

552. **B** — Without GROUP BY, HAVING treats all remaining rows as a single group. If the condition holds, one aggregate row is returned; if not, the result is empty.
553. **D** — In the logical order, FROM and WHERE run before SELECT, so an alias defined in SELECT is unknown to WHERE. Repeat the expression (WHERE salary * 12 > 6000). ORDER BY runs after SELECT and can use the alias.
554. **C** — SET NULL keeps the child rows and clears the foreign key so it no longer points at a missing parent. CASCADE would delete the rows, RESTRICT or NO ACTION would reject the delete, and SET DEFAULT would use the default.
555. **D** — With NO ACTION (or RESTRICT), the DBMS refuses any delete that would leave child rows pointing at a parent that no longer exists. CASCADE and SET NULL must be requested explicitly.
556. **B** — A foreign key must identify exactly one parent row, so the referenced columns must be a candidate key: the primary key or a UNIQUE column set. Matching data types alone is not enough.
557. **B** — MIN and MAX need only an ordering, and strings are ordered alphabetically, so they work on text. COUNT works on any type. SUM and AVG need numbers.
558. **B** — OR is TRUE if either side is TRUE, whatever the other side is. TRUE AND UNKNOWN is UNKNOWN, NOT UNKNOWN is UNKNOWN, and UNKNOWN OR FALSE is UNKNOWN.
559. **B** — INSERT ... SELECT inserts one row for each row the query returns. The source rows stay in staff, and archive must already exist with compatible columns.
560. **A** — Clauses must be written as SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ... ORDER BY. The logical evaluation order is different (FROM first, SELECT near the end), but the written order is fixed.
561. **A** — Every expression on the right of SET is evaluated with the row's values from before the update, so b = a uses the old a (1). The values swap. A programming-language style reading, where a changes first, would wrongly give 2 and 2. MySQL is a known exception that assigns left to right.
562. **C** — sara's privilege depends on ali's grant. CASCADE revokes the privilege from ali and from everyone who received it through ali. With RESTRICT instead, the REVOKE would fail because a dependent grant exists.
563. **C** — DISTINCT applies to the whole output row, not to the first column only. A row is removed only if another row has the same dept and the same name. To get one row per department you need GROUP BY dept.
564. **C** — The SQL standard raises a 'string data, right truncation' error when a non-blank value is too long for the column. Some systems differ: SQLite ignores the declared length, and MySQL in non-strict mode truncates with a warning, so the exam answer follows the standard.
565. **D** — Tech has Faisal (700), Hira (650) and Javed (500). Only the first two meet salary >= 600, so the count is 2.
566. **B** — Rows are sorted by dept first, alphabetically: Admin comes before Sales and Tech. Admin has only Mehwish, so she is first. salary DESC only orders rows within the same department.
567. **A** — Bushra's bonus is NULL. COALESCE returns its first non-NULL argument, so COALESCE(bonus, 0) is 0 and the total is 550 + 0 = 550. Without COALESCE, 550 + NULL would be NULL.
568. **D** — BETWEEN 450 AND 650 includes both ends, covering 450, 500, 550 and 650. NOT BETWEEN keeps the rest: 400 (Asim), 700 (Faisal) and 300 (Mehwish), so 3.
569. **B** — DECIMAL(p, s) has p significant digits in total, s of them after the decimal point. With 6 digits and 2 decimals, 4 digits remain before the point, so the maximum is 9999.99.
570. **D** — When a column with a DEFAULT is added, the existing rows take the default value, so all 7 rows have status 'active'. Without a DEFAULT they would hold NULL and the count would be 0.
571. **A** — Sales starts with 400 + 550 + 450 = 1400. Each of the 3 Sales rows gains 100, adding 300, so the sum is 1700.
572. **A** — Tech has three rows (Faisal, Hira, Javed), so COUNT(*) is 3. Hira's bonus is NULL, and COUNT(bonus) skips NULLs, so it is 2. 100 is SUM(bonus), not a count.
573. **B** — With no GROUP BY, the whole table is one group. The average salary is 3550 / 7 ≈ 507.1, which is above 500, so the group passes and COUNT(*) gives 7.
574. **D** — CASE takes the first branch that is true. 700 and 650 go to A. 550, 500 and 450 go to B, because 450 meets >= 450. 400 and 300 go to C. So B has 3.
575. **D** — Sorted by salary descending, the order is Faisal 700, Hira 650, Bushra 550 and so on. OFFSET 1 skips the first row and LIMIT 1 returns the next, so the result is the second-highest earner, Hira.

576. **B** — Grouping by two columns makes one group per distinct (course, sem) pair: (DB, 1) with S1, S2 and S3, (OS, 1) with S1, (DB, 2) with S2 and (OS, 2) with S3. That is 4 groups. 2 would be grouping by course alone, and 6 is the number of rows.

577. **D** — ESCAPE '!' makes !% a literal percent sign, so the pattern means: anything, then '0%', then anything. '50%OFF' and '150%' contain '0%'. '50OFF%' has its % after 'F', and '5OFF' has no % at all.

578. **D** — The primary key is the pair (sid, cid). The pair (2, 101) already exists, so inserting it again violates the key, even though the grade is different. The other pairs are new, and grade may be NULL because it is not part of the key.

579. **D** — Deleting department 2 sets dept_id to NULL for employees 11 and 12, which stay in the table. So there are still 4 rows, and only employee 10 has a non-NULL dept_id (employee 13 was NULL already). COUNT(dept_id) is 1.

580. **C** — ORDER BY 2 refers to the second column of the SELECT list, salary. Sorting it descending puts the highest salary, Faisal's 700, first. It does not sort by the literal value 2 or by the second table column.

581. **B** — The subquery is evaluated against the table as it was before the update, giving about 507.1. Four rows are below it (400, 500, 300 and 450), and each gains 50. The total rises from 3550 to 3750.

582. **D** — SUM(x) is 7 and COUNT(x) is 3, both integers, so SQLite performs integer division and gives 2. AVG always returns a floating-point value, 2.333... . SQL Server and PostgreSQL behave the same way; MySQL's / would give 2.3333.

583. **C** — WHERE first removes the rows with a NULL bonus: Bushra, Hira and Mehwish. Sales keeps Asim and Rizwan (850) and Tech keeps Faisal and Javed (1200). Both have 2 rows, so HAVING keeps both, and ORDER BY total DESC puts Tech first.

584. **A** — Only rows with a bonus take part. Sales keeps Asim (400) and Rizwan (450), so its maximum is 450, not Bushra's 550. Tech keeps Faisal (700) and Javed (500). Mehwish is Admin's only employee and has no bonus, so Admin forms no group at all.

585. **B** — The Tech average is (700 + 650 + 500) / 3 ≈ 616.7, computed before any row is removed. Every salary below it is deleted: 400, 550, 500, 300 and 450. Only Faisal (700) and Hira (650) remain, so the count is 2.

586. **C** — A DELETE with no WHERE removes every row but leaves the table, its columns and constraints in place. That is why it is a classic mistake: the statement is valid and runs without warning.

587. **C** — BETWEEN 650 AND 450 expands to salary >= 650 AND salary <= 450, which is impossible. Standard BETWEEN does not swap the bounds (only BETWEEN SYMMETRIC does). Written as BETWEEN 450 AND 650 it would return 4 rows.

588. **A** — For the three NULL bonuses, bonus > 25 is UNKNOWN and NOT UNKNOWN is still UNKNOWN, so Q1 keeps only Javed (20). Q2 tests IS NULL explicitly, so it returns Javed plus Bushra, Hira and Mehwish: 4.

589. **A** — An aggregate query without GROUP BY always produces one row. COUNT(*) of an empty set is 0, while SUM, MAX, MIN and AVG have no values to work on and return NULL.

590. **C** — WHERE keeps only Faisal and Hira, both in Tech, so GROUP BY sees one department. Departments with zero matches vanish rather than showing 0. To list them, count conditionally, for example SUM(CASE WHEN salary > 600 THEN 1 ELSE 0 END) with no WHERE.

591. **A** — Ticket 1 omits status, so it gets 'new'. Ticket 3 asks for the default with the DEFAULT keyword, so it also gets 'new'. Ticket 2 supplies NULL explicitly, and an explicit value always overrides the default. The count is 2.

592. **C** — An SQL statement is atomic: if any row violates a constraint, the whole statement's effect is undone. None of the three new rows is kept, so t still holds only (1, 'a'). Checked in SQLite.

593. **C** — Q1 counts rows, so S2's two DB rows plus OS reach 3 and S2 is wrongly included. Q2 counts distinct courses, so S2 has only 2 and only S1 qualifies. This GROUP BY / HAVING COUNT(DISTINCT ...) pattern is SQL's way to write division.

594. **C** — Deleting department 2 cascades to emp, removing employees 11 and 12 and leaving 1 row. In project the rule is SET NULL, so all 3 rows stay but projects 100 and 101 now have NULL dept_id, leaving 1 non-NULL value.

595. **A** — 'Every employee earns more than 400' is the same as 'the lowest salary is above 400', so HAVING MIN(salary) > 400 is right. On the staff data it returns only Tech (minimum 500). Sales' minimum is 400 and Admin's is 300.

596. **B** — COUNT(*) counts Mehwish's row, giving 1. COUNT(bonus) ignores the NULL, giving 0. AVG(bonus) has no values to average, so it is NULL.

597. **D** — After DISTINCT, one dept row may stand for several table rows with different salaries, so 'sort by salary' has no single value to use. Standard SQL, and PostgreSQL, reject it. Some systems, such as SQLite, accept it and pick a value, so do not rely on them for this rule.

598. **D** — The first transaction commits row 4, giving 4 rows. The second transaction deletes everything and inserts 9, but ROLLBACK undoes both statements, restoring the 4 committed rows.

599. **C** — Q1 removes NULL-bonus rows first, so Admin, whose only row has a NULL bonus, never forms a group. Q2 groups all rows and then counts non-NULL bonuses, so Admin appears with 0. Sales and Tech have 2 non-NULL bonuses each in both queries.

600. **B** — MAX on a text column uses string order, comparing the first character first. '9' is greater than '1' and '2', so '9' is the maximum, even though 100 is the largest number. Store such values in a numeric column, or CAST them, to compare them as numbers.

601. **D** — A correlated subquery uses a value from the current row of the outer query, for example WHERE x.did = e.did. Conceptually it is re-run for every outer row. A non-correlated subquery can run once on its own, and the number of columns or its position does not decide correlation.

602. **D** — An ordinary view stores only its defining query. Each time you query the view, the DBMS evaluates that query against the current base tables, so the view always shows current data. Storing the rows physically is what a materialised view does.

603. **B** — A trigger follows the event-condition-action pattern: when the event happens on the table (and the optional condition holds), the action runs automatically. Nobody calls it directly, which is the key difference from a stored procedure.

604. **B** — Both are stored in the database server. A stored procedure is invoked by name, for example CALL update_fees(2026) in standard SQL and MySQL or EXEC in SQL Server, and it can take IN, OUT and INOUT parameters. A trigger has no caller: the INSERT, UPDATE or DELETE event fires it.

605. **A** — Both need union-compatible inputs (same number of columns with compatible types; column names do not have to match). UNION applies set semantics and removes duplicates, which usually costs a sort or hash. UNION ALL simply appends the rows, so it is faster when duplicates do not matter.

606. **D** — Joining a table to itself is a self join. You give the table two aliases, for example employee w and employee m, and join on w.manager_id = m.id. It is the standard way to read a hierarchy stored in one table.

607. **D** — A FULL OUTER JOIN returns matched pairs plus unmatched rows from both sides, padded with NULLs. LEFT keeps unmatched rows only from the left table, INNER keeps only matches, and CROSS JOIN pairs every row with every row. SQLite only added RIGHT and FULL joins in version 3.39.

608. **D** — A materialised view saves the result rows on disk, so expensive joins and aggregates are not recomputed on every read. The cost is storage and staleness: it must be refreshed when the base tables change. Hiding columns is something any ordinary view can already do.

609. **B** — WITH CHECK OPTION stops rows from 'disappearing' out of the view: any row inserted or updated through the view must still satisfy the view's WHERE condition. Setting price to 150 through this view would be rejected. SQLite does not support WITH CHECK OPTION; PostgreSQL, Oracle, MySQL and SQL Server do.

610. **C** — For each row the UPDATE changes, OLD holds the row as it was and NEW holds it as it is now, so a trigger can log OLD.salary and NEW.salary. INSERT triggers have only NEW, and DELETE triggers have only OLD.

611. **C** — EXISTS only tests for non-emptiness. Even a row made entirely of NULLs counts, which is why people write SELECT 1 inside it. That also means EXISTS is never UNKNOWN, unlike IN when NULLs are present.

612. **D** — An update to a row of an aggregated view, such as changing a department's AVG(salary), cannot be mapped back to particular base rows, so the view is not updatable. Single-table views that keep the key and use plain projection and selection usually are. SQLite views are read-only unless you add INSTEAD OF triggers.

613. **D** — Both give tied rows the same rank. With scores 90, 90, 85, RANK() gives 1, 1, 3 because two rows used up positions 1 and 2. DENSE_RANK() gives 1, 1, 2 with no gap. ROW_NUMBER() is the one that numbers tied rows differently.

614. **C** — A scalar subquery stands in for a single value, so it must return one column and at most one row. If it returns no rows the value is NULL. More than one row is a run-time error in standard SQL.

615. **D** — NATURAL JOIN matches every pair of same-named columns, whatever the declared keys, and keeps one copy of each. That is why it is risky: two unrelated columns that happen to share a name, such as 'name' or 'updated_at', silently become join conditions.

616. **B** — > ALL means 'greater than every value in the set'. Over an empty set that condition holds vacuously, so it is TRUE and every outer row qualifies. > ANY over an empty set is FALSE. SQLite does not support ANY or ALL, so test this in PostgreSQL or MySQL, or rewrite with NOT EXISTS.

617. **A** — Ali matches two enrolments (2 rows) and Sara one (1 row). Omar and Hina have no match, but a LEFT JOIN keeps them with NULL course (2 rows). Total 5. The enrolment of student 9 has no student, so a LEFT JOIN from student drops it.

618. **B** — A CROSS JOIN pairs every row of one table with every row of the other, so the result has 3 × 4 = 12 rows. Adding the counts (7) is the usual slip.

619. **C** — UNION ALL keeps all 3 + 2 = 5 rows. UNION removes duplicates across the whole result, including the repeated 'Ali' inside lahore, leaving Ali, Sara and Omar: 3 rows.

620. **A** — The subquery returns the Lahore departments, 10 and 30. IN then keeps employees in either: Ali and Sara (10) and Zain (30). Omar and Hina are in department 20, which is in Karachi.

621. **A** — The subquery averages salaries in the current employee's own department: department 10 averages (90 + 80) / 2 = 85, department 20 averages 60 and department 30 averages 40. Only Ali (90 > 85) and Hina (70 > 60) beat their department average; Zain equals his.

622. **C** — NOT EXISTS keeps a department when no employee row has its did. Departments 10, 20 and 30 all have staff; 40 (Legal) has none, so only Legal is returned. This pattern is called an anti-join.

623. **D** — An ordinary view is re-evaluated against the base table every time it is queried. By then Pen (50), Ruler (30) and Bag (now 80) are all under 100, so the count is 3. A view is not a snapshot taken at CREATE VIEW time.

624. **C** — FOR EACH ROW fires the trigger once per inserted row, not once per statement. Three rows are inserted across two statements, so n = 3 and total = 5 + 3 + 2 = 10. SQLite only supports row-level triggers; standard SQL also has FOR EACH STATEMENT, which would fire twice here.

625. **C** — Sorted by marks: 90, 90 (rank 1), 85, 85 (rank 3), 80. RANK leaves gaps, so 80 is rank 5 because four rows come before it. DENSE_RANK counts distinct values, so 80 is the third distinct score: 3. SQLite supports window functions from version 3.25.

626. **B** — The inner self join matches each worker's mgr to a staff id: Bilal → Asma, Danish → Asma and Fatima → Bilal, giving 3 rows. Asma's mgr is NULL, and NULL = 1 is never TRUE, so she has no row. A LEFT JOIN would add her with a NULL manager.

627. **C** — EXCEPT returns distinct rows of the first query that do not appear in the second. Removing 2 and 4 from {1, 2, 3, 4} leaves 1 and 3. 5 is only in b, so it is never in the result. Oracle calls this operator MINUS.

628. **C** — The subquery finds the highest salary, 90. The outer query then takes the maximum of the salaries below 90, which is 70. Because both Sara and Omar earn 90, the 'second highest distinct salary' is 70, not 90.

629. **D** — The anchor gives k = 1. Each recursive step adds 2 while the previous k is below 9: 3, 5, 7, 9. When k = 9 the condition k < 9 fails, so the recursion stops. That is 5 rows: 1 + 3 + 5 + 7 + 9 = 25.

630. **B** — The LEFT JOIN keeps unmatched departments with NULLs on the emp side, and WHERE e.eid IS NULL keeps only those. An inner join never produces the NULL row, '= NULL' is never TRUE, and moving the test into ON keeps every department.

631. **B** — In the first query the filter is part of the ON condition, so departments without a matching high earner are still kept with NULLs: IT 2 rows, HR 1, Sales 1 and Legal 1, total 5. In the second the WHERE runs after the join and removes every NULL-padded row, which turns the LEFT JOIN into an inner join: 3 rows.

632. **C** — For each student the subquery looks for a fine row with f.sid = s.id. Only Sara has one. The NULL row never equals any id, so it simply fails to match; it does not block the result. Unlike NOT IN, NOT EXISTS is safe with NULLs, so Ali and Omar are returned.

633. **B** — Joining on cust pairs each of Ali's 2 orders with each of his 3 payments: 6 rows. Each order amount appears 3 times, so SUM gives 3 × 300 = 900. This join fan-out is a classic reporting bug; aggregate each table in a subquery first, then join the totals.

634. **A** — = expects a scalar subquery, so a second row is a run-time error in standard SQL, PostgreSQL, Oracle, MySQL and SQL Server. The fix is IN. SQLite is the odd one out: it silently uses the first row (did 10) and returns Ali and Sara, which is why you should not trust SQLite for this question.

635. **A** — With ORDER BY and no frame clause, the default frame is RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW. RANGE includes peers, rows with the same day, so both day-2 rows see 10 + 20 + 5 = 35. Writing ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW would give a row-by-row total instead.

636. **B** — The LEFT JOIN produces one NULL-padded row for Legal. COUNT(*) counts rows, so a = 1. COUNT(e.eid) skips NULLs, so b = 0. To count employees per department after an outer join, count a column from the inner side.

637. **A** — The subquery has no WHERE x.did = e.did, so it is not correlated: it returns the company average, (90 + 80 + 50 + 70 + 40) / 5 = 66. Ali, Sara and Hina exceed 66. The intended per-department version returns only Ali and Hina, because Sara (80) is below her department's 85.

638. **A** — The BEFORE INSERT trigger aborts only the statement that inserts -20. In autocommit mode the first and third inserts are separate statements, so they succeed: 2 rows, 500 + 300 = 800. RAISE(ABORT, ...) is SQLite syntax; standard SQL and MySQL use SIGNAL SQLSTATE, and PostgreSQL uses RAISE EXCEPTION inside a trigger function.

639. **A** — A single-table view that keeps the key and uses only projection and selection maps each view row to exactly one base row, so updates can be passed through. DISTINCT, GROUP BY and aggregates destroy that one-to-one mapping. In SQLite even the first view is read-only unless an INSTEAD OF trigger is defined.

640. **A** — Keys 2 and 3 match (2 rows). Key 1 exists only in t1 and keys 4 and 5 only in t2; each becomes a NULL-padded row (3 rows). Total 5. A quick check: |matches| + unmatched-left + unmatched-right = 2 + 1 + 2. FULL OUTER JOIN works in SQLite only from version 3.39.

641. **A** — INTERSECT keeps only rows present in both inputs and, like UNION, removes duplicates unless INTERSECT ALL is used. The other options describe UNION ALL, EXCEPT and a CROSS JOIN.

642. **B** — Set operations are union-compatible only when both inputs have the same number of columns and each pair of positional columns has compatible types. Column names do not need to match: the result simply takes the names from the first SELECT.

643. **C** — Views are a standard security tool. A view exposing only the permitted columns (or rows) can be granted to a user who has no privilege on the base table, so the hidden columns are never reachable through it.

644. **D** — An INSTEAD OF trigger is defined on a view. When a DML statement targets the view, the DBMS skips the statement and runs the trigger body instead, which can translate the change into updates on the underlying tables. This is how many non-updatable views are made writable.

645. **A** — A stored procedure lives on the server and runs many statements in one call, reducing client-server traffic and centralising business logic. It runs only when called (unlike a trigger) and still obeys privileges and transactions.

646. **B** — IN parameters carry values into the procedure, OUT parameters carry results back to the caller, and INOUT parameters do both. This is how procedures, which have no return value like functions do, hand results back.

647. **C** — ROW_NUMBER() always numbers rows 1, 2, 3 and so on inside each partition and breaks ties arbitrarily. RANK() and DENSE_RANK() give tied rows the same number, and NTILE(1) puts every row in bucket 1.

648. **D** — PARTITION BY defines independent windows: SUM(salary) OVER (PARTITION BY dept), for example, totals each department separately. Unlike GROUP BY it never reduces the number of rows returned.

649. **A** — LAG looks back a given number of rows (one by default) within the window order and returns NULL when no such row exists. LEAD is the forward-looking twin.

650. **B** — WITH name AS (SELECT ...) defines a CTE that exists only for the one statement that follows. It makes complex queries readable and, with WITH RECURSIVE, allows recursive queries.

651. **C** — EXISTS performs a semi-join: each customer is returned once if any matching order exists. A plain INNER JOIN repeats the customer once per order, a CROSS JOIN pairs everything, and NOT EXISTS returns the customers with no orders.

652. **D** — In standard SQL the individual SELECTs of a set operation cannot carry their own ORDER BY. A single ORDER BY at the end applies to the combined result, usually referring to column names or positions of the first SELECT.

653. **A** — A FOR EACH ROW trigger fires once for every affected row, with OLD and NEW available. A FOR EACH STATEMENT trigger fires once per triggering statement, however many rows it touches, even zero.

654. **B** — BEFORE row triggers run before the row is stored, so they can validate the incoming values, raise an error, or (in most DBMSs) modify NEW. AFTER triggers see the final row and suit auditing or cascading work.

655. **C** — Functions are expressions: SELECT tax(price) FROM item works because a function returns a value. Procedures are statements run with CALL. Both are stored on the server and both accept parameters; neither fires on events like a trigger.

656. **D** — The anchor member runs once to seed the result. The recursive member then runs repeatedly on the rows produced by the previous step, and evaluation ends when a step adds nothing. A missing stop condition therefore causes an infinite loop.
657. **A** — CASCADED (the default when CHECK OPTION is given in standard SQL) checks the new row against the view and all underlying views. WITH LOCAL CHECK OPTION checks only the conditions of views that themselves carry a check option.
658. **B** — A materialised view stores its result, which makes reads fast but means base-table changes are not visible until a refresh (immediate, deferred or on demand). An ordinary view recomputes on every use, so it is always current.
659. **C** — x NOT IN (1, NULL) expands to x <> 1 AND x <> NULL. The second test is UNKNOWN, so the conjunction is at best UNKNOWN and WHERE keeps only TRUE rows. NOT EXISTS does not have this trap.
660. **D** — = ANY (also written = SOME) is TRUE if x equals at least one value, which is exactly IN. = ALL needs x to equal every value, <> ANY is TRUE whenever some value differs, and an uncorrelated EXISTS ignores x altogether.
661. **A** — NOT IN requires x to differ from every value returned, which is <> ALL. <> ANY is TRUE if x differs from just one value, so it is far weaker. Both NOT IN and <> ALL share the same NULL behaviour.
662. **B** — With ORDER BY and no frame, the default is RANGE ... CURRENT ROW, which includes every peer (row with the same ORDER BY value), so tied rows share a total. ROWS counts physical rows, so each row stops at itself.
663. **C** — Triggers cascade: a statement inside a trigger body is an ordinary DML statement and can fire other triggers. The cascade runs inside the original transaction, and DBMSs cap the nesting depth to stop infinite recursion.
664. **D** — Per-row evaluation is only the logical meaning. Optimisers routinely rewrite EXISTS, IN and many correlated scalar subqueries into joins, semi-joins or anti-joins with hash or index lookups. Ignoring the outer value would give wrong answers.
665. **A** — Bilal and Hina are in both clubs. Hina appears twice in debate, but INTERSECT removes duplicates, so the subquery has 2 rows. Ayesha and Usman are each in only one club.
666. **B** — The window restarts for each department. In Sales the salaries in descending order are Asad 90, Danish 70, Sana 50, so Danish is 2. Kamran in HR is numbered separately.
667. **C** — The LEFT JOIN keeps Usman with a NULL amount because he has no orders, and COALESCE replaces that NULL with 0. Without COALESCE the output would show NULL.
668. **D** — Marks of 60 or more are labelled High: 85 and 60, so 2. The rest (45 and 30) are Low, also 2. The boundary value 60 satisfies >= 60, which is the usual slip.
669. **A** — The CTE totals sales per city: Lahore 150, Karachi 120, Quetta 40. The outer query keeps the cities whose total is above 130, which is only Lahore.
670. **B** — LAG(rev) returns the previous month's revenue, NULL for the first month. So change is 100 - NULL = NULL, then 120 - 100 = 20, then 110 - 120 = -10.
671. **C** — ROW_NUMBER restarts per department and orders by salary descending, so rn = 1 marks the highest earner in each: Asad (90) in Sales and Kamran (80) in HR. Wrapping the window in a CTE is what allows filtering on rn.
672. **D** — The subquery groups orders by customer and keeps ids with at least two orders: 1 (two orders) and 3 (three). Customer 2 has one order. IN then matches Ayesha (1) and Zara (3).
673. **A** — The derived table sums salaries per department: IT 150, HR 90, Ops 60. The outer AVG over those three totals is (150 + 90 + 60) / 3 = 100. Averaging the five raw salaries instead would give 60.
674. **B** — A RIGHT JOIN keeps every row of the right table (course). DB matches two enrolments (2 rows), OS matches one (1 row) and AI matches none, so it appears once with NULLs. Total 4. The unmatched enrolment for ML is dropped.
675. **C** — Three rows have qty = 0. The AFTER DELETE trigger is row-level (FOR EACH ROW), so it fires once per deleted row and writes three log rows.
676. **D** — Views store no rows, so both are recomputed from the base table product. After the update Pen costs 150. v1 keeps price < 200 (Pen 150, Mouse 120) and v2 keeps those with stock > 0, which leaves Mouse, because Pen's stock is 0.

677. **A** — The anchor selects the direct reports of 1 (ids 2 and 3). The recursive member then adds their reports: 4 (under 2) and 5 (under 4). Employee 6 reports to 7, outside the chain. So 4 subordinates are found; employee 1 is not counted because the anchor starts at the reports.

678. **B** — NTILE(n) divides the ordered rows into n groups as evenly as possible, and when the rows do not divide evenly the earlier buckets get the extra row. Five rows into 2 buckets gives 3 and 2.

679. **C** — The scalar subquery counts orders whose cid matches the current customer. Bilal (id 2) has none, and COUNT over zero rows is 0, not NULL. Every customer is listed because there is no join that could drop him.

680. **D** — LEAD(score) returns the score from the following row in roll order. Roll 1 sees 70, roll 2 sees 90 and roll 3 has no next row, so it gets NULL (no default value was given).

681. **A** — The anchor row is (1, 1). Each step adds 1 to n and multiplies f by the new n while n < 5, giving (2, 2), (3, 6), (4, 24), (5, 120). MAX(f) is 5! = 120.

682. **B** — The frame covers the previous row and the current row. Day 1 has only itself: 10. Day 2: (10 + 30) / 2 = 20. Day 3: (30 + 50) / 2 = 40. Day 4: (50 + 50) / 2 = 50.

683. **C** — The INSTEAD OF trigger intercepts the insert on the view and runs its body: one row goes into dept and one into emp. Each table already had one row, so both now hold 2.

684. **D** — The query expresses division: a student is kept if there is no course they have not taken. The courses are DB and OS. Ayesha took both; Bilal took only DB; Hina took OS and AI but not DB. Only Ayesha qualifies.

685. **A** — Inserting the order fires t1, which lowers the stock of item 1 from 10 to 3. That UPDATE fires t2, whose WHEN clause (NEW.qty < 5) is true, so one row goes into reorder. Only one order is inserted, so stock is updated once and t2 fires once.

686. **B** — WHERE is evaluated row by row before any grouping, so an aggregate such as AVG cannot appear there. A non-correlated scalar subquery computes the company average once and each row is compared with it.

687. **C** — The subquery returns 10 and NULL. For HR, 20 NOT IN (10, NULL) means 20 <> 10 AND 20 <> NULL, which is UNKNOWN, so the row is not kept. The same happens to every department. Use NOT EXISTS or add WHERE did IS NOT NULL to the subquery.

688. **D** — Logical order is FROM, WHERE, GROUP BY, HAVING, window functions, SELECT, ORDER BY. The alias r and the RANK() value do not exist yet when WHERE runs, so the query is rejected. Wrapping it (WITH x AS (...) SELECT ... WHERE r = 1) is the standard fix.

689. **A** — An equi-join keeps a pair only when a.k = b.k is TRUE. 2 = 2 matches once. NULL = NULL is UNKNOWN, not TRUE, so the NULL rows never pair. The result is 1 row.

690. **B** — dept LEFT JOIN emp gives Ali (IT), Sana (IT) and a NULL row for Legal. The following INNER JOIN on project needs e.eid to match, and NULL never matches, so Legal is lost. Ali and Sana each have one project, giving 2 rows. Making the second join a LEFT JOIN too keeps Legal.

691. **C** — The subquery never refers to the outer customer, so it asks only whether any order above 500 exists. One does (Ayesha's 800), so EXISTS is TRUE for every customer and all three are returned. The fix is to add AND o.cid = c.id.

692. **D** — The UPDATE changes three rows (price < 100). A FOR EACH ROW trigger runs once per affected row, so counter becomes 3. A statement-level trigger, which SQLite does not support, would have fired once.

693. **A** — A scalar subquery that returns no rows evaluates to NULL; it is not an error. Comparing anything with NULL gives UNKNOWN, and WHERE discards UNKNOWN, so the result is empty.

694. **B** — With ORDER BY and no frame clause, the frame is RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW. The last row of that frame for marks 70 is the row itself, so LAST_VALUE returns 70. Adding ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING, or using MAX(marks) OVER (), gives 90.

695. **C** — ROW_NUMBER gives the tied Sales pair 1 and 2, so only one of them survives rn = 1: one row per department, 2 in total. RANK gives both tied rows rank 1, so the second query keeps both Sales staff plus HR's top earner: 3 rows.

696. **D** — Set operations match columns by position, not by name. SELECT * FROM b returns (city, name), so 'Karachi' lands under the first result column, name, and 'Sana' under city. Both columns are TEXT, so the query is union-compatible and runs. List b's columns explicitly as SELECT name, city to fix it.

697. **A** — For each employee the correlated subquery computes the maximum salary of that employee's own department. Asad and Bina both earn the Sales maximum of 90, and Kamran earns the HR maximum of 80, so all three are returned. Ties are not broken.

698. **B** — Bilal has two rows, both for DB, so COUNT(*) = 2 equals the number of courses and he wrongly passes. Ayesha has DB and OS. Counting distinct course codes gives Bilal 1, so only Ayesha would remain.

699. **C** — Every aggregate except COUNT returns NULL over an empty set, so the first query gives NULL. COALESCE(SUM(amount), 0) converts that NULL to 0. COUNT(*) would have returned 0 directly.

700. **D** — The anchor is Lahore. Step 1 adds Karachi, step 2 adds Quetta, step 3 would add Lahore again, but UNION removes the duplicate, so the step adds nothing new and recursion ends with 3 rows. With UNION ALL the cycle would loop forever.

701. **C** — X → Y says X determines Y: if two rows share the same X values, they must share the same Y values. It does not work backwards, and it does not need X to be unique; X only needs to be unique when X → R, which makes X a superkey.

702. **C** — 1NF is about the shape of the data: each cell holds one indivisible value, so a cell like '0300-1, 0321-2' breaks 1NF. The other options describe 2NF, BCNF and 3NF.

703. **A** — 2NF = 1NF plus no partial dependency: no non-prime attribute may be determined by a proper subset of a candidate key. In Enrol(StudentID, CourseID, CourseTitle), CourseID → CourseTitle is partial, so the relation is not in 2NF.

704. **B** — In Emp(EmpID, DeptID, DeptName), EmpID → DeptID and DeptID → DeptName, so DeptName depends on the key only through DeptID. That transitive dependency is what 3NF removes. Multivalued dependencies are 4NF and join dependencies are 5NF.

705. **A** — BCNF requires every determinant of a non-trivial FD to be a superkey, with no exceptions. 3NF adds the escape clause 'or every attribute of Y is prime', which is why 3NF is weaker than BCNF.

706. **D** — Prime means 'member of some candidate key', not only of the chosen primary key. If R has keys AB and BC, then A, B and C are all prime. Getting this right decides 2NF and 3NF answers.

707. **C** — When the same fact is stored in several rows, changing only some copies leaves the data inconsistent: an update anomaly. An insertion anomaly is being unable to add a fact without unrelated data; a deletion anomaly is losing a fact when you delete a row. 'Phantom' belongs to transactions, not design.

708. **D** — Armstrong's axioms are reflexivity, augmentation and transitivity. They are sound and complete. Union, decomposition and pseudotransitivity are derived rules that can be proved from those three.

709. **A** — The binary test: R1 ∩ R2 → R1 or R1 ∩ R2 → R2, meaning the shared attributes are a superkey of one side. Covering all attributes is necessary but not enough. Keeping each FD inside one fragment is dependency preservation, a different property.

710. **D** — A partial dependency needs a proper subset of a key, and a single-attribute key has no non-empty proper subset. So R is automatically in 2NF. It can still have transitive dependencies, so 3NF and BCNF are not guaranteed.

711. **C** — The 3NF synthesis algorithm always gives a lossless, dependency-preserving result. BCNF decomposition is always lossless but may have to give up a dependency, as with R(A, B, C) where AB → C and C → B. That trade-off is why designers sometimes stop at 3NF.

712. **D** — X → Y is trivial when Y ⊆ X, so it holds in every relation. AB → A qualifies. A → AB is not trivial because B is not in A, although it is equivalent to A → B.

713. **A** — A minimal cover must be equivalent to F, with no extraneous left-side attributes and no redundant FDs. Elmasri-Navathe also split right sides into single attributes, while Silberschatz's canonical cover merges FDs with the same left side. Left sides do not have to be keys; a cover can contain DeptID → DeptName in a relation keyed on EmpID.

714. **C** — Denormalisation adds controlled redundancy, such as storing DeptName in Emp, so that common reports avoid a join. It brings back the anomalies normalisation removed and usually uses more storage, so it is a performance trade-off, common in data warehouses.

715. **D** — The key is all three attributes, so there are no non-trivial FDs and the table is already in BCNF. The redundancy comes from the multivalued dependencies Name →→ Subject and Name →→ Hobby. 4NF removes them by splitting into (Name, Subject) and (Name, Hobby).

716. **A** — With attributes A and B, the only possible non-trivial FDs are A → B and B → A. Either one makes its left side a key of the whole relation, so every determinant is a superkey and the relation is in BCNF. If neither holds, the key is AB and there are no non-trivial FDs at all.

717. **D** — Start with {A}. A → B adds B. B → C adds C. Nothing else applies, so A^+ = {A, B, C}, which means A is a key of R. Always include the attribute itself in its own closure.

718. **C** — A never appears on a right-hand side, so it must be in every key. A^+ = {A, B, C, D}, so A alone is a key. AB and ABCD are superkeys but not minimal, and D determines nothing.

719. **B** — B is on no right side, so every key contains B. B^+ = {B}, so B needs help. AB^+ = {A, B, C, D}. BC^+: C → D, D → A, giving everything. BD^+: D → A, then AB → C, giving everything. None has a smaller key inside it, so the keys are AB, BC and BD.

720. **B** — The candidate keys are AB and AC, so A, B and C are all prime. AB → C has a superkey on the left. C → B does not (C^+ = {B, C}), so R is not in BCNF; but B is prime, so 3NF allows it. Highest normal form: 3NF.

721. **D** — The only candidate key is AB. C is non-prime and depends on A, a proper part of the key. That partial dependency breaks 2NF, so R is only in 1NF. Splitting into (A, C) and (A, B, D) fixes it.

722. **A** — StudentID is the only key and it is a single attribute, so there can be no partial dependency: 2NF holds. DeptID → DeptName has a non-key determinant and a non-prime right side, a transitive dependency, so 3NF fails. Answer: 2NF.

723. **B** — By transitivity, A → B and B → C give A → C. Dependencies do not reverse, so nothing lets you derive C → A, B → A or C → B from F.

724. **A** — Start with {B, C}. BC → D adds D, D → E adds E, and E → A adds A. That is every attribute, so BC is a key. Stopping after one or two rounds is the usual mistake; keep applying FDs until nothing changes.

725. **C** — A superkey here is any set that contains A. The other three attributes can each be in or out, giving $2^3 = 8$ superkeys: A, AB, AC, AD, ABC, ABD, ACD and ABCD. 16 counts every subset, including those without A.

726. **B** — For (A, B) and (A, C) the common attribute is A, and A → B makes A a key of (A, B), so the join is lossless. In the other options the common attribute (B, C or nothing) determines neither fragment, so the join can create spurious tuples.

727. **D** — C and D never appear on a right side, so every key contains CD, but CD^+ = {C, D}. Add one more: ACD gives B, then E; BCD gives E, then A; CDE gives A, then B. Each reaches all five attributes, and no two-attribute set does. So there are three keys.

728. **A** — Prime attributes are those in some candidate key, here A and B. Everything else, C and D, is non-prime. 2NF and 3NF rules are written in terms of non-prime attributes, so this is always the first step.

729. **B** — Split A → BC into A → B and A → C. In AB → C, A is extraneous because B → C already holds, so it reduces to B → C. A → C is then redundant (A → B → C), and duplicates go. {A → B, B → C} is left. The second option still has redundant A → C; the third loses B → C; the fourth cannot derive B → C.

730. **B** — A multi-valued attribute becomes its own relation with one row per value, keyed on (ID, Phone). Phone1, Phone2, Phone3 is a repeating group: it caps the number of phones and leaves NULLs. Keeping the list in one text cell is exactly the 1NF violation.

731. **D** — R1 ∩ R2 = C, and C → B makes C a key of R1, so the join is lossless. R1 has key C and R2 has key AC, so both are BCNF. But A, B and C never sit together in one fragment, so AB → C cannot be checked locally: it is lost. This is the textbook case where BCNF costs dependency preservation.

732. **B** — A is the only key, and it is a single attribute, so no partial dependency is possible: 2NF holds. B → C, C → D and D → E all have non-key determinants and non-prime right sides, which are transitive dependencies, so 3NF fails.

733. **C** — Losing an unrelated fact (the phone) when deleting a row (the course) is a deletion anomaly. It happens because Instructor → InstructorPhone is a transitive dependency on CourseID. Decomposing into Course(CourseID, Title, Instructor) and Instructor(Instructor, InstructorPhone) keeps the phone when the course goes.

734. **B** — The key of R is AC. (A, B) and (C, D) are each in BCNF, but on their own they share no attribute, so their join is a Cartesian product: lossy. Adding (A, C), the key, makes the chase reach a full row, so the join is lossless. (A, B, C) is lossless but not BCNF because A → B is a partial dependency.

735. **A** — The candidate keys are AB, BC and BD, so all four attributes are prime. C → D and D → A have non-superkey left sides, so R is not in BCNF. But their right sides are prime, which 3NF allows, and with no non-prime attributes 2NF holds trivially. Highest: 3NF.

736. **D** — Saad appears with DB and with AI, so the same teacher maps to two courses: teacher → course fails. Every course has one teacher and one day, and each (sid, course) pair occurs once. Note that sample data can disprove an FD but can never prove one holds in general.

737. **C** — Two sets are equivalent when each implies every FD of the other. Adding A → C changes nothing, since F already implies it. The other three all fail to imply something in F: {A → B, A → C} and {A → BC} cannot give B → C, and {B → C, A → C} cannot give A → B.

738. **A** — The key is AB. A partial dependency needs a non-prime attribute determined by A alone or B alone; A^+ = {A} and B^+ = {B}, so there is none and 2NF holds. D depends on the key through C, which is a transitive dependency, so 3NF fails. The student confused transitive with partial dependency.

739. **B** — A and B determine each other, and so do C and D. A key needs one attribute from {A, B} and one from {C, D}: AC, AD, BC and BD, four keys. No single attribute reaches all four attributes, and any three-attribute set contains one of these keys, so it is not minimal.

740. **C** — The key of R is AD, since D appears in no FD. Synthesis makes one relation per FD of the minimal cover, (A, B) and (B, C), then adds a relation holding a key, (A, D), because no relation contains one yet. The result is lossless, dependency-preserving and in 3NF.

741. **A** — An insertion anomaly occurs when a fact cannot be stored without an unrelated fact: here a department needs an employee row, and EmpID (the key) cannot be NULL. Moving (DeptID, DeptName) into its own table removes it.

742. **B** — A superkey determines the whole relation; a candidate key is a superkey with nothing removable. The designer then chooses one candidate key as the primary key. The last option describes a foreign key.

743. **C** — Augmentation adds the same attributes Z to both sides of a valid FD. Reflexivity gives X → Y when Y ⊆ X, transitivity chains X → Y and Y → Z, and decomposition splits X → YZ into X → Y and X → Z.

744. **D** — Decomposition splits the right-hand side of an FD. Union is the reverse (X → Y and X → Z give X → YZ). Both are derived rules provable from Armstrong's three axioms.

745. **A** — Normalisation splits relations using functional dependencies so that each fact lives in one place, which removes anomalies. It usually increases the number of joins, which is why designers sometimes denormalise for speed.

746. **B** — Full dependency means every attribute of X is needed. If some proper subset of X already determines Y, the dependency is partial. A single-attribute X is always full, but full dependencies can also have composite left sides.

747. **C** — StudentName depends on StudentID alone, which is only part of the composite key, so it is a partial dependency and violates 2NF. Grade needs both key attributes, and CourseID → CourseID is trivial.

748. **D** — In a transitive dependency the key determines a non-key attribute, which in turn determines another non-prime attribute. It is what 3NF removes. Dependence on part of a key is a partial dependency (2NF).

749. **A** — 3NF allows two escape routes: a superkey on the left, or a prime attribute on the right. BCNF keeps only the first, which is why BCNF is stricter.

750. **B** — Lossless (non-additive) join is about the rows: rejoining must not create extra, spurious tuples. Covering every attribute is a basic requirement of any decomposition, and checking FDs locally is dependency preservation, a separate property.

751. **C** — Dependency preservation means constraints can be enforced fragment by fragment, without joining tables. It is independent of losslessness: a decomposition can have either property without the other.

752. **D** — An MVD X ↠ Y says the set of Y values associated with an X value does not depend on the remaining attributes. It is the basis of 4NF. A single teacher per course would be the FD Course → Teacher.

753. **A** — Pseudotransitivity combines augmentation and transitivity: augment X → Y with W to get WX → WY, then chain with WY → Z. The last option is false in general, because X alone need not determine Z.

754. **B** — The only extra freedom in 3NF is the 'A is prime' clause. A relation with an FD such as C → B, where C is not a superkey but B is part of a key, passes 3NF and fails BCNF. Multivalued dependencies are the concern of 4NF.

755. **C** — X^+ is every attribute X determines. If that covers all of R, X is a superkey; if additionally no proper subset does, it is a candidate key. $X^+ = X$ means X determines nothing new.

756. **D** — 5NF requires every join dependency to be implied by the candidate keys, removing redundancy that only appears when a relation must be split into three or more projections. 4NF handles multivalued dependencies, a special binary case.

757. **A** — To test a left-side attribute, drop it and check whether the smaller left side still determines the right side, using F: compute A^+ and see whether it contains C. Removing extraneous attributes is one step of building a minimal cover.

758. **B** — A binary split is lossless if R1 ∩ R2 → R1 or R1 ∩ R2 → R2. If the shared attributes determine neither side, rows that merely share those values get paired wrongly. Losing FDs is a separate issue.

759. **C** — An M:N relationship cannot be held by a single foreign key, so it gets its own relation whose key combines both participants' keys. Relationship attributes such as Grade go there too.

760. **D** — The N side carries the foreign key, since each of its rows relates to one row on the 1 side. A list of IDs in Department would break 1NF, and a bridge table is needed only for M:N.

761. **A** — Each phone becomes its own row in StudentPhone(StudentID, Phone), keyed on both. Fixed repeating columns limit the count and waste space, and a comma-separated value is not atomic, so it breaks 1NF.

762. **B** — An MVD is trivial when it holds in every instance of R: either Y is part of X, or X and Y together cover R, leaving no 'other' attributes for Y to be independent of. 4NF requires every non-trivial MVD to have a superkey on the left.

763. **C** — If X → Y holds then X ↠ Y holds too. 4NF demands a superkey on the left of every non-trivial MVD, so in particular every non-trivial FD has a superkey on the left, which is BCNF.

764. **D** — A weak entity cannot be identified on its own. Its relation takes the owner's key as a foreign key and combines it with the partial key, so two employees can each have a dependant named Ali. EmpID alone would allow only one dependant per employee.

765. **A** — Start with {A}. A → C adds C, then C → D adds D. B is never reached, so B → E cannot fire. A^+ = {A, C, D}.

766. **B** — $(AC)^+$ = {A, B, C, D}, which contains B and D, so AC → BD holds (augmentation plus union). A^+ = {A, B} does not contain D, and FDs never reverse, so B → A and D → C do not follow.

767. **C** — A, C and E appear on no right side, so every key must contain them. $(ACE)^+$ = {A, B, C, D, E}, so ACE is the only candidate key. AC misses E, and ACD misses E, so it is not even a superkey.

768. **D** — A^+ = {A, B, C}, so A is the only candidate key. Every non-trivial FD has A, a superkey, on the left, so R is in BCNF.

769. **A** — A^+ = {A, B, C} and B^+ = {A, B, C}, so both A and B are candidate keys. Prime attributes belong to some candidate key, so A and B are prime and C is non-prime.

770. **B** — B is on no right side, so it is in every key, but B^+ = {B}. Pairing B with A, C, D or E works: each pair's closure reaches all five attributes (for example BE gives A via E → A, then C via AB → C, then D). None of these has a smaller key inside it.

771. **C** — D is on no right side, and A, B and C determine each other, so the keys are AD, BD and CD. Every attribute is prime, so 3NF holds automatically. A → B has a non-superkey on the left, so BCNF fails.

772. **D** — The common attribute of the first split is A, and A → B makes A a key of (A, B), so the join is lossless. In the others the shared attributes (C, none, D) determine neither fragment: C^+ = {C, D} and D^+ = {D}.

773. **A** — Split right sides: A → B, A → C, B → C, AB → D. In AB → D, B is extraneous because A^+ already contains B, so it becomes A → D. A → C is redundant via A → B and B → C. What remains is {A → B, A → D, B → C}.

774. **B** — A → B lives in (A, B) and C → D lives in (C, D). B and C never share a fragment, and the projected FDs (A → B, A → C, C → D) do not imply B → C, so it is lost.

775. **C** — A^+ and E^+ are both the full set, so A and E are keys. BC reaches D, then E, then A; CD reaches E, then A and B. So BC and CD are keys too, giving 4: A, E, BC and CD.

776. **D** — B → C violates BCNF, giving (B, C) and (A, B, D). In (A, B, D) the key is AD and A → B still violates BCNF, so it splits into (A, B) and (A, D). All three fragments are now in BCNF.

777. **A** — Each partial dependency gets its own table keyed by the part of the key it depends on, and the attribute that needs the full key stays with the full key. Every fragment is then in BCNF and the joins are lossless.

778. **C** — Two sets are equivalent when each implies every FD of the other. Under F, $A^+ = B^+ = C^+$ = {A, B, C}, and the same holds under G (A → C → B → A). So A → B follows from G, and every FD of G follows from F.

779. **D** — From AC: A → B adds B, C → D adds D. Now BD is present, so BD → E adds E, and E → F adds F. $(AC)^+$ is the whole relation, so AC is a superkey.

780. **A** — A is on no right side, so it is in every key, but A^+ = {A, B, C}. Adding D gives CD → E, then F. Adding E gives F, then D; adding F gives D, then E. So AD, AE and AF are all keys. AC adds nothing beyond A^+.

781. **B** — AB → E reduces to A → E, because A^+ contains B. A → E is then redundant: A → B, B → D, A → C and CD → E already give it. A → C cannot go, because without it A^+ = {A, B, D}. The result has single right sides and nothing removable.

782. **C** — Group FDs with the same left side: (A, B, C) from A, and (C, D) from C → D. D → C gives (D, C), the same schema, so it is dropped. The only key is AE (E appears in no FD), and no schema contains it, so (A, E) is added.

783. **D** — Join (A, B) and (B, C) on B, a key of (B, C), to get (A, B, C); join that with (C, D) on C, a key of (C, D). In the other options some join step shares only attributes that are a key of neither side, so a chase table never yields an all-known row.

784. **A** — Supersets of AB: $2^2 = 4$ (AB, ABC, ABD, ABCD). Supersets of CD: 4 (CD, ACD, BCD, ABCD). ABCD is counted twice, so 4 + 4 − 1 = 7.

785. **B** — An instance can refute an FD (two rows agreeing on X but not Y) but can never prove one. ID → City is plausible from the business rules, not from three rows. City → ID is refuted by the data, since Lahore maps to 1 and 2.

786. **C** — C^+ = {A, C} and D^+ = {B, D}, so neither C nor D is a superkey, which breaks BCNF. Their right sides A and B are prime, so 3NF still holds. E depends on whole keys only, so there is no 2NF problem.

787. **D** — Both fragments contain Lahore twice, so the join pairs each employee with each project: (Ali, ERP), (Ali, CRM), (Sana, ERP), (Sana, CRM). Two of these never existed. Extra rows are information loss, because you can no longer tell which pairs are real.

788. **A** — order_total is derived data: deliberate denormalisation. It speeds up reads but creates an update anomaly risk whenever a line changes and the total does not. Triggers, application transactions or a materialised view keep it in step.

789. **B** — ProductName depends on ProductID, only part of the composite key, which is a partial dependency of a non-prime attribute, so 2NF fails. Splitting off Product removes it, and both new relations are in BCNF.

790. **C** — From A you get B, but BC → D needs C as well, which A does not supply. Pseudotransitivity (A → B, BC → D) gives CA → D, so the C must stay on the left.

791. **D** — A surrogate key adds a candidate key but removes no dependency. DeptID is still not a superkey and DeptName is not prime, so the transitive dependency remains. Only moving (DeptID, DeptName) to its own table fixes it.

792. **A** — With Course ↠ Teacher and Course ↠ Book, CTB must hold every teacher-book pair: 2 × 3 = 6 rows. Split into (Course, Teacher) with 2 rows and (Course, Book) with 3 rows, the same facts need 5 rows, and adding a book now means 1 row instead of 2.

793. **B** — B^+ = {B, D}, so B is not a superkey and B → D breaks BCNF. D belongs to the key CD, so it is prime and the 3NF condition is met. The other FDs have keys (A, E, CD) on the left.

794. **C** — Join (A, B) with (B, C) on B (a key of (B, C)), then with (A, D) on A (a key of (A, B, C)), then with (D, E) on D (a key of (D, E)). Every step is lossless. The chain through (C, D) fails because C determines nothing beyond itself.

795. **D** — All four sets are equivalent to F, but a minimal cover may contain no redundant FD. In the first, A → C follows from A → B and B → C, so it is redundant. The other three have single right sides and lose equivalence if any FD is dropped, which shows minimal covers are not unique.

796. **A** — Each FD sits entirely inside one fragment, so both are preserved. With no common attribute, the join is a Cartesian product and pairs every (A, B) row with every (C, D) row. R's key AC is in no fragment, which is why 3NF synthesis would add (A, C).

797. **B** — A must be in every key, but D, E and F form a cycle (CD → E, E → F, F → D, with C from A), so any one of them completes the key: AD, AE and AF. The no-right-side rule tells you what every key must contain, not that the first key found is the only one.

798. **C** — C → A has part of a key on the left, but its right side A is prime (it is in key AB). 2NF and 3NF only restrict non-prime right sides, so both hold. D depends on full keys only. C is not a superkey, so BCNF fails.

799. **D** — D → B makes D a key of (B, D), so the join is lossless. (B, D) has key D, and (A, C, D) has keys A and CD with only A → CD and CD → A inside it, so both are BCNF. B and C never share a fragment, so BC → A cannot be checked locally.

800. **D** — Atomicity means all of a transaction's actions happen or none do. The half-finished transfer is rolled back using the log, so the debit disappears. Durability is about keeping committed work, and this transaction never committed.

801. **C** — Durability means committed changes survive crashes. The recovery manager ensures it by forcing log records to stable storage at commit and redoing committed work after a failure. Concurrency control provides isolation instead.
802. **D** — Isolation means each transaction behaves as if it ran alone, even when many run at once. Locking, timestamp ordering and similar protocols enforce it by allowing only safe interleavings. Atomicity and durability belong to the recovery component.
803. **C** — Consistency means a transaction run alone on a consistent database leaves it consistent, such as a transfer keeping the total balance unchanged. The DBMS cannot know every business rule, so the programmer must code it correctly, with declared constraints catching some errors. The other ACID properties are enforced by DBMS components.
804. **B** — The states are active, partially committed (last statement done, output may still be in memory), committed, failed and aborted. A transaction becomes committed only once its commit record is safely in the log, so a failure here can still abort it.
805. **C** — Swapping two conflicting operations can change the result, which is why their order matters. Two reads never conflict, operations on different items never conflict, and operations of the same transaction keep their program order anyway.
806. **C** — In a serial schedule there is no interleaving: T1 completes, then T2, and so on, in any order. Serial schedules are correct by definition, and serialisability asks whether an interleaved schedule is equivalent to one of them.
807. **D** — If Tj committed first and Ti later aborted, Tj would have committed a value that never officially existed, with no way to undo it. Recoverability just orders the commits. Requiring Ti to commit before Tj even reads the item is the stronger cascadeless condition.
808. **C** — Cascadeless means 'avoids cascading aborts'. If every read sees committed data, aborting a transaction never invalidates what another one read. It says nothing about serialisability, and transactions can still abort for other reasons.
809. **D** — A dirty read sees uncommitted data, which may be rolled back later. Getting two different committed values is a non-repeatable read, new matching rows is a phantom, and one write overwriting another based on a stale read is a lost update.
810. **A** — In the SQL standard, READ UNCOMMITTED allows dirty reads; READ COMMITTED blocks them but allows non-repeatable reads and phantoms; REPEATABLE READ also blocks non-repeatable reads; SERIALIZABLE blocks all three. Real systems differ in detail, for example PostgreSQL's REPEATABLE READ also blocks phantoms.
811. **B** — Conflict equivalence implies view equivalence, so CSR is a subset of VSR. The extra VSR schedules all contain blind writes, which are writes with no earlier read of the item by that transaction. Testing VSR is NP-complete, which is why systems rely on CSR.
812. **A** — Strictness blocks both reads and overwrites of uncommitted data. That makes undo simple: restore the before-image. Holding exclusive locks until commit, as strict two-phase locking does, is one way to produce strict schedules, but it is a protocol rather than the definition.
813. **A** — Once rollback has restored the database, Silberschatz gives two options: restart the transaction if the failure was not caused by its own logic (for example a deadlock or hardware fault), or kill it if the logic itself is wrong. It never resumes halfway or keeps partial changes, because that would break atomicity.
814. **B** — A phantom appears when a re-run predicate query sees rows that another committed transaction inserted or deleted. Locking only the rows read the first time cannot stop it; you need predicate, index-range or table locking, or the SERIALIZABLE level.
815. **B** — Strict forbids reading and overwriting uncommitted data; cascadeless forbids only reading it; recoverable only orders commits. Each condition implies the next, so strict schedules are cascadeless and cascadeless schedules are recoverable. W1(A) W2(A) C1 C2 is cascadeless but not strict.
816. **D** — A serial schedule is just an ordering of the whole transactions, so there are $3! = 3 \times 2 \times 1 = 6$. In general, n transactions give n! serial schedules.
817. **C** — Choose which 2 of the 5 positions belong to T2; the order inside each transaction is fixed. That is $C(5, 2) = 5! / (3! \times 2!) = 10$. $5! = 120$ would allow each transaction's operations to be reordered.
818. **D** — R2(A) comes before W1(A): different transactions, same item, one write, so T2 → T1. W2(B) before R1(B) gives T2 → T1 again. R1(A) and R2(A) are both reads, so they do not conflict. The graph has a single edge T2 → T1.

819. **A** — Every conflict on A and on B has T1's operation first (for example W1(A) before R2(A), W1(B) before R2(B)), so all edges go T1 → T2. The graph is acyclic and its only topological order is T1, T2.
820. **A** — R2(A)–W1(A) forces T2 before T1, while R1(A)–W2(A) and W1(A)–W2(A) force T1 before T2. The cycle T1 → T2 → T1 means no serial order matches. This is the lost-update pattern: T1's write is overwritten by T2, which read the old value.
821. **C** — R3(A) before W1(A) gives T3 → T1. R1(B) before W2(B) gives T1 → T2. R2(C) and W3(D) touch different items, so they do not conflict. The only topological order of T3 → T1 → T2 is T3, T1, T2.
822. **C** — Both read 100. T1 writes 90, then T2 writes 100 + 20 = 120, overwriting T1's change. Any serial order gives 110, so T1's update is lost. A schedule that is not conflict serializable caused it.
823. **D** — The first transaction commits, moving 100 from account 1 to account 2: 400 and 400. The second transaction's changes are undone by ROLLBACK, so they never become visible. SQLite writes BEGIN where standard SQL writes START TRANSACTION; both are accepted by most systems.
824. **C** — T2 reads the A that T1 wrote, so T2 depends on T1. T2 commits first; if T1 then aborted, T2's committed result would rest on a value that never existed and could not be undone. That violates recoverability, so the schedule is not cascadeless or strict either.
825. **A** — T1 commits before T2, the transaction that read from it, so the schedule is recoverable. But T2 reads A before T1 commits, a dirty read; if T1 aborted, T2 would have to roll back too. So it is not cascadeless, and therefore not strict.
826. **B** — T1 reads A, T2 changes A and commits, then T1 reads A again and sees a different committed value. That is a non-repeatable read. It is not dirty, because the second read sees committed data, and no rows are inserted, so it is not a phantom.
827. **B** — W1(A) before R2(A) gives T1 → T2, and W1(B) before R3(B) gives T1 → T3. W2(C) and R3(D) use different items, so T2 and T3 are unordered. The topological orders are T1, T2, T3 and T1, T3, T2: two serial schedules.
828. **B** — The conflicting pairs are R1(A)–W2(A), W2(A)–W1(A) and R2(B)–W1(B). R1(A) and W1(A) belong to the same transaction, so they do not count. Three pairs, and they produce edges both ways between T1 and T2, so the schedule is not conflict serializable.
829. **D** — There are C(4, 2) = 6 interleavings. In any non-serial one, each transaction reads A before the other writes it, giving edges both ways and a cycle. Only the two serial schedules are conflict serializable.
830. **A** — R1(A)–W2(A) gives T1 → T2 and W2(A)–W1(A) gives T2 → T1: a cycle, so it is not CSR. But it is view equivalent to T1, T2, T3: T1 reads the initial A in both, and T3 makes the final write in both. W2(A) and W3(A) are blind writes, which is what makes the gap possible.
831. **D** — The only conflict is W1(A)–R2(A), so the graph is T1 → T2 with no cycle: conflict serializable. But T2 reads from T1 and commits before T1 does, so it is not recoverable. Serialisability and recoverability are separate checks.
832. **A** — T2 read A written by T1, so T2 must roll back. T3 read B written by T2, so T3 must go too. T4 only read C, which no aborted transaction wrote, so it is unaffected. This chain is a cascading rollback, which cascadeless schedules avoid.
833. **A** — All conflicts go T1 → T2, so it is conflict serializable. T2 reads A before T1 commits, a dirty read, so it is not cascadeless. T1 commits before T2, so it is still recoverable.
834. **D** — Seeing the same value on re-reading means preventing non-repeatable reads, which REPEATABLE READ does. Phantoms, the new customers, are tolerated, so SERIALIZABLE is stronger than needed and would cost concurrency.
835. **A** — There are no reads, so nothing reads uncommitted data: the schedule is cascadeless and therefore recoverable. But W2(A) overwrites A while T1 is uncommitted, which strictness forbids. If T1 aborted, restoring A's before-image would wipe out T2's write.
836. **B** — In R1 W1 R2 W2, T2 reads T1's result 90 and writes 110, the same as the serial order T1, T2. In the other three, both transactions read 100 before either writes, so the later write wipes out the earlier one, ending at 120 or 90.
837. **B** — R1(A)–W3(A) gives T1 → T3 and R2(B)–W3(B) gives T2 → T3. T1 and T2 share no item, so they can go in either order. Both T1, T2, T3 and T2, T1, T3 are valid.
838. **B** — Once COMMIT succeeds, the change must survive any later failure. Losing it means the log, including the commit record, was still in volatile memory, which breaks the write-ahead logging rule. Nothing was half applied, so atomicity held.

839. **C** — S1's only conflict is W1(A) before R2(A). The first option just swaps W2(B) with R1(A), which touch different items, so every conflicting pair keeps its order. The others put R2(A) before W1(A), reversing that conflict.
840. **B** — When normal execution can no longer proceed, an active transaction moves to the failed state. Only after its changes have been rolled back does it move on to the aborted state.
841. **B** — Committed is a final state: once a transaction commits, its effects are permanent and can only be reversed by a separate compensating transaction. Partially committed can still fail (for example if the log cannot be written), and active can fail at any time.
842. **A** — While one transaction waits for disk I/O, another can use the CPU, so more transactions finish per second and short transactions are not stuck behind long ones. Concurrency is what creates the need for serializability control; it does not provide it.
843. **B** — A blind write such as W2(A) with no earlier R2(A) writes a value that does not depend on what was in A. Blind writes are what allow some schedules to be view serializable without being conflict serializable.
844. **D** — Consistency says that a transaction run in isolation takes the database from one consistent state to another, here keeping A + B constant. Atomicity is about all-or-nothing execution, isolation about concurrent runs and durability about surviving crashes.
845. **A** — In autocommit mode every statement is wrapped in its own transaction and committed on success. To group several statements into one atomic unit you must turn autocommit off or start an explicit transaction with BEGIN.
846. **D** — If T2 has read a value written by uncommitted T1 and T1 aborts, T2 must also roll back, and so must anyone who read from T2. Cascadeless schedules prevent this by allowing reads of committed data only.
847. **A** — REPEATABLE READ stops a row you already read from changing, but new rows matching your predicate can still appear. Only SERIALIZABLE rules out phantoms as well.
848. **A** — Stable storage is an idealised storage that never loses data; in practice it is approximated by keeping copies on several non-volatile devices. Main memory, caches and the buffer pool are volatile and are lost on a crash.
849. **C** — An aggregate that mixes some values from before an update and some from after gives a total that matches no consistent state. Elmasri-Navathe call this the incorrect summary (inconsistent analysis) problem.
850. **D** — All updated pages are first flushed in the new copy. Then the db-pointer is overwritten atomically on disk to point to the new copy, and that single write is the commit point. A crash before it leaves the old copy in force.
851. **B** — The SQL standard specifies SERIALIZABLE as the default. Many products such as PostgreSQL and Oracle default to READ COMMITTED for speed, which is why this is a common trap.
852. **B** — Conflict equivalence requires the same operations and the same relative order for every conflicting pair. One schedule can then be turned into the other by swapping adjacent non-conflicting operations. Reads-from plus final writes defines view equivalence instead.
853. **D** — View equivalence uses three conditions: the same initial reads, the same reads-from pairs and the same final writes. Requiring the same order of conflicting pairs is the stronger conflict-equivalence condition.
854. **D** — A precedence-graph cycle check runs in about O(n squared) time for n transactions. Deciding view serializability is NP-complete. Conflict serializability is the stricter class, so it accepts fewer schedules, not more.
855. **D** — No transaction touches an item until the previous transaction has committed, so the strictness condition holds. Every strict schedule is cascadeless and every cascadeless schedule is recoverable.
856. **C** — If T2 reads T1's uncommitted write and commits first, then T1 aborts, T2's result is based on a value that never officially existed. T2 cannot be rolled back because it already committed. Recoverability is independent of serializability, so no precedence-graph cycle is implied.
857. **C** — A savepoint marks a point inside a transaction. Rolling back to it discards only the work done after that point. The transaction is still open and must later end with COMMIT or ROLLBACK.
858. **C** — For example, A := A + 10 and A := A x 2 both give 20 when A starts at 10, but differ for any other starting value. Conflict and view equivalence depend only on the reads and writes, so they hold for every initial state.
859. **A** — At the commit point every operation has succeeded and is in the log, so the commit record can be written and the transaction is committed. The changed data pages themselves may reach disk later; the log allows them to be redone after a crash.

860. **A** — Two transactions can each read overlapping data from their snapshots and then update different items, so first-committer-wins never sees a conflict, yet the result matches no serial order. Snapshots prevent dirty and non-repeatable reads, and first-committer-wins stops lost updates on the same item.
861. **D** — In a strict schedule nobody else has read or written an item since the aborting transaction wrote it, so putting back the old value (before-image) is always correct. In non-strict schedules another transaction's later write could be destroyed by that restore.
862. **A** — Even READ UNCOMMITTED forbids a dirty write, meaning a write to an item that another uncommitted transaction has already written. Dirty reads are allowed at READ UNCOMMITTED, and only SERIALIZABLE prevents phantoms.
863. **D** — The precedence test treats every read and write as opaque, so it cannot see that increments commute. Conflict serializability is therefore sufficient for correctness but not necessary.
864. **D** — Choose which 2 of the 6 positions belong to T2: (4 + 2)! / (4! x 2!) = 720 / 48 = 15. 720 is 6!, which ignores the fixed order inside each transaction.
865. **C** — Both read 50. T2 writes 60, then T1 writes 50 x 2 = 100, overwriting T2's update (a lost update). The serial orders would give 110 (T1 then T2) or 120 (T2 then T1).
866. **C** — T2 read the uncommitted value 200, so B = 250. Rolling back T1 restores A but cannot change B, which T2 already committed. Reading uncommitted data is a dirty read, and this schedule is also not recoverable.
867. **C** — T1 and T2 both read the same A, T2 writes and commits, then T1 writes a value based on its old read. T2's committed update disappears, which is a lost update. No uncommitted data is read, so it is not a dirty read.
868. **D** — W1(A) and R2(B) touch different items, so they do not conflict and can be swapped. The other three pairs act on the same item and include a write, so they conflict and their order matters.
869. **A** — The ROLLBACK undoes both the insert of 2 and the delete of 1, so only row 1 remains. The insert of 3 then commits on its own. The table holds rows 1 and 3, so COUNT(*) is 2.
870. **C** — The count is 6! / (2! x 2! x 2!) = 720 / 8 = 90. 720 ignores the fixed order inside each transaction, and 6 counts only the serial schedules.
871. **D** — R3(Y) before W2(Y) and W1(Y) gives T3 -> T2 and T3 -> T1. W2(Y) before W1(Y) gives T2 -> T1. R1(X) and R3(X) are both reads, so no edge. The only topological order is T3, T2, T1.
872. **A** — R1(A)-W2(A) gives T1 -> T2, R2(B)-W3(B) gives T2 -> T3, and R1(C)-W3(C) gives T1 -> T3. That is 3 edges and no cycle, so the schedule is equivalent to T1, T2, T3.
873. **A** — On A: W1(A)-R2(A), W1(A)-W3(A) and R2(A)-W3(A). On B: W2(B)-R1(B). R2(B) and W2(B) are in the same transaction, and R2(B)-R1(B) are two reads, so neither counts. Total 4.
874. **C** — T2 reads from T1 and T3 reads from T2, so T1 must commit before T2, and T2 before T3. Only C1 C2 C3 meets both conditions.
875. **A** — 10 - 3 = 7, then the savepoint is set. The - 5 is undone by ROLLBACK TO s1, so qty goes back to 7. Adding 1 gives 8, which COMMIT makes permanent.
876. **B** — READ COMMITTED only guarantees that each read sees committed data at the time of that read. T2 committed 4 before T1's second read, so T1 sees 4. Preventing this needs REPEATABLE READ or higher.
877. **A** — T2 reads A after the debit (80) but B before the credit (100), so it prints 180. Any serial order would print 200. This is the incorrect summary problem, and the schedule is not conflict serializable.
878. **B** — 6! / (1! x 2! x 3!) = 720 / 12 = 60. 720 is 6!, 6 is the number of serial schedules, and 36 is not given by any of the standard formulas.
879. **B** — The edges are T1 -> T2, T1 -> T3 and T2 -> T3 on A, plus T3 -> T1 on B. T1 -> T3 -> T1 is a cycle, so no serial order exists. A single edge such as T2 -> T3 can never form a cycle by itself.
880. **C** — The only edges are T1 -> T2 and T3 -> T4. Any order that keeps T1 before T2 and T3 before T4 works: 4! / (2! x 2!) = 6 orders. 24 ignores the edges and 2 assumes the two pairs must stay together.
881. **C** — R2(A) reads the initial A, so T2 must come before T1 (the only other writer before it). T3 performs the final write, so T3 must be last. That leaves T2, T1, T3. The blind writes of T1 and T3 are why a non-CSR schedule can still be view serializable.
882. **D** — There are 4! / (2! x 2!) = 6 interleavings. Only two place W2(A) between R1(A) and W1(A), which creates T1 -> T2 and T2 -> T1. R2(B) touches no item of T1, so the other 4 are conflict serializable.
883. **B** — T3 must be last because T2 -> T3 and T1 -> T3, and T2 must follow both T1 and T4. T1 and T4 can come in either order, giving T1, T4, T2, T3 and T4, T1, T2, T3.

884. **B** — In the first schedule the edges are T1 -> T2 (A), T1 -> T3 (C) and T2 -> T3 (B), which is acyclic and equivalent to T1, T2, T3. Each of the others has T1 -> T2 on A and T2 -> T1 on B, a two-node cycle.
885. **C** — Seeing data that was later rolled back is a dirty read, which only READ UNCOMMITTED permits. READ COMMITTED is the weakest level that prevents it.
886. **D** — Serializability and recoverability are separate ladders. W1(A) R2(A) C2 C1 is conflict serializable (T1 then T2) but not recoverable, so if T1 aborts, committed T2 has read a value that never existed.
887. **C** — The cycle T1 -> T2 -> T1 makes it non-conflict-serializable (it is not view serializable either). But W1(A) happens only after T2 commits and nobody reads dirty data, so the schedule is strict. The two ladders are independent.
888. **B** — The danger is a phantom: another transaction inserts a row matching 'Hall 3'. Only SERIALIZABLE prevents phantoms in the SQL standard.
889. **B** — Cascadeless means every read sees committed data. Only when C1 comes before R2(A) does T2 read a committed value.
890. **B** — Reading one side of a transfer before it happens and the other side after is the incorrect summary problem. The stored data is fine; the audit saw an inconsistent mix. Running the audit at a level that prevents it, or on a snapshot, fixes it.
891. **D** — S1's conflicting pairs are W1(A) before R2(A) and R2(B) before W1(B). Swapping the adjacent non-conflicting W1(A) and R2(B) keeps both orders, so R2(B) W1(A) R2(A) W1(B) is conflict equivalent.
892. **C** — In R1(A) R3(B) W2(A) W2(B), T1 reads A before T2 writes it (T1 -> T2) and T3 reads B before T2 writes it (T3 -> T2). T1 and T3 share no item, so there are no other edges.
893. **A** — In the first interleaving A sees T1 then T2 (A = 220) but B sees T2 then T1 (B = 210). Its precedence graph has T1 -> T2 and T2 -> T1, so it matches no serial order.
894. **A** — Both T2 and T3 read A from T1 before T1 ended. T3 can be rolled back in a cascade, but T2 committed after reading dirty data, which is exactly what recoverability forbids.
895. **D** — Without a commit record on stable storage the transaction did not commit. Atomicity requires the recovery manager to undo all its changes, and the user was never told it succeeded.
896. **B** — The edges are T1 -> T2 (A), T2 -> T1 (B), T2 -> T3 (C) and T3 -> T2 (D). Both cycles pass through T2, so removing T2 is the only single removal that breaks them both.
897. **A** — Conflict equivalence forces T1, T2, T3, T4. For view equivalence only two things matter: T4 reads A from T3, and T3 makes the final write. So T2, T1, T3, T4 also works, giving 2 view-equivalent orders.
898. **B** — The first schedule has the cycle T1 -> T2 -> T1 on A. But nobody reads A, and T3 writes A last, so any order ending with T3 is view equivalent, such as T1, T2, T3. Its blind writes make it view serializable.
899. **C** — In the first schedule W2(A) overwrites A while T1, its writer, is uncommitted, which strictness forbids. Nobody reads dirty data, so it is cascadeless, and the single edge T1 -> T2 makes it conflict serializable.
900. **B** — In the S/X compatibility matrix only S with S is compatible: several transactions may read an item together. Any combination involving an exclusive (X) lock conflicts, because a writer needs sole access to the item.
901. **A** — Concurrency control makes interleaved transactions behave as if they ran one after another, which is the isolation property. Atomicity and durability are provided by the recovery manager (logs, undo and redo), and a transaction's own logic is the programmer's job.
902. **B** — 2PL splits every transaction into a growing phase, where it may obtain locks but release none, and a shrinking phase, where it may release locks but obtain no new ones. Once it releases its first lock it can never lock again.
903. **C** — In a lost update, both transactions read the old value, and the second write is based on that stale value, so the first update vanishes. The other options describe the dirty read, the unrepeatable read and the incorrect summary problem.
904. **B** — T2 has read a value that never officially existed, because T1 rolled it back. Elmasri-Navathe call this the temporary update or dirty read problem. It is why strict 2PL holds exclusive locks until commit.
905. **B** — An edge Ti -> Tj in the wait-for graph means Ti waits for a lock Tj holds. A cycle means each transaction waits for the next one round the loop, so none can proceed: a deadlock. The system breaks it by rolling back a victim. A cycle in the precedence graph is a different test, for serialisability.

906. **A** — Basic 2PL guarantees conflict serialisability: transactions can be serialised in the order of their lock points. It does not prevent deadlock (two transactions can each hold a lock the other wants), and it does not prevent cascading rollback, which needs strict 2PL.
907. **A** — Strict 2PL keeps every exclusive (write) lock until the transaction commits or aborts. Nobody can read or overwrite uncommitted data, so the schedules are strict and cascadeless. Acquiring all locks up front is conservative 2PL, which is the variant that prevents deadlock.
908. **C** — Conservative 2PL makes a transaction predeclare and lock every item it needs before it starts. If any lock is unavailable it takes none and waits, so it never holds one lock while waiting for another, and deadlock is impossible. Basic, strict and rigorous 2PL can all deadlock.
909. **A** — Under rigorous 2PL no lock is released before commit, so a transaction that commits first has finished all its conflicting accesses before any later conflicting transaction could lock those items. The equivalent serial order is therefore the commit order (Silberschatz).
910. **A** — The lock point is the end of the growing phase, when the transaction obtains its last lock. Ordering transactions by their lock points gives a serial schedule that is conflict equivalent to the 2PL schedule.
911. **D** — Basic TO assigns each transaction a timestamp at start and only allows conflicting operations in timestamp order. Any operation that would break that order causes a rollback, so the result is conflict equivalent to the serial order of increasing timestamps.
912. **D** — If a newer transaction has already written X, an older write to X is obsolete and would be overwritten anyway, so Thomas' rule simply skips it. A write that a newer transaction has already read (TS(T) < R-timestamp(X)) must still cause a rollback. The resulting schedules are view serialisable but not always conflict serialisable.
913. **C** — In wound-wait an older transaction 'wounds' (rolls back) a younger lock holder, which is preemption. Wait-die is non-preemptive: an older requester waits, while a younger requester 'dies' (rolls itself back). So in wait-die only older transactions wait, and in wound-wait only younger ones do.
914. **D** — Both schemes favour older transactions. Because a restarted transaction keeps its original timestamp, it grows relatively older as others finish, until it is the oldest and can no longer be rolled back. A new timestamp on each restart would let it starve.
915. **A** — IX means 'I intend to write somewhere below'. It is compatible with IS and IX, since both only announce finer locks. S, SIX and X would lock the whole subtree in a way that conflicts with a pending exclusive lock below, so they are incompatible with IX.
916. **D** — S is compatible with S, so T2 shares A with T1. An X lock is incompatible with any lock held by another transaction, so T3 waits until both T1 and T2 release A.
917. **D** — In wait-die an older requester is allowed to wait for a younger holder. TS 5 < TS 12, so T1 is older and waits. Only a younger requester dies.
918. **D** — In wound-wait an older requester never waits for a younger one: it wounds the holder. T1 (TS 5) is older than T2 (TS 12), so T2 is rolled back and T1 proceeds.
919. **B** — T3 (TS 20) is younger than T1 (TS 8). In wait-die a younger requester dies. It restarts with its original timestamp 20, which is what prevents starvation.
920. **C** — In the first sequence the last lock, X(B), is acquired before the first unlock, U(A), so the transaction has a clean growing then shrinking phase. Each of the others unlocks A and then requests a lock on B, which breaks 2PL.
921. **A** — Strict 2PL needs 2PL plus exclusive locks held until commit. The first sequence locks everything before its first unlock and releases X(A) only after C; releasing the shared lock on B early is allowed. The second releases X(A) before commit, and the last two unlock and then lock again.
922. **C** — r2(X) succeeds and sets R-timestamp(X) to 10. T1 then tries to write X, but a younger transaction (TS 10) has already read X, so T1's write comes too late. Since TS(T1) = 5 < R-timestamp(X) = 10, T1 is rolled back. Thomas' write rule does not help here, because it only covers TS(T) < W-timestamp(X).
923. **D** — r1(X) sets R-timestamp(X) to 3, and r3(X) raises it to max(3, 5) = 5. w2(X) is allowed because 7 is not less than R-timestamp 5 or W-timestamp 0, so W-timestamp(X) becomes 7. R-timestamp is the largest timestamp of any successful reader.
924. **C** — After w2(X), W-timestamp(X) = 9. T1's write has TS 4 < 9, so it is obsolete: in timestamp order T2's value would overwrite it anyway. Thomas' rule skips the write and lets T1 go on. Basic TO without the rule would roll T1 back.

925. **C** — r1(A) before w2(A) gives T1 -> T2, and r2(A) before w1(A) gives T2 -> T1. The precedence graph has a cycle, so no serial order is conflict equivalent. This is the lost-update pattern.
926. **B** — Conflicts on A: w1 before r3 (T1 -> T3), w1 before w2 (T1 -> T2) and r3 before w2 (T3 -> T2). r2(B) and w3(C) conflict with nothing. The only topological order is T1, T3, T2.
927. **A** — The wait-for graph has edges T1 -> T2, T2 -> T3, T3 -> T1 (because T3 waits for A, held by T1) and T4 -> T1. The cycle is T1 -> T2 -> T3 -> T1. T4 is blocked, but it is not on the cycle: it would proceed once the deadlock is broken.
928. **D** — Before locking a node in X (or IX) mode, a transaction must hold IX or SIX on its parent, all the way from the root. So it takes IX on the database, IX on F and then X on r. IS on the ancestors only permits S or IS locks below.
929. **B** — Both transactions read 100. T1 writes 70, then T2 writes 100 + 50 = 150, wiping out T1's update. Any serial order would give 120. This is the lost update problem, and 2PL would prevent this interleaving.
930. **D** — The precedence graph has T1 -> T2 and T3 -> T1, so the only serial order is T3, T1, T2. Under 2PL, T1 must unlock X before w2(X), so it must already hold its lock on Y by then. But w3(Y) happens after w2(X) and before w1(Y), while T1 would hold Y. So 2PL schedules are a strict subset of conflict-serialisable ones.
931. **C** — Track the stamps: r1(A) R(A)=10; w2(A) W(A)=20; r3(A) R(A)=30; w1(B) W(B)=10; r2(B) R(B)=20; w3(B) W(B)=30. Finally w1(A) has TS 10 < R-timestamp(A) = 30, so T1 is rolled back. All earlier operations pass.
932. **B** — Wait-die: T1 is older than holder T2, so it waits; T3 is younger than T2, so it dies. Wound-wait: T1 is older than T2, so T2 is wounded and T1 gets A; then T3, younger than the new holder T1, waits.
933. **A** — T2 reads A written by T1 and commits first. If T1 then aborted, T2's committed result would rest on a value that never existed, and it could not be undone. That makes the schedule non-recoverable, even though it is conflict equivalent to T1, T2.
934. **D** — T2 commits after T1, the transaction it read from, so the schedule is recoverable. But T2 read A before T1 committed, so if T1 aborted, T2 would have to be rolled back too: a cascading rollback. Hence it is not cascadeless.
935. **B** — The only conflict is r1(A) before w2(A), so T1 must precede T2. T3 touches only B and can sit in any of three positions: T3 T1 T2, T1 T3 T2 or T1 T2 T3. That gives 3 conflict-equivalent serial schedules.
936. **C** — T1 gets X(A); T2 and T1 both get S(B). T2's X(A) waits for T1. T1's upgrade of B to X waits for T2's shared lock. The wait-for graph has T1 -> T2 -> T1, a deadlock. Strict 2PL does not prevent deadlock.
937. **C** — In r1(X), w2(X), w1(X), the last write has TS(T1) < W-timestamp(X) = TS(T2), while R-timestamp(X) = TS(T1) is not larger than TS(T1). Basic TO rolls T1 back; Thomas' rule ignores the obsolete write. Late reads and writes that a younger transaction already read are rejected by both.
938. **A** — There are two cycles: T1 -> T2 -> T3 -> T1 and T3 -> T4 -> T3. T3 is the only transaction on both, so rolling it back clears the graph. Real systems also weigh cost (work done, locks held) when picking a victim.
939. **B** — After both reads, T1 and T2 each hold S(A). T1's upgrade waits for T2, and T2's upgrade waits for T1, giving a cycle. 2PL turns the lost update into a deadlock, which the system resolves by rolling one back. Taking X(A) at the start (update locks) avoids this.
940. **A** — The transaction first reads and computes into local copies (read phase), then is checked for conflicts (validation phase), and only if it passes are its updates copied to the database (write phase). Validating after writing would be too late to avoid damage.
941. **B** — Optimistic control lets transactions run without locks and checks only at the end. When conflicts are rare almost everyone passes validation, so there is little waiting and few restarts. With heavy contention many transactions fail validation and redo their work.
942. **A** — Locking single records lets many transactions work on different records of the same table, but each lock must be requested, stored and released. Locking a whole table is cheap to manage but blocks everyone else who needs any part of it.
943. **B** — If everyone locks A before B before C, no transaction can hold a later item while waiting for an earlier one, so a circular wait is impossible. This is a deadlock-prevention technique, not detection.
944. **B** — Victim selection aims at minimum cost: how long the transaction has run, how many items it has used and how many more it needs. Rolling back a transaction outside the cycle would not break the deadlock.

945. **D** — Conservative 2PL avoids deadlock by taking all locks up front, which means the transaction must predeclare everything it will touch. That is often unknown in advance, and holding all locks from the start also reduces concurrency.
946. **D** — The lock table keeps, for each locked item, a list of granted locks and waiting requests. A conflicting request is appended to the queue and granted later when the holders release their locks. Rolling back the holder happens only in schemes such as wound-wait.
947. **C** — When a transaction holds thousands of row locks, the DBMS may swap them for a single table lock to save lock-table space. Converting S to X on one item is a lock upgrade (conversion), not escalation.
948. **A** — An upgrade acquires a stronger lock, so it counts as acquiring and must happen in the growing phase. A downgrade gives up some rights, so it counts as releasing and belongs to the shrinking phase.
949. **D** — SIX = S + IX. The holder reads the whole subtree under the S part and can request X locks on individual descendants under the IX part. A typical use is scanning a file while updating a few records.
950. **C** — A node can be locked only if its parent is already held in a suitable intention mode, so locking goes top-down. A node can be unlocked only when none of its children is still locked by the transaction, so unlocking goes bottom-up.
951. **C** — S lets the holder read the whole subtree. Another reader (S) or intended reader below (IS) is fine. IX, SIX and X would allow writes inside that subtree, so they conflict.
952. **D** — Index locking makes a transaction lock the index leaf nodes covering the range it searched. Any insert into that range must update those leaves and therefore conflicts. Predicate locking locks the condition itself. Deadlock schemes and Thomas' rule do nothing for phantoms.
953. **A** — By refusing to let a new S request jump ahead of an older waiting X request, the queue is served fairly and T1 eventually gets its lock.
954. **C** — A timeout cannot tell a long wait from a true deadlock. Too short a limit wastes work by aborting innocent transactions; too long a limit leaves deadlocked transactions stuck.
955. **C** — Partial rollback keeps more of the victim's work but needs extra bookkeeping about which locks were taken when. Total rollback simply aborts and restarts the transaction.
956. **A** — After its lock point a transaction may unlock X(A) and keep working. Another transaction can then read A, and if the first one later aborts, the reader must roll back too. Strict 2PL fixes this by holding X locks until commit.
957. **B** — The reader gets the version with the largest write timestamp not greater than its own timestamp, so a read never waits or fails. Rollbacks happen only on writes that would invalidate a read already made by a younger transaction.
958. **D** — At commit, the system checks whether any concurrent transaction has already committed a write to the same item. If so, the committing transaction is aborted, which prevents lost updates.
959. **C** — SIX reads the whole subtree and may write parts of it. Another transaction may only intend to read lower down (IS). IX would allow writes that clash with the S part, and S would clash with the IX part.
960. **A** — Either the two did not overlap at all, or Tk's writes all finished before Ti validated and Ti read nothing Tk wrote. Because Tk's writes occur before Ti's writes in that case, write-write overlap is harmless.
961. **D** — Let Qk be the version with the largest write timestamp not above TS(T). If R-timestamp(Qk) > TS(T), a younger transaction read Qk and should have seen T's write, so T must be rolled back. Otherwise T overwrites or creates a version.
962. **B** — A transaction may lock an item only if it holds the item's parent, and it may unlock at any time, so the protocol is not two-phase. It is deadlock-free and conflict serialisable, but early unlocks can give non-recoverable or cascading schedules unless extra commit dependencies are enforced.
963. **B** — Basic TO accepts r2(X) after an uncommitted w1(X) when TS(T1) < TS(T2), and nothing stops T2 from committing first. Recoverability needs extra rules, such as doing all writes at the end or making readers wait for the writer to commit.
964. **A** — Reading a leaf needs S on that leaf and intention-shared (IS) on every ancestor, taken top-down. S on the ancestors would lock far more than needed, and IX signals an intention to write.
965. **C** — X on F would let T2 change any record in F, which conflicts with T1 reading somewhere below. X is compatible with no other mode, so T2 waits.

966. **B** — In wound-wait an older requester wounds the holder, but a younger requester simply waits. T3 is younger than T1, so it waits.
967. **C** — Breaking the single cycle needs only one victim, and T2 is the cheapest at 15. Rolling back all three would waste work.
968. **C** — A read is rejected only when TS(T) < W-timestamp(X); 12 is not less than 8, so it succeeds. R-timestamp becomes max(15, 12) = 15. Being younger than an earlier reader is no problem for a read.
969. **C** — A wait-for graph has only transactions as nodes, with an edge from the waiting transaction to the one it waits for. T2 waits for T1, so the edge is T2 -> T1.
970. **A** — The first validation condition is finish(T1) < start(T2). Here 10 < 12, so the transactions did not overlap at all and T2 passes without any set comparison.
971. **A** — SIX gives S on the whole file for the scan and allows X locks on the few records being updated. S alone does not allow the updates, and IX would force an S lock on every record that is read.
972. **A** — In the first sequence every lock comes before the first unlock, so it is 2PL. But X(A) is released before commit, which strict 2PL forbids. The second is rigorous (and therefore strict). The third and fourth lock again after an unlock, so they are not 2PL at all.
973. **B** — r2(Y) set R-timestamp(Y) = 20. A write is rejected if TS(T) < R-timestamp, and 10 < 20, so T1 is rolled back. Thomas' rule only covers TS(T) < W-timestamp; it never rescues a write that a younger transaction has already read past.
974. **D** — w2(A): R-timestamp(A) = 0 but W-timestamp(A) = 30 > 20, so the obsolete write is ignored. w1(B): R-timestamp(B) = 20 > 10 after r2(B), so the write is rejected even with Thomas' rule.
975. **C** — Two shared locks on A are compatible, so T1 and T2 both get S(A). T3's X(A) conflicts with both and waits. B is free, so T2's S(B) is granted. Nobody waits for T3, so there is no deadlock.
976. **B** — Ordering 2PL transactions by their lock points gives an equivalent serial schedule: step 3 (T2), then step 6 (T1), then step 9 (T3).
977. **B** — A reader gets the version with the largest write timestamp that is not greater than its own: 10 <= 15 < 20. Reads never fail under multiversion TO. The R-timestamp of that version becomes max(old, 15).
978. **D** — S(A) is compatible with the held S locks, but an earlier request (T3's X) is still waiting and conflicts with it. Granting T4 first could starve T3, so T4 joins the queue.
979. **A** — The precedence graph has the cycle T1 -> T2 -> T1. Basic 2PL only ever produces conflict-serialisable schedules, so no placement of locks can produce it. If T1 released A early, it could not lock B afterwards.
980. **B** — r3(B) sets R-timestamp(B) = 30. w1(A) sets W-timestamp(A) = 10, and r2(A) is fine because 20 >= 10. Then w2(B) has TS 20 < R-timestamp(B) = 30, so T2 is rolled back. r1(C) would also fail later, but it comes after.
981. **B** — Condition 1 fails because T1 finished (10) after T2 started (3). Condition 2 needs T1 to finish before T2 validates, but 10 > 9, so it fails even though {X} and {Y} are disjoint. T2 is rolled back.
982. **A** — The version to check is Q5, the one with the largest write timestamp not above 12. Its read timestamp 18 > 12, so a younger transaction read Q5 when it should have seen this write. The writer must be rolled back.
983. **B** — T3 waits for T1 and T2 on A; T1 waits for T3 on B. The cycle is T1 -> T3 -> T1. T2 waits for nobody, so it can finish and release A, but T1 still holds S(A), so the deadlock remains.
984. **D** — T2 is older than T3, so it wounds T3 and takes A. Then T1 is older than T2, so it wounds T2 and takes B. Both T3 and T2 are rolled back and restart with their original timestamps.
985. **D** — With multiple versions, a report reads the version that was current when it started, so it needs no shared locks. Writers create new versions and do not wait for readers.
986. **A** — Deadlock needs a circular wait. If every transaction takes the two tables in the same order, one of them always gets the first lock and the other simply waits, so no cycle can form.
987. **A** — The precedence graph tests serialisability of a finished or planned schedule. The wait-for graph records who is currently waiting for whom, and only a cycle there is a deadlock.
988. **D** — When a transaction touches almost the whole file, a single coarse S lock gives the same protection at a fraction of the lock-management cost. This is the reason multiple-granularity locking exists.
989. **D** — r2(A) before w1(A) forces T2 before T1. TO insists that conflicting operations follow timestamp order (T1 before T2), so w1(A), with TS 1 < R-timestamp(A) = 2, is rejected. TO accepts a strict subset of the conflict-serialisable schedules.

990. **C** — There are two separate cycles: T1 -> T2 -> T1 and T3 -> T4 -> T5 -> T3. The edge T2 -> T3 joins them but is not part of a cycle, so one victim from each cycle (for example T1 and T3) is needed.
991. **C** — Each transaction's check was valid in its snapshot, and their write sets did not overlap. The combined result (0 on call) matches no serial order. Running at true SERIALIZABLE (for example serialisable snapshot isolation) would abort one of them.
992. **A** — Basic, strict and rigorous 2PL all allow circular waits, because locks are still taken one at a time during execution. Preventing deadlock needs predeclaration, ordering, or a timestamp-based prevention scheme.
993. **B** — TO is deadlock-free but can starve a long transaction. Silberschatz notes that if a transaction keeps being restarted, conflicting transactions can be blocked for a while to let it complete.
994. **C** — T2 and T3 share S(B). Both then wait for T1's X(A). T1's S(B) is compatible with their shared locks, so it is granted. The wait-for graph is T2 -> T1 and T3 -> T1, with no cycle. T1 finishes and releases A.
995. **D** — In the first schedule T1 can take S(C) at the very start, together with X(A), reach its lock point early and release A before r2(A). T2 never uses C, so nothing blocks it. Locks may be taken before they are needed, which is the key to this question.
996. **D** — Wait-die: T3 (younger) asks T2 and dies, freeing C; T2 (younger) asks T1 and dies; T1 then finds C free. Wound-wait: T3 and T2 are younger requesters, so they wait; T1 (older) asks T3 for C and wounds it. Only T3 rolls back.
997. **D** — w3(A) sets W-timestamp(A) = 3. Then w2(A) has TS 2 < 3 while R-timestamp(A) = 2 is not above 2. Basic TO rolls T2 back; Thomas' rule drops the obsolete write. What remains, r1(B) r2(A) w3(A) w3(B), is equivalent to timestamp order T1, T2, T3.
998. **C** — T1 validates first and passes. T2: T1 finished at 6, after T2 started (2), and before T2 validated (7), but T1 wrote A and T2 read A, so T2 fails. T3: T1 finished at 6, before T3 validated (8), and wrote nothing T3 read, so T3 passes; aborted T2 is not checked.
999. **B** — Reading a record needs IS on F, which SIX allows. Updating a record needs IX on F, which would let T3 write inside the subtree that T1 is reading under its S part, so T3 waits.
1000. **B** — An update log record names the transaction, the data item, the old value (before-image, used for undo) and the new value (after-image, used for redo). Elmasri-Navathe write the same thing as [write_item, T, X, old_value, new_value].
1001. **B** — WAL says the undo information in the log must be on stable storage before the changed page overwrites the disk copy. Otherwise a crash could leave a change on disk with no old value to undo it. The commit record and all redo information must also be forced before commit.
1002. **D** — At a checkpoint the system writes modified buffers to disk and records which transactions are active. After a crash, work committed before the checkpoint is already on disk, so recovery can start from the last checkpoint instead of the start of the log.
1003. **A** — Transaction failures and system crashes leave the disk intact, so the log is enough. A disk failure destroys the stored database itself, so recovery restores the last archival dump and then redoes committed work from the log.
1004. **B** — With deferred update nothing reaches the database before the transaction commits, so an uncommitted transaction never needs undoing. Committed work may still be only partly on disk at a crash, so it must be redone. Hence NO-UNDO/REDO (Elmasri-Navathe).
1005. **B** — Immediate update lets uncommitted changes reach the disk, so they may need undoing. Because commit does not force all pages, committed changes may be missing from the disk and need redoing. This is the UNDO/REDO algorithm.
1006. **B** — Steal lets a page changed by an uncommitted transaction be written to disk to free a buffer frame. If that transaction then aborts or the system crashes, the change on disk must be undone, so the log must keep before-images.
1007. **B** — No-force means a transaction can commit before all its changed pages are on disk. After a crash those committed changes may be lost from the disk, so they must be redone from the after-images in the log.
1008. **A** — An idempotent operation gives the same result whether applied once or many times. Writing a logged after-image (set X to 25) is idempotent, so if recovery itself crashes and restarts, repeating the redo is harmless.
1009. **A** — Once the commit record is safely in the stable log, recovery can always redo the transaction, so it counts as committed even if none of its pages are on disk yet. Elmasri-Navathe call this the commit point.

1010. **B** — Undo must restore the oldest before-image last, so it walks the log backward. Redo must leave the newest committed value in place, so it walks forward and applies after-images in their original order.
1011. **D** — Shadow paging writes changes to new pages and keeps the old (shadow) page table untouched. Commit just switches to the new page table. On a crash the shadow table still describes the old consistent state, so nothing is undone or redone (Elmasri-Navathe). Deferred update still needs redo.
1012. **D** — A simple checkpoint suspends updates until every dirty buffer is flushed, which can take a long time. A fuzzy checkpoint writes the checkpoint record and lets transactions continue while the buffers are flushed in the background. Recovery uses the last checkpoint whose flushing finished.
1013. **C** — ARIES first analyses the log from the last checkpoint to find dirty pages and active transactions. It then redoes (repeats history) from the earliest needed point, and finally undoes the loser transactions backward.
1014. **C** — A CLR records the undo as a redo-only action and points to the next record still to be undone. If recovery crashes, the restart redoes the CLRs and continues from where it stopped, instead of undoing the same update again. Silberschatz's redo-only records <Ti, X, V> do the same job.
1015. **D** — Truly stable storage cannot exist, but replicating each block on several non-volatile devices (for example mirrored disks, or a remote site) that are unlikely to fail together comes close. A single disk or main memory can lose data.
1016. **B** — T1 has a commit record, so its change must survive: redo sets A to 150. T2 never committed, so its change must vanish: undo sets B back to 200.
1017. **D** — Under deferred update an uncommitted transaction never writes to the database, so there is nothing to undo for T2; its log records are simply ignored. T1 committed, so its new value A = 150 is redone.
1018. **C** — T1 committed before the checkpoint, so its changes are already on disk. T2 and T4 committed after it, so they are redone. T3 has no commit record, so it is undone.
1019. **C** — T2 (committed) changed B from 30 to 40. T3 (uncommitted) then changed it from 40 to 45. Undo restores T3's before-image, so B ends as 40. Going back to 30 would wrongly undo committed T2.
1020. **C** — Undo writes the old value (before-image) from the log record, which is 40. The new value 70 is what redo would use, and only for committed transactions.
1021. **D** — The checkpoint gives {T1, T2}. Scanning forward: <T3 start> adds T3, <T1 commit> removes T1, <T4 start> adds T4, and <T4 abort> removes T4 (its rollback already finished). That leaves {T2, T3}.
1022. **A** — The redo phase replays every update record and redo-only record after the checkpoint, whether or not the transaction committed: <T1, C, 0, 30>, <T3, A, 10, 40>, <T4, D, 0, 50> and <T4, D, 0>. That is 4. Losers are fixed later in the undo phase.
1023. **D** — No-steal means uncommitted changes never reach the disk, so no undo. Force means all of a transaction's pages are on disk when it commits, so no redo. The price is poor performance, which is why most systems use steal/no-force and need both.
1024. **D** — Only T2 lacks a commit record. It made three writes: B 15 to 20 (before the checkpoint), C 30 to 35 and B 20 to 45. All three are undone, in reverse order, including the one before the checkpoint, because the checkpoint flushed it to disk.
1025. **C** — A: T3 committed, so its new value 40 stays. B: T2 is undone, and walking back its writes restores 20 and then 15. C: T2's write is undone to 30, the value T1 committed. Restoring B to only 20 forgets T2's first write.
1026. **C** — T1 committed before the checkpoint, so its write is already on disk. T2 committed after it and is redone. T3 and T4 never committed, so under deferred update they wrote nothing to the database and are simply ignored, not undone.
1027. **A** — A loser is a transaction with a start record but no commit or abort record at the crash. T1 and T3 both committed; only T2 did not.
1028. **B** — The undo phase walks backward, so it meets <T3, B, 30, 40> first. For each update it restores the old value and logs a redo-only record with that old value: <T3, B, 30>, then <T3, A, 10>. On reaching <T3 start> it writes <T3 abort>.
1029. **C** — The <T4 abort> record means T4's rollback finished: <T4, D, 0> logged its undo. Repeating history replays both records, which leaves D = 0, and T4 is removed from the undo-list, so it is not undone twice.
1030. **A** — This is exactly what the WAL rule forbids. The uncommitted value 99 is on disk, but the before-image 10 never reached the stable log, so recovery cannot undo it. Forcing the log record before writing the page would have prevented this.

1031. **A** — Only committed transactions' writes after the last checkpoint need redoing, because the checkpoint forced everything earlier to disk. T2 and T4 committed after the checkpoint, and their post-checkpoint writes are A and D.
1032. **D** — Repeating history makes the database match the moment of the crash, losers included. The undo-list ends as {T3}, so the backward pass undoes B (to 40) and then C (to 50), reading past the checkpoint until <T3 start>. Textbooks differ here: Elmasri's basic UNDO/REDO redoes only committed transactions.
1033. **D** — Writing the after-image 110 is idempotent: it gives 110 however many times it runs. Re-applying 'add 10' to a page that already holds 110 gives 120. That is why logical redo needs page LSNs (as in ARIES) to know whether the change is already on the page.
1034. **B** — T1 and T2 committed, and T3 did not. Undoing T3 walks back past the checkpoint: Y goes from 8 to 7 and X from 4 to 3. The value 3 is what committed T2 wrote, so it is the correct final value of X.
1035. **A** — T2 read a dirty value from T1, so when T1 aborts, T2's work rests on data that never committed and must also be undone. Strict 2PL holds T1's exclusive lock on A until it ends, so T2 could not have read A, and rollbacks never cascade.
1036. **A** — Only committed transactions ever write to the database under deferred update. T1's A = 10 was on disk by the checkpoint, and T2's B = 20 is redone. T3's C = 30 and T4's A = 40 never reached the disk, so C stays 0 and A stays 10.
1037. **C** — The undo-list is {T2}, because the checkpoint listed T2 and it never committed. The backward pass must undo every T2 update, including <T2, B, 15, 20> before the checkpoint, and it stops on reaching <T2 start>.
1038. **C** — Force trades recovery work for run-time cost: commit has to wait for random page writes, and a frequently updated page is written again at every commit. That is why most systems choose no-force and rely on redo from the log.
1039. **A** — The last logged values are A = 40 (T3, committed), C = 35 (T2) and B = 45 (T2). T2 is undone, so B becomes 15 and C becomes 30, while A keeps 40. Two items differ.
1040. **A** — A system crash loses the contents of volatile storage (main memory and the buffers) but leaves non-volatile storage such as disk intact. Recovery uses the log on disk to redo committed work and undo uncommitted work.
1041. **C** — Undo removes the effects of transactions that did not commit (atomicity), and redo makes sure committed changes survive a crash (durability). Isolation is the job of concurrency control, and consistency also depends on the application's own logic.
1042. **A** — Undo reverses an update, so it restores the before image (BFIM) that was logged with the update. Redo is the operation that writes the after image (AFIM). An archival dump is used only for media recovery.
1043. **C** — No-steal forbids the buffer manager from writing (stealing the frame of) a page that holds uncommitted changes. Because uncommitted values never reach disk, recovery never needs to undo them. Forcing pages at commit is the separate force policy.
1044. **B** — Under force, a transaction's updated pages are forced to disk as part of commit, so committed work is always on disk and redo is not needed. The cost is many random writes at every commit. Forbidding uncommitted pages on disk is no-steal, not force.
1045. **B** — Updated pages are written to new disk locations and recorded in the current page table, while the shadow page table still points to the old pages. Commit atomically switches the on-disk pointer so the current table becomes the shadow table; until then a crash simply leaves the old table in force.
1046. **C** — In-place updating writes the buffer back to the same disk block, destroying the before image. Logging the before image first (the undo part of the WAL rule) is therefore the only way to undo the change. Shadowing writes the new version elsewhere and keeps the old one.
1047. **D** — Rollback scans Ti's records backward, restoring each old value and logging a redo-only record. Only when it reaches <Ti start> is <Ti abort> written, which tells later recovery that nothing of Ti remains to be undone.
1048. **C** — Main memory, and therefore the buffer pool and any unwritten log buffer, is volatile storage and is lost on a crash. Disks and tapes are non-volatile, so the log and data files survive a crash; only a media failure destroys them.
1049. **D** — A remote backup site receives the primary's log and keeps a synchronised copy of the database. If a fire, flood or other disaster destroys the primary, the backup takes over. The primary still keeps its own log; the backup is for availability and disaster recovery.

1050. **B** — The buffer manager sets the dirty bit when a page in memory is changed. Before reusing the frame it must write a dirty page back to disk (after its log records, by WAL); a clean page can simply be discarded. Pinning is a separate counter.
1051. **D** — Output to stable storage happens in blocks, so batching many small log records into one block write saves I/O. The buffer is volatile, so it must still be forced before a commit is acknowledged and before a data page whose records it holds is written (WAL).
1052. **D** — The dirty page table lists each dirty buffer page with its RecLSN, the LSN of the first log record that dirtied it. The transaction table tracks active transactions and their last LSN, and the master record points to the latest checkpoint.
1053. **B** — Updates may reach disk before commit, so a crash can leave uncommitted values that must be undone. Because every update is on disk by commit time (force), committed work never needs redo. Hence UNDO/NO-REDO.
1054. **D** — Deferred update writes nothing to the database before commit, which is a no-steal policy. All changed pages must therefore be held in memory until commit, which limits transaction size. Committed work is redone from the log, and checkpoints work normally.
1055. **C** — Because a changed page is written to a fresh location, data that was stored contiguously becomes scattered, and the old versions become garbage once the transaction commits. Copying and writing the page table at commit is also costly. Recovery itself is simple, since the shadow table is on disk.
1056. **C** — During redo, a record with LSN L for page P is reapplied only if P's PageLSN is less than L. A PageLSN of L or more proves the update is already on disk, which keeps redo idempotent. The LSN of the first dirtying update is the RecLSN in the dirty page table.
1057. **D** — When an update is undone, its CLR stores UndoNextLSN = that update's PrevLSN. If recovery later meets the CLR, it jumps straight to UndoNextLSN, so work already undone is never undone again. The record just undone is skipped, not revisited.
1058. **B** — Media recovery copies the last dump back to disk, then replays the log records written since the dump so that committed work is restored; transactions that had not committed are rolled back. Undo alone cannot help because the data itself is gone.
1059. **B** — Each commit needs the log forced to stable storage. By waiting a few milliseconds for other transactions to commit, the system writes one log block that holds many commit records, so far fewer disk writes are needed per commit, at the cost of slightly longer commit latency.
1060. **B** — One-safe commit gives the fastest commits, but log records still in transit are lost if the primary is destroyed, so the backup may lack some committed transactions. Two-safe and two-very-safe wait for the backup, which avoids that loss at the cost of latency.
1061. **D** — Redo-only records describe the compensating writes. Repeating history then restores the database exactly as it was at the crash, rollback included. Such records are never undone, and the later <Ti abort> keeps Ti off the undo list.
1062. **D** — A fuzzy checkpoint record is written before its dirty pages are flushed, so a crash may interrupt the flush. The system therefore updates a last-checkpoint pointer (ARIES's master record) only after the flush completes, and recovery starts from the checkpoint that pointer names.
1063. **D** — PrevLSN links a transaction's records into a backward chain. Rollback and the undo phase jump from record to record along that chain instead of scanning the whole log. Skipping up-to-date pages is the job of PageLSN.
1064. **C** — T2 committed, so its write is redone and Y = 15. T1 has no commit record, so its write is undone to the before image and X = 20. Recovery must not keep T1's value 30 or discard T2's committed 15.
1065. **D** — T1 and T3 both have commit records, so they are winners and are redone. T2 has no commit record, so all of its writes (B and A) are undone. Where a transaction's records sit relative to others does not matter, only whether it committed.
1066. **B** — No-steal keeps uncommitted changes off the disk, so undo is never needed. No-force lets a transaction commit before its pages are written, so committed changes may be missing from disk and must be redone.
1067. **C** — Steal allows uncommitted changes to reach disk, so they may need to be undone. Force writes all of a transaction's pages before it commits, so committed work is already on disk and never needs redo.
1068. **B** — Undo writes the old value 7 back to Z and logs a redo-only record holding just the value written, <T4, Z, 7>. <T4 abort> is appended only when the backward scan reaches <T4 start>.

1069. **C** — The redo phase replays <T1, A, 50, 80>, <T2, B, 10, 20> and the redo-only <T1, A, 50>, giving A = 50 and B = 20. T1 has an abort record and T2 a commit record, so the undo-list is empty and nothing more is done.

1070. **A** — The undo-list starts as the checkpoint's {T1, T2}. Scanning forward, T2 commits (removed), T3 and T4 start (added), T4 aborts (removed) and T5 starts (added). That leaves {T1, T3, T5}; T5 wrote nothing but still gets an abort record.

1071. **D** — ARIES redo starts at RedoLSN, the smallest RecLSN in the dirty page table, because that is the oldest update that might not be on disk. Here min(120, 95, 140) = 95, which is earlier than the checkpoint itself.

1072. **C** — Redo reapplies a record only if the page's PageLSN is less than the record's LSN. Since 230 >= 200, the page already contains this update and later ones, so redo skips it. Lowering PageLSN to 200 would also be wrong.

1073. **C** — T1 is a loser, so both of its writes are undone, newest first: A goes back to 60 and then to 50. T2 committed, so B keeps 6. Undoing only T1's last write would leave A = 60.

1074. **A** — T1 committed, so P = 8 is kept. T2 is a loser: its write of P is undone to its own before image, 8, and Q goes back to 1. Restoring P to 4, the oldest value in the log, would wipe out T1's committed update.

1075. **C** — Recovery works from the last checkpoint. T1 committed before it, so the flush already put T1's changes on disk. T3 committed after it and is redone. T2 never committed, so it is undone, not redone.

1076. **A** — T1 committed before the dump, so its changes are already in the dump. T2 committed after the dump, so its updates must be replayed from the log. T3 never committed, so its effects must not appear in the recovered database.

1077. **A** — Both T1 and T2 are losers. Each of their 4 updates gets a redo-only record, and each transaction gets an <abort> record when its <start> is reached: 4 + 2 = 6. Forgetting the abort records gives 4.

1078. **D** — Only T1 committed, so only its writes are redone, in log order: A becomes 15 and then 25. T2's writes never reached the database under deferred update, so B and C keep their old values 1 and 2.

1079. **B** — WAL needs the update's log record on stable storage before page P is written, and all of T's records, ending with the commit record, forced before T commits. Only the first order meets both. Writing P first leaves an uncommitted value on disk with no undo information.

1080. **A** — The undo-list is {T1, T3}. One backward scan meets <T1, Y, 4, 9>, then <T3, W, 7, 8>, then <T3 start> (abort T3), then <T1, X, 1, 2> before the checkpoint, then <T1 start> (abort T1). Records follow log order, not one transaction at a time.

1081. **A** — T2 had already rolled back and T3 committed, so B ends at T3's 22. The losers are T1 (from the checkpoint list) and T4. Undo restores D = 40, C = 30 and, reading back past the checkpoint, A = 10. Stopping at the checkpoint would leave A = 11.

1082. **B** — ToUndo starts with each loser's last LSN: 40 for T1 and 20 for T2. The CLR at 40 is not undone; it sends T1 to UndoNextLSN 10. Taking the largest LSN each time, ARIES undoes 20 (T2) and then 10 (T1). LSN 30 was already compensated, so it is not undone again.

1083. **D** — Analysis adds P2 (RecLSN 60) and P3 (RecLSN 90) to the dirty page table, which already holds P1 with RecLSN 30, so redo starts at min = 30. T1 commits at LSN 80 and leaves the transaction table; T2 has no commit, so it is the only loser.

1084. **A** — LSN 30 is skipped because P3 is not in the dirty page table. LSNs 20 and 40 are skipped because P1's PageLSN is 40, which shows both are on disk. P2's PageLSN is 30, which is below 50 and 60, so those two are reapplied.

1085. **A** — Undo restores the loser's own before image. T2 changed A from 20 to 30, so A goes back to 20, the value T1 committed. Going back to 10 undoes T1 too, which breaks durability for a committed transaction.

1086. **A** — Under no-steal, a page changed by an uncommitted transaction must stay in memory. Once every frame holds such a page, nothing can be evicted, so the transaction cannot continue. This limit on transaction size is why real systems prefer steal and accept the need for undo.

1087. **A** — A transaction is committed only when its commit record reaches stable storage. The log buffer is volatile, so after the crash the log may have no commit record for T. Recovery then rolls T back, which breaks durability for a commit the client already saw.

1088. **B** — Repeating history makes the database match the log at the moment of the crash, whatever reached disk. Undo can then work from one known state with the ordinary rollback procedure, which also supports logical undo and fine-grained locking as in ARIES.

1089. **D** — Each checkpoint limits how far back redo must start, so frequent checkpoints shorten restart. But each one writes dirty pages and a checkpoint record, which costs I/O (and, for simple checkpoints, pauses updates) during normal work.
1090. **C** — Steal lets uncommitted changes reach disk, and undoing them needs before images. A new-value-only log supports redo but not undo, so it works only with no-steal (deferred update). With steal, a crash could leave uncommitted values that nothing can remove.
1091. **A** — T2 still has no abort record, so it is on the undo-list. The backward scan skips the redo-only <T2, C, 9> and undoes <T2, C, 9, 10> again, then B, then writes <T2 abort>. Repeating the undo is harmless because writing a stored before image is idempotent. ARIES CLRs would avoid the repeat.
1092. **C** — Undo must process records newest first. Backward, A goes to 20 (undoing T2) and then to 10 (undoing T1), the value before either transaction. A forward undo would set A = 10 and then 20, leaving a loser's value in the database.
1093. **B** — Repeating history replays every update after the checkpoint, losers included, then undoes the undo-list {T1, T3}. T2 committed after the checkpoint and is a winner. T1 wrote X before the checkpoint, so undo must read back to <T1 start>. Elmasri-Navathe never redo losers.
1094. **A** — After redo (repeating history) the values are A = 11, B = 22, C = 31 and D = 41. Undo rolls back the losers T1 and T4, giving A = 10, C = 30 and D = 40. B belongs to the committed T3 and stays 22, so 3 items change.
1095. **D** — A fuzzy checkpoint only records the dirty page table and the transaction table. A page that became dirty before the checkpoint may still be unwritten, so redo starts at the smallest RecLSN in the dirty page table, which can be well before the checkpoint.
1096. **C** — One forced write now commits 10 transactions, so 200 writes per second support 200 x 10 = 2,000 commits per second. The price is latency: a transaction may wait for the group to fill or for a short timeout before its block is forced.
1097. **B** — Two-safe waits for the commit record to reach both sites when both are up, so a primary disaster loses nothing. If the backup is down, it commits at the primary alone, so processing continues. Two-very-safe would block whenever the backup is down; one-safe can lose committed work.
1098. **A** — With many concurrent transactions, the current page table holds changes from all of them, and switching it would commit every one at once. Elmasri-Navathe therefore note that a multi-user system must add logs and checkpoints to shadow paging to commit or undo transactions separately.
1099. **B** — Media recovery restores the dump and then replays the log written since it. Here the log died with the data, so all work committed after Sunday is lost. This is why the log is kept on a separate device, ideally duplicated on stable storage.
1100. **A** — Elmasri-Navathe define a primary index on the field that both orders the file and is unique (the ordering key). It has one entry per data block, pointing at the block's first (anchor) record. An index on an ordering non-key field is a clustering index; one on a non-ordering field is a secondary index.
1101. **D** — Density is about how many entries the index has. Dense: one entry per search-key value (Silberschatz) or per record (Elmasri). Sparse: entries for only some values, typically one per data block, which works only when the file is sorted on that key.
1102. **B** — B+ tree internal nodes hold only keys and child pointers, to guide the search. All record pointers sit in the leaves, and each leaf points to the next, so a range query finds its first key and then walks along the leaf level.
1103. **A** — A hash function scatters neighbouring keys into unrelated buckets, so there is no order to walk for a range. Hashing is excellent for equality look-ups, typically about one bucket access, but a range query needs an ordered structure such as a B+ tree.
1104. **D** — When the ordering field is not a key, Elmasri-Navathe call the index a clustering index: one entry per distinct value, pointing to the first block holding that value. A primary index needs the ordering field to be a key.
1105. **D** — Both index types rely on the file being physically sorted on the indexed field, and a file can be sorted in only one way. So it has at most one of them, plus any number of secondary indexes. This is why a SQL table can have only one clustered index.
1106. **C** — A sparse index works only when records between two index entries are known to be in order, which needs the file sorted on that field. A secondary index is on a non-ordering field, so records with any value can sit anywhere, and each one needs its own entry.

1107. **A** — A B-tree node holds a data (record) pointer with each key, which uses space. B+ tree internal nodes drop those pointers, so each block fits more keys and children. Higher fan-out means fewer levels and fewer block reads per search.

1108. **B** — If the full bucket's local depth is less than the global depth, several directory entries already point to it, so it can split without changing the directory. Only when local depth equals global depth must the directory double, raising the global depth by 1.

1109. **D** — Linear hashing grows the file gradually: a split pointer moves through the buckets in order, splitting the next one whenever the load rises, and it uses hash functions h_i and h_(i+1). It needs no directory, but it uses overflow chains until a bucket's turn to split comes.

1110. **C** — A clustered index (Silberschatz: clustering or primary index) stores rows in index-key order, so there can be only one. Non-clustered (secondary) indexes are separate structures pointing at the rows, so you can have many.

1111. **A** — A heap file appends new records at the end, so insertion is fast. Because records are in no order, finding one by value means reading on average half the blocks (b/2) for a key, or all b blocks otherwise.

1112. **A** — Static hashing fixes M buckets up front. As records are added, buckets fill up and overflow chains grow, so a look-up may read several blocks. Dynamic schemes (extendible and linear hashing) add buckets as needed.

1113. **A** — Each level of a multilevel index is a primary index on the level below, with one entry per block. Each index block holds bfri entries, so each level is about fo = bfri times smaller, and a search reads about log_fo(r1) index blocks plus the data block.

1114. **C** — A B+ tree is balanced: insertions and deletions keep all leaves at the same depth. Every equality search follows one root-to-leaf path, so it reads exactly 'height' nodes. Nodes need only be at least half full, not completely full.

1115. **B** — Each primary-index entry pairs the key of a block's first record, the anchor, with a pointer to the block. To find key K you look for the last entry with an anchor ≤ K and read that block. That is why the index is sparse.

1116. **C** — bfr = floor(B / R) = floor(1024 / 100) = 10 records per block (unspanned, so the fraction is wasted). Then b = ceil(r / bfr) = ceil(30000 / 10) = 3,000 blocks.

1117. **A** — 50 = 7 × 7 + 1, so 50 mod 7 = 1. Bucket 7 does not exist, since K mod 7 always lies between 0 and 6.

1118. **B** — The file has 3,000 blocks. On a key field the search stops when it finds the record, so on average it reads half the blocks: b / 2 = 1,500. A non-key search, or one that fails, reads all 3,000.

1119. **D** — Binary search over b = 3,000 blocks needs ceil(log2 3000) block accesses. 2^11 = 2,048 < 3,000 ≤ 4,096 = 2^12, so the answer is 12.

1120. **D** — Index entries are 9 + 6 = 15 bytes, so bfri = floor(1024 / 15) = 68. One entry per data block gives 3,000 entries in ceil(3000 / 68) = 45 index blocks. Binary search on them costs ceil(log2 45) = 6, plus 1 to read the data block: 7.

1121. **B** — A secondary index on a key is dense: 30,000 entries. With bfri = 68, it needs ceil(30000 / 68) = 442 blocks. Binary search costs ceil(log2 442) = 9, plus 1 data block: 10 accesses.

1122. **B** — Level 2 needs ceil(442 / 68) = 7 blocks and level 3 needs ceil(7 / 68) = 1 block, the top. That makes 3 levels. A search reads one block per level plus the data block: 4 accesses, against 10 with binary search on the single-level index.

1123. **B** — 6p + 10(p − 1) ≤ 512 gives 16p ≤ 522, so p ≤ 32.6 and p = 32. An internal node holds up to 32 tree pointers and 31 keys.

1124. **D** — Each leaf entry is a key plus a record pointer, 10 + 7 = 17 bytes, and the leaf also needs one 6-byte next-leaf pointer. 17 × p_leaf + 6 ≤ 512 gives p_leaf ≤ 29.8, so p_leaf = 29.

1125. **A** — The full root has 32 children, each of those internal nodes has 32 children, so there are 32 × 32 = 1,024 leaves. Each leaf holds 29 record pointers, giving 1,024 × 29 = 29,696. In practice nodes are about 69% full, so real capacity is lower.

1126. **D** — 12, 33, 40 and 47 all give remainder 5, while 25 and 18 give 4. So bucket 5 holds 4 keys and bucket 4 holds 2, and the other buckets are empty. If a bucket block holds fewer than 4 records, bucket 5 needs an overflow block.

1127. **D** — Non-root internal nodes must have at least ceil(n / 2) = ceil(2.5) = 3 children. Leaves must hold at least ceil((n − 1) / 2) = 2 keys. The root is exempt: it may have just 2 children, or be a single leaf.

1128. **C** — Inserting 40 overflows [10 20 30 40], which splits into [10 20] and [30 40], with 30 copied up to a new root. 60 splits [30 40 50 60], adding 50; 80 splits [50 60 70 80], adding 70. The root [30, 50, 70] has 4 pointers, which n = 4 allows, so it does not split.
1129. **C** — With global depth 2, the first 2 bits of the hash value index the directory: 10. That entry may point to a bucket shared with other entries if the bucket's local depth is less than 2. Some textbooks use low-order bits instead, so check the convention.
1130. **C** — Through a non-clustered index each matching record can cost its own block read, so 6,000 records mean up to about 6,000 data-block reads plus the index. A full scan reads all 3,000 blocks once. For low-selectivity conditions, optimisers choose the scan.
1131. **C** — 0110 overflows the single bucket: global depth becomes 1, giving buckets 0 = {0010, 0110} and 1 = {1010, 1100}. 0001 overflows bucket 0, whose local depth 1 equals the global depth, so the directory doubles to depth 2 and bucket 0 splits into 00 = {0010, 0001} and 01 = {0110}. Entries 10 and 11 share one bucket: 3 buckets in all.
1132. **C** — Every non-root node has at least ceil(60/2) = 30 children. 30^3 = 27,000 < 500,000 ≤ 810,000 = 30^4, so log_30(500,000) ≈ 3.86 rounds up to 4. A look-up therefore reads at most 4 nodes, against about 19 levels for a balanced binary tree.
1133. **B** — With clustering, rows in the date range sit next to each other, so the system reads a few consecutive blocks. With a non-clustered index each matching row may sit in a different block, so the number of block reads approaches the number of rows.
1134. **B** — Fewest leaves: all full with 3 keys, ceil(1000 / 3) = 334. Most leaves: all at the minimum of 2 keys, 1000 / 2 = 500. Any B+ tree with these rules has between 334 and 500 leaves.
1135. **A** — Sparse (one entry per data block): 3,000 entries, ceil(3000 / 68) = 45 blocks. Dense (one entry per record): 30,000 entries, ceil(30000 / 68) = 442 blocks. The sparse index is about 10 times smaller because each data block holds 10 records.
1136. **B** — Every key leaves remainder 0 mod 8, so all six land in bucket 0. Mod 7 gives 2, 3, 5, 1, 2 and 4, so the fullest bucket holds 2. A prime M avoids clustering when keys share a common factor with the table size, which is why textbooks recommend it.
1137. **A** — Sequential inserts fill each leaf to 3 and split it into 2 + 2, giving leaves [1 2], [3 4], [5 6], [7 8], [9 10]. The root collects 3, 5, 7 and then 9; at 9 it has 5 pointers and splits into [3 5] and [9], with 7 moving up. That is 5 leaves + 2 internal nodes + 1 root = 8 nodes in 3 levels.
1138. **D** — Because the file is ordered on dept_no, all 50 employees of a department lie in consecutive blocks. A clustering index has one entry per distinct dept_no pointing to the first such block, so the query reads a small index plus a few adjacent data blocks.
1139. **C** — 6p + 17(p − 1) ≤ 512 gives 23p ≤ 529, so p = 23. The B+ tree internal node leaves out record pointers: 6p + 10(p − 1) ≤ 512 gives 32. Higher fan-out is the main reason DBMSs use B+ trees.
1140. **D** — With spanned blocking, a record that does not fit in the space left in a block continues in the next block via a pointer, so no space is wasted. Unspanned blocking leaves the leftover space unused. Spanning is required when a record is larger than a block.
1141. **A** — To keep the order, the new record must go between its neighbours, so on average half of the file's records move. Systems avoid this with free space in each block or an unsorted overflow file that is merged in later.
1142. **C** — Hashing computes the bucket address straight from the key, so an equality look-up usually costs about one block access. Binary search on a sorted file needs about log2 b accesses, and a heap file needs a linear scan.
1143. **C** — A bitmap index keeps one bit vector per distinct value, so it is compact when there are few values and fast to combine with AND/OR. With unique values it would need one bitmap per row, and heavy concurrent updates make bitmaps costly to maintain.
1144. **C** — A composite key is compared column by column, like words in a dictionary: first dept, then salary to break ties. That is why the index helps queries on dept, or dept and salary, but not on salary alone.
1145. **B** — If all the columns in the query's SELECT list and WHERE clause are in the index, the DBMS can answer from the index alone (an index-only scan) and skip the extra block access per row. Being dense is a different property.

1146. **A** — ISAM builds a static multilevel index over a sorted file. Inserted records that do not fit go into overflow areas, and the index is not rebalanced, so long chains build up until the file is reorganised. A B+ tree splits and merges nodes to stay balanced.

1147. **C** — A collision happens when the hash function maps a new key to an address that is already occupied by a record with a different key. The file then needs a resolution method such as open addressing, chaining or a second hash function.

1148. **D** — With chaining, each bucket has a pointer to a list of overflow records that hashed to it. Searching follows the chain. Placing the record in the next free slot is open addressing, and applying another function is multiple (double) hashing.

1149. **D** — Records become variable-length when they have variable-length fields, repeating fields, optional fields, or mixed record types. The file then needs separators or length prefixes. Sorting or hashing does not change record length.

1150. **D** — Indexes speed up reads but every change to the table must also change each affected index, which costs extra block writes and space. That is why heavily written tables should carry only the indexes that queries really use.

1151. **A** — The search key is whatever attribute(s) the index is built on. It need not be the primary key and need not be unique; a table can have several indexes with different search keys.

1152. **A** — A block pointer gives only the disk address of a block. A record pointer adds the offset or slot of the record within that block, so it identifies one record. That is why record pointers are usually a little larger.

1153. **B** — Because the file is not ordered on the field, matching records are scattered. The common option adds a level of indirection: each index entry points to a block of record pointers. Following physically adjacent records only works for a clustering index.

1154. **C** — Underflow is fixed locally. If a sibling has more than the minimum, entries are redistributed and the parent's separator is updated. If not, the two nodes merge and a separator is removed from the parent, which can underflow in turn. The tree shrinks only when the root is left with one child.

1155. **B** — Sorting the entries first lets the leaves be filled in order, then the upper levels are built from the leaf keys. Each node is written once, nodes can be packed full, and no top-down search or split happens per entry.

1156. **A** — Linear probing tries home address + 1, + 2, and so on (wrapping round) until a free position appears. Searches follow the same sequence. Linking overflow records is chaining, and a second function is double hashing.

1157. **A** — The bucket is identified by its first d' bits, and the directory uses d bits, so every combination of the remaining d − d' bits points to the same bucket: 2^(d − d') entries. Only when d' = d is there exactly one entry.

1158. **B** — Linear hashing splits buckets in a fixed round-robin order, so the overflow triggers a split of bucket n = 2 and n then moves on. Bucket 5 keeps its overflow record for now and is split when n reaches it. There is no directory to double.

1159. **D** — When each distinct clustering value has its own block or block cluster, a new record with that value goes into that block without shifting other records. The index stays sparse with one entry per distinct value.

1160. **C** — Appending a unique attribute such as the record ID makes the composite key unique. Deletion can then find the exact entry quickly, and look-ups on the original attribute become range searches over that prefix.

1161. **D** — Each bitmap has one bit per row. A bitwise AND of the two keeps a 1 only where both conditions hold, which is very fast. OR would answer the disjunction instead.

1162. **B** — Hashing performs well only if keys are spread evenly. A poor hash function, or many duplicate or patterned keys, piles records into a few buckets and creates long overflow chains. A good function and a suitable number of buckets reduce skew.

1163. **A** — A sparse index entry says, for example, that keys from 100 up to the next entry are in block 7. That is only true if the file is ordered on the field. Without that order, a record could be in any block, so every record needs its own entry.

1164. **C** — bfr = floor(4,096 / 300) = 13, since a 14th record would not fit. b = ceil(10,000 / 13) = 770. The value 733 is what spanned storage would need, which unspanned blocks cannot reach.

1165. **D** — Each block holds floor(4,096 / 300) = 13 records, using 3,900 bytes. The remaining 4,096 − 3,900 = 196 bytes cannot hold another record, so they are wasted under unspanned organisation.

1166. **B** — 123 = 11 × 11 + 2, so 123 mod 11 = 2 and the record goes to bucket 2. Bucket 11 does not exist, because the addresses run from 0 to 10.

1167. **C** — A bucket with local depth d' is shared by 2^(d − d') directory entries. Here 2^(3 − 1) = 4 entries point to it, since only the first bit matters for that bucket.

1168. **D** — There is one bitmap per value, each with 1,000,000 bits = 125,000 bytes. Four of them take 4 × 125,000 = 500,000 bytes. Counting a single bitmap gives 125,000, and counting bits as bytes gives 4,000,000.

1169. **B** — With spanning, no space is wasted at the end of blocks, so b = ceil(10,000 × 300 / 4,096) = ceil(732.4) = 733. Unspanned storage needs 770 blocks, because 196 bytes per block are lost.

1170. **B** — bfr = 10, so b = 5,000 blocks. Index entries are 12 + 8 = 20 bytes, so bfri = 102 and the sparse index has ceil(5,000 / 102) = 50 blocks. Binary search on it takes ceil(log2 50) = 6, plus 1 data block = 7. Binary search on the file itself would need 13.

1171. **D** — A clustering index has one entry per distinct value, so 500 entries. bfri = floor(1,024 / 10) = 102, giving ceil(500 / 102) = 5 blocks. Binary search over 5 blocks takes ceil(log2 5) = 3 accesses, after which the first data block holding the value is read.

1172. **B** — The first level has ceil(50,000 / 102) = 491 blocks, the second ceil(491 / 102) = 5, and the third 1. That is t = 3 levels, and a search reads one block per level plus the data block: 3 + 1 = 4. Binary search on the 491 blocks alone would need 9 + 1 = 10.

1173. **D** — Internal: 20p − 12 ≤ 1,024 gives p ≤ 51.8, so p = 51. Leaf: 22 × p_leaf + 8 ≤ 1,024 gives p_leaf ≤ 46.2, so 46. Using the 8-byte block pointer instead of the 10-byte record pointer in leaves would give 50.

1174. **A** — 15, 5, 25 fill the first leaf. 10 overflows it: [5 10] stays and [15 25] splits off, with 15 copied to a new root. 20 joins [15 20 25]; 30 overflows it into [15 20] and [25 30], and 25 goes up. 35 then joins the last leaf.

1175. **A** — First compute h0(20) = 0. Since 0 < n, bucket 0 has already been split, so use h1(20) = 20 mod 8 = 4. Stopping at h0 would wrongly give bucket 0.

1176. **C** — 22 goes to slot 2; 32 and 42 collide and probe to 3 and 4; 15 goes to its home slot 5. 25 also hashes to 5, which is full, so it probes to slot 6.

1177. **D** — Full leaves need ceil(1,000,000 / 99) = 10,102 leaves. The level above needs ceil(10,102 / 100) = 102 nodes, then ceil(102 / 100) = 2, then a root. That is 4 levels. Three full levels reach only 100 × 100 × 99 = 990,000 keys.

1178. **D** — The entries are sorted by dept first, so a search must fix a leading prefix of the key. Conditions on dept, or dept and salary, can seek into the index; a condition on salary alone matches entries spread across every dept. SQLite's query plan confirms this.

1179. **B** — 8, 3, 12, 6 fill a leaf as [3 6 8 12]. Inserting 1 gives five keys, so [1 3 6] stays and [8 12] splits off, with 8 copied up as the root. 15 and 10 join the right leaf, and 4 joins the left: [1 3 4 6] and [8 10 12 15].

1180. **B** — The root has 51 children, each of those has 51 leaf children, so there are 51 × 51 = 2,601 leaves. Each full leaf holds 46 record pointers: 2,601 × 46 = 119,646. 2,346 is the capacity of a two-level tree.

1181. **A** — The root points to 35 internal nodes, each pointing to 35 leaves: 35 × 35 = 1,225 leaves. Each leaf has 32 record pointers, so 1,225 × 32 = 39,200. Using 35 for the leaves too gives 42,875.

1182. **D** — 1110 doubles the directory to depth 1. 1001 then fills the 1-bucket again, so the directory doubles to depth 2 and that bucket splits into 10 [1001] and 11 [1100 1110]. 1111 overflows 11, doubling to depth 3. Bucket 10 keeps local depth 2, so 2^(3 − 2) = 2 entries (100 and 101) point to it.

1183. **B** — 12 overflows bucket 0, which splits into 0 and 2 (all of 4, 8, 12 stay in 0); n = 1. 9 overflows bucket 1, which splits into 1 [5 9] and 3 [7]; n wraps to 0 and the level rises. 15 mod 4 = 3, so it goes to bucket 3. There are 4 buckets.

1184. **A** — With increasing keys, every split happens in the rightmost leaf and leaves 2 keys behind, which are never added to again. So every leaf ends with 2 keys: 20 / 2 = 10 leaves. Bulk loading with full leaves would need only ceil(20 / 3) = 7.

1185. **C** — Every insert must update all 6 indexes, so each row costs several extra random writes. A daily date query needs only one index; dropping the rest removes most of the overhead.

1186. **A** — An equality on gender matches about half the rows, so following a B+ tree to each row costs more than scanning. Bitmaps on low-cardinality columns are tiny, and ANDing several of them narrows the result before any row is read.

1187. **A** — Every column Query 1 needs (dept, salary) is in the index, so it is answered by an index-only scan. Query 2 also needs name, which is not in the index, so each matching entry leads to an extra table access. SQLite reports a covering index for Query 1 only.

1188. **C** — A heap file gives the cheapest inserts, and a job that reads every row gains nothing from order or hashing. Sorting or hashing add insertion cost, and extra indexes add more, for no benefit to a full scan.

1189. **A** — A clustered index stores rows in key order. Clustering on customer_id puts a customer's 200 orders in a few consecutive blocks, while a non-clustered index might need up to 200 block reads. The rare date queries can use a non-clustered index.

1190. **D** — The last two digits of YYYYMMDD are DD, so addresses are limited to 01 to 31 and 69 buckets stay empty. Days 1 to 28 occur in all 12 months, giving 12 records each. Using a prime M, such as 97, spreads the keys across many more buckets.

1191. **B** — Records are separated by more bits of their hash values. Identical hash values share every bit, so each split puts all 6 back in one bucket and the directory doubles again and again. Real systems cap this with overflow buckets.

1192. **A** — Going down the tree reads 3 nodes, one more leaf finishes the entries, and each of the 300 rows needs its own data block: about 304 reads, against 5,000 for a scan. The index wins here because few rows match; with thousands of matches the scan would win.

1193. **D** — Leaf [20] now has 1 key, below the minimum of 2. Together with its sibling it has 4 keys, too many for one leaf, so the leaves redistribute: 15 moves across, giving [5 10] and [15 20], and the parent's separator becomes 15.

1194. **B** — Leaf [20] underflows, and its sibling [5 10] has no key to spare. Their 3 keys fit in one leaf, so they merge into [5 10 20] and separator 20 is removed from the root. A root with a single child is deleted, so the tree shrinks from 2 levels to 1.

1195. **C** — ISAM's index levels never change, so the new records pile into overflow chains that each search must walk. The B+ tree splits nodes on insertion and stays balanced, so look-up cost grows only with the logarithm of the size. Splits happen on inserts, not searches.

1196. **C** — B+ tree: ceil(500,000 / 29) = 17,242 leaves, then 539, 17 and 1 node, so 4 levels. A full B-tree of order 23 holds 23^L − 1 keys: 279,840 with 4 levels, which is too few, so it needs 5. The B+ tree's larger fan-out saves a level.

1197. **C** — A split rehashes the old bucket's records with the next function. Here 4, 8 and 12 are all multiples of 4, so K mod 4 = 0 keeps them in bucket 0 and bucket 2 stays empty. Linear hashing accepts such temporary overflow; later splits may fix it.

1198. **C** — A dense index has an entry for every emp_id, so finding or not finding the value in the index settles the question. A sparse index has only block anchors, so it can locate the block but must read it to see whether the value is really there.

1199. **B** — Email look-ups are pure equality, where a hash index needs about one block access. The age query is a range with sorting, which needs ordered keys, so a B+ tree is required there; a hash on age would be useless for BETWEEN.

1200. **D** — The parser checks syntax and translates the SQL into an internal form such as relational algebra. The optimiser then picks a low-cost evaluation plan, and the evaluation engine runs that plan and returns the result. Compilation and linking belong to program builds, not query processing.

1201. **D** — A plan says not only which operations run (selection, join, projection) but how: for example 'use the B+-tree index on dept for the selection, then a hash join'. Two plans for the same query give the same answer but can differ hugely in cost.

1202. **B** — Disk I/O dominates query cost for disk-resident data, so Silberschatz counts block transfers (each taking time tT) and seeks (each taking tS). CPU cost is real but usually left out for simplicity. The size of the answer or of the SQL text does not measure the work done.

1203. **B** — Pushing selections down the tree shrinks intermediate results early, so later joins process fewer tuples. Heuristic optimisers also push projections down and avoid Cartesian products. Joining the largest relations first usually makes intermediates bigger, not smaller.

1204. **B** — With pipelining, the output of, say, a selection feeds directly into the next join or projection as it is produced. This saves the cost of writing and re-reading temporary relations, which materialised evaluation pays for every intermediate result.

1205. **C** — Nested-loop join simply tests the condition on every pair of tuples, so it works for any theta condition. Hash join (including the Grace variant) and merge join rely on matching equal values of the join attributes, so they handle equi-joins and natural joins only.
1206. **B** — V(A, r) is the number of distinct values that appear for A in r. The other statistics are n_r (number of tuples), b_r (number of blocks) and l_r (tuple size in bytes). V(A, r) drives the estimate for an equality selection, n_r / V(A, r).
1207. **B** — The optimiser generates equivalent plans and picks the one with the lowest estimated cost, using catalogue statistics. Estimates can be wrong, so the chosen plan is not provably the fastest. It must never change the result, so returning fewer rows is not allowed.
1208. **D** — Materialisation evaluates the operator tree bottom-up and writes each intermediate result to a temporary relation, which the parent then reads. Its cost is the cost of the operations plus writing and re-reading those temporaries, which pipelining avoids.
1209. **A** — A sort cannot know which tuple comes first until it has seen every tuple, so it blocks the pipeline. Selection, projection without DISTINCT and UNION ALL can each pass tuples onwards one at a time as they arrive.
1210. **A** — This is the cascade of selections: a selection on θ1 AND θ2 equals applying θ2 and then θ1. It lets the optimiser push each part of a condition to where it is cheapest, such as next to the relation whose attributes it uses. The other rules are projection cascade, join commutativity and selection over union.
1211. **C** — In demand-driven (pull) pipelining, the parent pulls tuples from its child by calling next(); such an operator is called an iterator. In producer-driven (push) pipelining, each operator eagerly fills a buffer for its parent. Materialisation writes whole results to disk instead.
1212. **D** — System R limited its search to left-deep trees, where each join's right (inner) input is a stored relation. This cuts the search space from (2(n−1))!/(n−1)! orders to n! and suits pipelining and indexed nested-loop joins. More general optimisers also consider bushy trees.
1213. **D** — A secondary index orders pointers, not the file, so consecutive matching records are scattered across the file. If many records match, one random I/O per record can exceed reading every block once sequentially. B+-tree secondary indices do support ranges; the problem is the cost per record.
1214. **A** — The build input is loaded partition by partition into an in-memory hash table. Using the smaller relation keeps each partition small enough to fit in memory and lowers the risk of recursive partitioning. The probe input is then streamed against the table.
1215. **B** — A sort order is 'interesting' if a later operation can use it, for example a later merge join, a GROUP BY or an ORDER BY on the same attribute. Keeping it can make the whole plan cheaper even though the sub-plan on its own is not the cheapest.
1216. **B** — For an equality selection on a non-key attribute, the estimate is n_r / V(A, r) = 10,000 / 50 = 200 tuples. The answer 50 confuses the number of distinct values with the result size.
1217. **D** — When the smaller relation fits in memory, each relation is read once, so the cost is b_r + b_s = 300 + 40 = 340 block transfers (the best case). 12,000 is the product b_r × b_s from the block nested-loop worst case.
1218. **A** — Worst case: n_r × b_s + b_r = 6,000 × 40 + 300 = 240,300, because each outer tuple causes a full scan of course. 12,300 is block nested-loop (b_r × b_s + b_r), 120,040 is the cost with course as the outer relation, and 340 is the best case.
1219. **C** — With M − 2 = 20 blocks for the outer, r is read in ⌈120 / 20⌉ = 6 chunks and s is scanned once per chunk: 6 × 500 + 120 = 3,120 transfers. 60,120 is the cost with only one outer block, and 620 would need all of r in memory at once.
1220. **A** — When both inputs are sorted on the join attribute, merge join reads each relation once: b_r + b_s = 300 + 40 = 340 transfers. 1,020 is the hash join estimate 3(b_r + b_s). Sorting would add cost only if the inputs were not already sorted.
1221. **A** — Partitioning reads and writes both relations, 2(b_r + b_s), and the build-and-probe phase reads them again, b_r + b_s. That gives 3(b_r + b_s) = 3 × 340 = 1,020. The full formula adds 4n_h for partly filled blocks, which this estimate ignores.
1222. **C** — With a uniform distribution, the fraction of tuples with marks ≥ 80 is (max − v) / (max − min) = (100 − 80) / 100 = 0.2, so 0.2 × 5,000 = 1,000 tuples. 4,000 is the estimate for marks < 80, the complement.
1223. **A** — For a conjunction, multiply the selectivities: n × (s1/n) × (s2/n) = 10,000 × 0.05 × 0.2 = 100. 2,400 is the estimate for the disjunction (OR), and 2,500 simply adds the two sizes.

1224. **B** — For a disjunction, n × (1 − (1 − s1/n)(1 − s2/n)) = 10,000 × (1 − 0.95 × 0.8) = 10,000 × 0.24 = 2,400. Simply adding 500 + 2,000 double-counts the 100 tuples estimated to satisfy both conditions.
1225. **C** — Because ID is a key of student, each takes tuple matches at most one student tuple, and the non-null foreign key means it matches exactly one. So the join has exactly as many tuples as takes: 10,000. 50,000,000 is the Cartesian product size.
1226. **C** — The estimate is the smaller of n_r·n_s / V(B, s) and n_r·n_s / V(B, r): min(2,000,000 / 100, 2,000,000 / 50) = min(20,000, 40,000) = 20,000. The lower value is used because it is likely to be more accurate.
1227. **B** — The equality condition on dept matches the index idx_dept, so SQLite finds rows through the index (SEARCH) instead of reading every row (SCAN). Output checked in SQLite 3.50. EXPLAIN QUERY PLAN is SQLite-specific; other DBMSs use EXPLAIN with their own output formats.
1228. **D** — Silberschatz gives (2(n − 1))! / (n − 1)! join orders for n relations: 6! / 3! = 720 / 6 = 120 for n = 4. 24 = 4! is the number of left-deep orders, and 12 is the count for 3 relations.
1229. **C** — Run creation makes ⌈1,000 / 11⌉ = 91 runs. Each merge pass merges M − 1 = 10 runs, so ⌈log10 91⌉ = 2 passes are needed. Cost = 1,000 × (2 × 2 + 1) = 5,000 transfers. 3,000 assumes one merge pass and 7,000 assumes three.
1230. **C** — Projection does not distribute over set difference. Take R = {(1, 0)} and S = {(1, 5)}: R − S = {(1, 0)}, so πA(R − S) = {1}, but πA(R) − πA(S) = {1} − {1} = ∅. Selection distributes over union and difference, and projection distributes over union.
1231. **C** — The selection keeps 5,000 / 25 = 200 students, 1/25 of the total. Each takes tuple matches exactly one student, so about 10,000 / 25 = 400 takes tuples survive. The join estimate n_r·n_s / max(V) = 200 × 10,000 / 5,000 gives the same 400.
1232. **A** — Materialisation writes the 50-block temporary relation and the projection then reads it back: 50 + 50 = 100 transfers. Pipelining passes tuples straight to the projection, so both the write and the re-read disappear.
1233. **C** — s outer: ⌈40 / 10⌉ × 300 + 40 = 4 × 300 + 40 = 1,240. r outer: ⌈300 / 10⌉ × 40 + 300 = 1,200 + 300 = 1,500. Using the smaller relation as the outer reduces the number of inner scans, so s as outer wins.
1234. **A** — M = 100: n_h = 64 and 100 > 65, so every partition fits. M = 50: n_h = 128, more partitions than one pass can create with 50 buffers. As a rule of thumb, M must exceed √b_s = 80. The smaller sizes fail even more clearly.
1235. **B** — Scan: one seek plus 1,000 transfers = 4 + 1,000 × 0.1 = 104 ms. Secondary index: (h + n)(tT + tS) = (3 + 400) × 4.1 ≈ 1,652 ms, since each matching record may need its own seek. With this many matches, the scan is cheaper.
1236. **D** — Because both inputs are already sorted, merge join reads each once: 800 + 200 = 1,000. Hash join costs 3(800 + 200) = 3,000. Block nested-loop with s outer costs ⌈200/50⌉ × 800 + 200 = 3,400, and with r outer ⌈800/50⌉ × 200 + 800 = 4,000.
1237. **A** — σθ(E1 ⋈ E2) = σθ(E1) ⋈ E2 holds only when θ uses attributes of E1 alone. A < C compares an R attribute with an S attribute, so it must be applied at or after the join. Random tests confirm the rewrite changes results, while pushing σA=1 to R is always safe.
1238. **D** — A disjunctive selection can use indices only if every disjunct has an access path; the record pointers from each are then unioned. With no index on B, any tuple might satisfy B = 7, so the system must scan the relation, testing both conditions in one pass.
1239. **D** — r1 ⋈ r2 ≈ 1,000 × 2,000 / max(100, 500) = 4,000. r2 ⋈ r3 ≈ 2,000 × 500 / max(50, 500) = 2,000. r1 and r3 share no attribute, so r1 ⋈ r3 is a Cartesian product of 500,000 tuples. Joining r2 and r3 first gives the smallest intermediate.
1240. **C** — Parsing and translation is the first step: the parser checks syntax, uses the catalogue to verify relation and attribute names, and produces an internal representation (a parse tree turned into relational algebra). Costing plans is the optimiser's job, and reading blocks is the evaluation engine's job.
1241. **C** — Cost estimation relies on catalogue statistics: the number of tuples n_r, the number of blocks b_r, the blocking factor f_r and the number of distinct values V(A, r). Running the query first would defeat the purpose of estimating cost before choosing a plan.
1242. **B** — f_r is the number of tuples of r that fit into one block. If the tuples are stored physically together, b_r = ⌈n_r / f_r⌉. The number of blocks is b_r, and the fraction satisfying a condition is the selectivity.
1243. **A** — Linear search reads every block and tests each record, so it works for any condition and any file organisation. Binary search needs the file sorted on the attribute, and index lookups need a suitable index.

1244. **A** — Binary search jumps to the middle block and halves the range each time, which only works if the records are stored sorted on the search attribute. A hash index gives a hash lookup, not a binary search, and the file does not need to fit in memory.

1245. **A** — Duplicate elimination brings equal tuples together, either by sorting (duplicates end up next to each other) or by hashing (duplicates land in the same partition). Plain pipelining cannot detect duplicates because it never remembers earlier tuples.

1246. **C** — Associativity says (E1 ⋈ E2) ⋈ E3 = E1 ⋈ (E2 ⋈ E3), so the grouping of joins can change. Commutativity only swaps the two inputs of a single join (E1 ⋈ E2 = E2 ⋈ E1). Together they give the optimiser its many join orders.

1247. **C** — A theta join is defined as a Cartesian product followed by a selection, so σθ(E1 × E2) can be rewritten as E1 ⋈θ E2. The join can then use an efficient algorithm instead of generating the whole product first.

1248. **B** — In producer-driven pipelining the lower operators push tuples into a buffer for their parent and pause only when the buffer is full. Demand-driven (lazy) pipelining is the opposite: the parent pulls tuples with next(). Writing everything to disk first is materialisation.

1249. **B** — Selectivity is the probability that a tuple satisfies the condition. For σA=v(r) under a uniform distribution it is 1 / V(A, r), so the estimated result size is n_r / V(A, r).

1250. **B** — Index nested-loop join scans the outer relation and, for each outer tuple, looks up matching inner tuples through an index on the inner relation's join attribute. That index lookup replaces a full scan of the inner relation. Sorted inputs are what merge join needs.

1251. **D** — Run creation reads M blocks into memory, sorts them in memory and writes them out as one run, so there are ⌈b_r / M⌉ initial runs. The M − 1 figure belongs to the merge stage, where one block is kept as the output buffer.

1252. **C** — Only the outermost projection matters in a cascade of projections, because each inner one keeps at least the attributes of the next. The other rules are either invalid in general or wrongly stated: a cascade of selections combines with AND, not OR.

1253. **D** — Plain EXPLAIN shows the chosen plan with estimated costs and rows without running the query. EXPLAIN ANALYZE executes it and adds the actual rows and timings, which shows where the estimates were wrong. Statistics are refreshed by the separate ANALYZE command.

1254. **D** — Formulas such as n_r / V(A, r) assume every value is equally common. A histogram records how many tuples fall in each range of values, so selections on popular or rare ranges are estimated much more accurately and the optimiser picks better plans.

1255. **B** — A correlated subquery refers to the outer tuple, so straightforward evaluation repeats it for every outer tuple. Rewriting it as a join (or semijoin) lets the optimiser choose efficient join algorithms and join orders over the whole sets.

1256. **D** — If r-tuple t and s-tuple u have the same join value, h gives them the same partition number i. The algorithm can then join only r_i with s_i. A different hash function for each relation would scatter matching tuples into unrelated partitions.

1257. **B** — Semantic optimisation uses knowledge of constraints. For example, if a CHECK constraint guarantees salary < 500,000, then σsalary > 600,000 must be empty and needs no table access. A foreign key can also make a join to the parent table unnecessary.

1258. **A** — Response time depends on what is already in the buffer and on how much disk activity overlaps in parallel, so it is hard to estimate before running the query. Silberschatz notes that optimisers therefore usually minimise resource consumption, such as block transfers and seeks, even though that does not always give the lowest response time.

1259. **A** — An average cannot be combined from partial averages unless their sizes are known. Keeping a sum and a count per group lets partial results be merged (sum1 + sum2, count1 + count2), and AVG = sum / count at the end. SUM, COUNT, MIN and MAX can be combined directly.

1260. **B** — When memory is larger than needed for partitioning, hybrid hash join uses the spare space to hold one build partition in memory. Probe tuples that hash to that partition are joined straight away, saving the write and re-read of both matching partitions.

1261. **A** — Because the file is ordered on A, all tuples with A ≥ v sit together after the first match. The index locates that first tuple (hi seeks and transfers), and the remaining b blocks are read one after another, costing roughly hi × (tT + tS) + b × tT.

1262. **B** — Cost estimates are only as good as the statistics behind them. ANALYZE recomputes values such as tuple counts, distinct values and histograms, so plans chosen after large inserts or deletes reflect the real data.

1263. **C** — Each run being merged needs one input buffer block and the merged output needs one more, so at most M − 1 runs are merged per pass. This is why the number of merge passes is ⌈log_{M−1}(b_r / M)⌉.

1264. **A** — One initial seek plus 500 sequential transfers: 4 + 500 × 0.1 = 4 + 50 = 54 ms. Treating every block as needing a seek would give thousands of milliseconds, but a contiguous file needs only the first seek.

1265. **B** — On a key attribute the scan can stop at the first match, so on average it reads half the file: 800 / 2 = 400 blocks. Without the key property it would have to read all 800 blocks. Ten is ⌈log2 800⌉, the binary-search figure, which needs a sorted file.

1266. **C** — b_r = ⌈n_r / f_r⌉ = ⌈12,000 / 40⌉ = 300 blocks. Multiplying instead of dividing gives 480,000, which confuses tuples per block with blocks per tuple.

1267. **A** — With no common attributes, a natural join is a Cartesian product, so every r-tuple pairs with every s-tuple: 300 × 40 = 12,000. The result is not empty and is not the sum of the sizes.

1268. **D** — σstatus = 'closed' is estimated at 8,000 / 4 = 2,000 tuples, so the negation keeps the rest: 8,000 − 2,000 = 6,000. 2,000 is the size of the equality, not of its negation.

1269. **C** — ⌈log2 1,000⌉ = 10, because 2^10 = 1,024. Each probe needs a seek and a transfer: 10 × (0.1 + 4) = 41 ms. 104 ms is a full linear scan (4 + 1,000 × 0.1), and 1.0 ms forgets the seeks.

1270. **D** — Three index levels plus one data block are each fetched with a seek and a transfer: (3 + 1) × 4.1 = 16.4 ms. 12.3 ms forgets the data block, and 16.0 ms counts only the seeks.

1271. **C** — A secondary index on a non-key may need one seek and one transfer per matching record: (3 + 20) × 4.1 = 94.3 ms. 14.3 ms treats the 20 records as one sequential run, which is only true for a clustering index. 82.0 ms leaves out the index traversal.

1272. **B** — Worst case: b_r × b_s + b_r = 200 × 50 + 200 = 10,200. Each outer block causes one full scan of s, plus each outer block is read once. 10,050 is the cost with s as the outer relation, and 250 is the case where one relation fits in memory.

1273. **D** — Every outer block is read once (100) and every one of the 2,000 outer tuples triggers an index lookup costing 4: 100 + 2,000 × 4 = 8,100. Multiplying the lookup cost by the number of blocks (100 × 4) instead of tuples is a common slip.

1274. **A** — Run creation gives ⌈5,000 / 21⌉ = 239 runs. Each pass merges up to M − 1 = 20 runs: 239 → 12 → 1, so 2 passes, which matches ⌈log20(238.1)⌉ = 2. 8 is a two-way merge (⌈log2 239⌉).

1275. **A** — 3 × (400 + 100) + 4 × 5 = 1,500 + 20 = 1,520. The 4n_h term covers partly filled partition blocks, which are written and read for both relations. 1,500 leaves that term out.

1276. **A** — The query filters on dept, the leftmost column of i, so SQLite searches the index. It needs only name, which is also in the index, so it never reads the table rows: a covering index. Without name in the index the plan would say USING INDEX instead.

1277. **B** — With no index on name, SQLite must read every row and then sort them, which it does with a temporary B-tree. Rows are stored in rowid order, not name order, so the extra sorting step is needed.

1278. **B** — 4,000 × (30 − 0) / (100 − 0) = 4,000 × 0.3 = 1,200. 2,800 is the size of the complement, marks > 30, and 40 treats the condition as an equality on one of 100 values.

1279. **C** — Multiply the selectivities: 20,000 × (2,000/20,000) × (5,000/20,000) × (400/20,000) = 20,000 × 0.1 × 0.25 × 0.02 = 10. 500 stops after two conditions, and 7,400 adds the sizes, which fits neither AND nor OR.

1280. **B** — (2 × 4)! / 4! = 40,320 / 24 = 1,680. 120 is 5!, the number of left-deep orders. 30,240 wrongly uses n instead of n − 1 (10!/5!). 14 is the number of tree shapes without assigning relations to the leaves.

1281. **A** — Chunks = ⌈100/10⌉ = 10. Transfers = 10 × 1,000 + 100 = 10,100, giving 1,010 ms. Seeks = 2 × 10 = 20, giving 80 ms. Total = 1,090 ms. Leaving out seeks gives 1,010 ms, and using M instead of M − 2 gives 9 chunks and 982 ms.

1282. **B** — Ages 35–39 are 5 of the 10 values in the 30–39 bucket: 600 × 0.5 = 300. Ages 40–44 are half of the 40–49 bucket: 1,000 × 0.5 = 500. Total 800. Taking both full buckets gives 1,600, and ignoring the histogram (2,000 × 10/30) gives about 667.

1283. **D** — Index traversal: 3 × 4.1 = 12.3 ms. The 50 matching blocks are read sequentially: 50 × 0.1 = 5 ms. Total 17.3 ms. 217.3 ms charges a seek for every data block, which happens with a secondary index, not a clustering one.

1284. **A** — Each disjunct has its own index, so SQLite runs one index search per disjunct and unions the rowids, reporting MULTI-INDEX OR. This is Silberschatz's disjunctive selection by union of identifiers. If either attribute lacked an index, a full scan would be needed.

1285. **C** — Index: (hi + n) × (tT + tS) = 8 × 4.1 = 32.8 ms. Linear scan: 4 + 1,000 × 0.1 = 104 ms. With few matches the index wins easily. A secondary index loses only when the number of matches is large, because each match may need its own seek.

1286. **C** — Wrapping the column in a function makes the predicate non-sargable: the index is ordered by name, not by upper(name). An expression index such as CREATE INDEX ON emp(upper(name)), or comparing the stored column directly, makes an index search possible.

1287. **D** — The estimate of 10 rows came from statistics gathered before the load. With such a small estimate a nested-loop join looks cheap. Refreshing the statistics lets the optimiser see 2 million rows and choose, for example, a hash join.

1288. **D** — UNION has set semantics, so it eliminates duplicates with a sort or hash step. When the inputs are known to be disjoint, UNION ALL returns the same rows without that step. INTERSECT would return different rows, and UNION ALL builds no index.

1289. **D** — r1 and r3 have no common attribute, so joining them first produces n_r1 × n_r3 tuples. The other orders always join relations that share an attribute. Avoiding Cartesian products is a standard heuristic.

1290. **A** — Block nested-loop with s outer: ⌈100/10⌉ × 1,000 + 100 = 10,100; with r outer: ⌈1,000/10⌉ × 100 + 1,000 = 11,000. Hash join: n_h = ⌈100/12⌉ = 9 and M = 12 > n_h + 1 = 10, so no recursive partitioning, and it costs 3,300. Merge join costs 1,100 only if the inputs are already sorted, which they are not.

1291. **D** — Index nested-loop: 100 + 5,000 × 3 = 15,100. Block nested-loop with 50 outer buffer blocks: ⌈100/50⌉ × 2,000 + 100 = 4,100. With plenty of memory, two scans of s beat 5,000 index lookups. 200,100 is the one-buffer worst case of block nested-loop.

1292. **D** — For AND, multiply selectivities: 10,000 × (500/10,000) × (200/10,000) = 10,000 × 0.05 × 0.02 = 10. Adding sizes is never right for AND; the result of an AND cannot be larger than either part.

1293. **B** — P1 totals 3,000 + 2,000 = 5,000. P2's output is sorted on the join attribute, an interesting order, so it costs 4,200 with no extra sort. Comparing join steps alone misses the sort that P1 still needs.

1294. **D** — With s as the build input, n_h = ⌈200/20⌉ = 10 and 20 > 11, so one partitioning pass works and the cost is 3 × (2,000 + 200) = 6,600. With r as build input, n_h = 100 and 20 is not greater than 101, so recursive partitioning would be needed.

1295. **D** — A composite index is sorted by dept first and by salary only within each dept, so salary = 5000 entries are spread throughout it. Without a condition on dept the index cannot be searched, which is the leftmost-prefix rule. An index on (salary) or (salary, dept) would allow a SEARCH.

1296. **C** — Silberschatz's conjunctive selection with one index picks the most selective condition for the index lookup and checks the rest in memory. Fetching 20 tuples through a secondary index costs far less than fetching 2,000, each possibly needing its own seek.

1297. **C** — Each takes row with a non-null ID matches exactly one student, because ID is the key of student and the foreign key guarantees the student exists. A NULL never equals anything, so those 1,000 rows join with no one: 10,000 − 1,000 = 9,000.

1298. **A** — Correlated: 100 + 1,000 × 20 = 20,100 transfers, since dept is rescanned for every emp tuple. Hash join: 3 × (100 + 20) = 360. Decorrelation removes the per-tuple rescans.

1299. **C** — With (dept, salary) SQLite searches dept = 'CS' and the matching entries are already in salary order, so no sort is needed (SEARCH emp USING INDEX x (dept=?)). An index on dept alone still needs USE TEMP B-TREE FOR ORDER BY. Indexes led by salary give SCAN emp USING INDEX x, which reads the whole table.

1300. **C** — The SQL form is GRANT <privilege list> ON <object> TO <user or role>. Reading rows needs the SELECT privilege. SQL has no READ table privilege and no ALLOW statement, and the grantee comes after TO.

1301. **D** — WITH GRANT OPTION lets the grantee grant the same privilege onward. It does not transfer ownership, add other privileges such as UPDATE, or protect the privilege from REVOKE.

1302. **B** — In DAC, the owner of an object decides, at their discretion, who gets which privileges, using GRANT and REVOKE. Labels and clearances enforced by the system describe mandatory access control. Encryption and firewalls are other security layers, not access control models.

1303. **A** — MAC assigns security classes (such as Top Secret, Secret, Confidential, Unclassified) to objects and clearances to subjects, and the system enforces rules such as Bell-LaPadula. Owner discretion is DAC, and job-based privileges describe RBAC.

1304. **C** — SQL injection happens when an application builds SQL by joining strings with untrusted input. Specially crafted input closes a string or adds operators so the database runs a different query, for example one that skips a password check.

1305. **D** — With parameterised queries the SQL structure is fixed first and the input is bound as a value, so it can never become SQL code. Hiding errors and renaming tables only slow attackers down, and HTTPS protects data in transit but not the query itself.

1306. **A** — CREATE ROLE teacher creates a named set of privileges. Granting privileges to the role and the role to users makes administration easier: when staff change jobs you change their roles, not dozens of individual grants. This is role-based access control (RBAC).

1307. **D** — Authentication verifies identity, for example with a password or certificate. Authorisation then checks the privileges of that identity, for example whether it holds SELECT on a table. A user can be authenticated yet still not authorised for an action.

1308. **A** — PUBLIC stands for all users of the database, including users created later. It does not publish data outside the database. Use it with care, because REVOKE from PUBLIC does not remove privileges granted to users individually.

1309. **D** — An audit trail records security-relevant actions, such as logins, privilege changes and updates to sensitive tables, with the user and time. It helps detect misuse and trace who made a change. Backups support recovery, not accountability.

1310. **A** — The simple security property is 'no read up': reading requires clearance ≥ classification. The *-property (star property) controls writing: 'no write down', so a subject may write only to objects at or above its own clearance.

1311. **D** — Even a legitimate Secret-cleared subject (or a Trojan horse running as that subject) could leak data by writing it to an Unclassified object. Forbidding writes to lower classes blocks that flow. 'No read up' is the separate simple security property.

1312. **A** — For example, a view showing only name and dept from emp, granted to a clerk, hides salaries. The clerk needs SELECT on the view, not on emp. Views do not encrypt or audit anything by themselves.

1313. **B** — If an attacker can query aggregates over sets that differ by one person, subtracting the results reveals that person's value. Defences include minimum query-set sizes, limiting overlap between queries and adding noise.

1314. **C** — If user U declares a foreign key referencing dept(dept_id), the owner can no longer delete a dept row that U's table still references. Because a foreign key can restrict another user's actions, creating one needs the REFERENCES privilege.

1315. **A** — Data-at-rest encryption makes storage media and backups unreadable without the key. It does not stop SQL injection or insider misuse, because queries through the DBMS see decrypted data. Concurrent updates are a concurrency-control issue.

1316. **B** — The query becomes WHERE name = 'ali' AND pass = '' OR '1'='1'. AND binds tighter than OR, so this is (name = 'ali' AND pass = '') OR '1'='1', which is true for every row. All 3 rows are returned. The query is valid SQL, so there is no error.

1317. **C** — admin'-- closes the string after admin and starts a comment, so the query becomes WHERE name = 'admin' and the password test is ignored. A double quote does not close a single-quoted string, --admin is just a name, and AND '1'='2' makes the condition false. Checked in SQLite.

1318. **B** — SQL allows column-level UPDATE privileges, written with the column list after the privilege: GRANT UPDATE (salary) ON emp TO ali. GRANT has no WHERE clause, and ALTER would concern changing the table's structure, not its data.

1319. **B** — B's privilege depended entirely on A's grant. CASCADE removes A's privilege and every privilege that can no longer be traced back to the DBA in the authorisation graph, so B loses SELECT too.

1320. **D** — In the authorisation graph, C has two incoming grants. Removing A removes the edge A → C, but the path DBA → B → C still exists, so C keeps SELECT. B is unaffected because its grant came straight from the DBA.

1321. **B** — RESTRICT tells the DBMS to refuse the revoke if it would cause cascading revocation of other privileges. B's grant can only be traced back through A, so the statement returns an error and nothing changes. CASCADE would remove both.

1322. **C** — REVOKE GRANT OPTION FOR removes only the right to pass the privilege on. A keeps SELECT itself. With CASCADE, any grants A had made using that option would also be revoked, but A has made none here.
1323. **A** — The ? placeholder binds u as a plain string value, so SQLite looks for a user literally named ' OR '1'='1. No such user exists, so 0 rows are returned. The quote characters never become part of the SQL code.
1324. **B** — The query becomes SELECT * FROM orders WHERE id = 0 OR 1=1. The condition 1=1 is true for every row, so all 4 orders are returned. No quote was needed, because the value is inserted in a numeric position.
1325. **A** — No read up: Secret may read Secret, Confidential and Unclassified objects, but not Top Secret. No write down: Secret may write only Secret or Top Secret objects, so writing Confidential or Unclassified objects is forbidden.
1326. **C** — The view exposes only the permitted columns, and the clerk receives SELECT on the view alone. Standard SQL cannot revoke part of a table-level SELECT by naming one column (the syntax shown is invalid), and the last two options still let the clerk read salaries.
1327. **C** — A role is a named set of privileges, and members get whatever the role currently holds. Adding SELECT to teacher after ali joined still gives ali that privilege. WITH ADMIN OPTION only controls whether ali can grant the role to others.
1328. **B** — UNION needs union-compatible inputs. The original SELECT returns two columns, so the attacker must return two as well, for example ' UNION SELECT name, pass FROM users--, which (checked in SQLite) lists every username and password beside the products.
1329. **C** — After A is removed, B and C still grant to each other, but that cycle no longer connects to the DBA. The SQL standard keeps a privilege only if a path from the root of the authorisation graph supports it, so B and C both lose SELECT.
1330. **D** — The two query sets differ by exactly one person, so subtracting the sums gives that person's salary: 1,250,000 − 1,040,000 = 210,000. This is an inference (tracker) attack; defences limit overlap between query sets or add noise.
1331. **B** — With a prepared statement, the DBMS parses the query with placeholders first; bound values are treated purely as data. Escaping depends on handling every context correctly (quotes, numeric positions, character sets), and one missed case reopens the hole.
1332. **D** — Escaping quotes only protects values inside a quoted string. Here the input is pasted straight into a numeric position, so 0 OR 1=1 becomes valid SQL logic. A bound parameter or converting the input to an integer first would stop it.
1333. **A** — Data that was safe to store can still be dangerous when reused. Concatenated into the UPDATE, the stored name ends the string and comments out the rest, so the query becomes WHERE name = 'admin' and changes the admin's password. Every query needs parameters, including those using data from the database.
1334. **D** — Least privilege limits what any successful attack can do. An account without DROP or ALTER rights cannot destroy tables even if another injection flaw appears. Hiding errors and renaming tables only slow attackers, and the grant option would widen the account's power.
1335. **C** — B's grant option came only through A, so it disappears; B's plain SELECT survives through the direct DBA → B grant. C's privilege was granted by B using that grant option, so it is no longer supported and is revoked by the cascade.
1336. **C** — Adding a direct DBA → B grant gives B a second path in the authorisation graph. Cascading revocation of A then removes A → B, but B keeps SELECT through the new edge. RESTRICT would simply fail while B depends on A.
1337. **A** — Stored procedures help only when they use static SQL or bind parameters. If the procedure concatenates input into a string and executes it, the same quote tricks work inside the database. Running it as the DBA would make an injection more damaging, not less.
1338. **B** — Each true/false reply answers one yes/no question about hidden data. Repeating this for each position recovers the whole password one character at a time. This is blind injection: no data is displayed, but the injected condition clearly runs (checked in SQLite).
1339. **D** — Under DAC, B owns the new table and may grant it freely, so data leaks without breaking any rule. Under MAC, the copy target would need a classification at least as high as the salary data, and C without clearance could not read it.
1340. **D** — Availability means authorised users can reach the data when they need it. A denial-of-service attack violates it, even though no data is read (confidentiality) or changed (integrity).

1341. **B** — The creator of a table is its owner and holds all privileges on it, with the ability to grant them. That is the basis of discretionary access control: the owner decides who else gets access.
1342. **D** — ALL PRIVILEGES is shorthand for all the privileges the grantor holds and can grant on that one object, such as SELECT, INSERT, UPDATE, DELETE and REFERENCES. It does not transfer ownership or extend to other tables.
1343. **C** — REVOKE removes privileges given by GRANT, for example REVOKE SELECT ON marks FROM ali. ROLLBACK undoes a transaction's changes, and DROP USER deletes the whole account rather than one privilege.
1344. **A** — Least privilege limits the damage from mistakes, insider misuse and compromised accounts. A reporting tool, for instance, needs SELECT on a few tables, not the right to DROP them.
1345. **A** — With RBAC the privileges are defined once per job function. When a person changes jobs, an administrator revokes one role and grants another instead of revoking and granting many individual privileges.
1346. **C** — MAC gives every object a classification and every subject a clearance from the same ordered set of levels, such as Top Secret > Secret > Confidential > Unclassified. The system compares the two on every access.
1347. **C** — TLS encrypts data in transit, so someone capturing network packets cannot read queries or results. Encryption at rest protects stored files and backups, not data on the wire.
1348. **D** — A salted, deliberately slow hash cannot be reversed, and the salt makes identical passwords hash differently. Plain text and reversible encryption expose every password if the table leaks, and fast unsalted hashes fall to precomputed tables.
1349. **C** — In time-based blind injection the page shows no data and no errors, so the attacker learns each true or false answer from the delay. UNION-based injection reads data directly in the page, and error-based injection reads it from error messages.
1350. **D** — Separation of duties stops one insider from committing fraud alone. Collusion between two people would then be needed, which is less likely and easier to detect.
1351. **C** — Integrity means data is changed only by authorised users in authorised ways. Unauthorised modification breaks it even if nobody outside sees the data.
1352. **B** — Data masking hides part of a sensitive value in what a user sees while the stored value stays complete for systems that need it. Transparent data encryption protects the stored files and shows full values to any authorised query.
1353. **D** — The access matrix records who may do what to which object. GRANT and REVOKE add rights to and remove them from its cells, and access control lists are its columns stored with each object.
1354. **C** — The ordinary *-property forbids writing down but allows writing up. The strong version allows writing only at the subject's own level, which also stops a low subject from overwriting or corrupting higher-level data.
1355. **D** — Polyinstantiation lets a Secret tuple and an Unclassified tuple share a key. If the system rejected the lower-level insert instead, the rejection itself would tell a low user that hidden data exists.
1356. **B** — Query-set-size control blocks small result sets, but the difference between two large sets can be a single individual. Trackers combine permitted queries to rebuild values the control was meant to hide, so extra measures such as noise are needed.
1357. **B** — With perturbation each aggregate carries a small random error. Large-group statistics remain accurate enough, but the differences attackers use to isolate one record are swamped by the noise. Differential privacy formalises this idea.
1358. **D** — Row-level security attaches a filter policy to a table, so each user sees only the rows the policy allows, whatever query they write. It is like giving each user a personal view, enforced by the DBMS rather than the application.
1359. **D** — Without the check option, a user who may update only through a CS view could set dept = 'EE' or insert EE rows, which then vanish from the view. WITH CHECK OPTION rejects such changes, keeping users inside the slice the view was meant to give them.
1360. **D** — Covert channels break the no-write-down rule without an explicit write, for example a Secret process making a lock or disk busy or idle to signal bits. Access control on objects does not stop them, so they must be analysed separately.
1361. **D** — Creating a view is not a way to gain rights. A view's creator gets only the privileges that follow from their privileges on the base relations. Without UPDATE on emp, for example, they cannot update through the view.
1362. **A** — If a dean grants privileges as an individual, revoking the dean's account cascades and removes every grant they made. Granting as the role, which the next dean also holds, keeps those grants in place.

1363. **D** — Placeholders bind data values only. The statement's structure, including table and column names and keywords such as ASC or DESC, must be fixed when it is prepared. Any dynamic identifier should be checked against an allow-list of real names.

1364. **B** — REVOKE removes only the named privilege, so ali keeps SELECT, INSERT and the rest. REVOKE ALL PRIVILEGES would remove everything, and the other two forms are not valid SQL.

1365. **B** — Adding rows needs INSERT and removing rows needs DELETE. ALL PRIVILEGES would also give SELECT and UPDATE, against least privilege, and UPDATE changes rows rather than removing them.

1366. **C** — The *-property allows writing only at or above the subject's level. Writing up to Secret is permitted; writing down to Unclassified is forbidden, because it could leak Confidential information.

1367. **C** — A column-level UPDATE privilege covers only the listed columns. Changing dept needs UPDATE on dept (or on the whole table), so the DBMS refuses the statement.

1368. **B** — A's own grant from the DBA is untouched. B loses SELECT because A's grant was B's only source, and the cascade removes C's privilege, which depended on B.

1369. **A** — C appears in the authorisation graph with two incoming edges. A's revoke removes only the edge from A. The edge from B, which itself has a path from the DBA, still supports C's privilege.

1370. **B** — Only a user who holds a privilege WITH GRANT OPTION, or the owner, may pass it on. B can use SELECT but cannot grant it, so the DBMS rejects the grant (some systems give a warning and grant nothing).

1371. **A** — Roles can be granted to roles, forming a hierarchy. dean holds instructor, so dean's members get instructor's privileges as well, and sara therefore can read marks.

1372. **B** — Privileges obtained through a role disappear when the role is revoked, but privileges granted directly to the user are separate grants and remain. Role privileges are not copied to members.

1373. **C** — The query becomes ... WHERE name = 'zzz' UNION SELECT name, pass FROM users--'. No product is called zzz, so the first part returns 0 rows; the UNION adds the 3 user rows. Both sides return two columns, so the UNION is valid and the trailing quote is commented out.

1374. **A** — The view keeps only the 3 CS rows and only the name and dept columns, so SELECT * on it returns 3 rows of 2 columns. This is how a view restricts both rows and columns, hiding salary from the clerk.

1375. **B** — The allowed range is $5 \le \text{size} \le 95$. A set of 3 is too small, and sets of 97 or 98 are too large, because their complement (3 or 2 records) could be found by subtracting from the total.

1376. **A** — Secret ≥ Confidential and {Nuclear, Army} ⊇ {Nuclear}, so S dominates the first object. S lacks the Navy category for two of the others, and Top Secret is above S's level.

1377. **A** — A column-level INSERT privilege lets the user supply values only for the listed columns; the others get their default or NULL. Supplying salary needs INSERT on salary, and UPDATE or SELECT need their own privileges.

1378. **C** — Only the first call sends oid as a bound parameter, separate from the SQL text, so it can never change the query's structure. The % and f-string forms paste the input into the SQL, and hand-escaping is fragile and easy to get wrong.

1379. **D** — The delay happens only when the CASE condition is true, so a 5-second delay means substr(pass,1,1) = 'a'. This is time-based blind injection: the response content may be identical, but the timing leaks one bit per request.

1380. **B** — After the first revoke A is gone, but C keeps SELECT through B, and D through C. After B revokes C, C has no incoming edge from anyone who still holds the privilege, so C loses it and the cascade removes D's. Only B remains.

1381. **A** — Top Secret ≥ Secret and {Nuclear, Army} ⊇ {Nuclear}, so the first object dominates S and writing there cannot leak Nuclear data downwards. (Secret, {}) lacks Nuclear, Confidential is lower, and (Top Secret, {Army}) also lacks Nuclear.

1382. **B** — Unlike execute(), executescript() runs several statements separated by semicolons. The input closes the INSERT, adds DROP TABLE logs, and comments out the rest, so logs is dropped and only notes is left. Stacked-query injection depends on such multi-statement APIs.

1383. **D** — Total before = 4 × 50,000 = 200,000. Total after = 5 × 52,000 = 260,000. The difference, 60,000, is Dr Asif's salary. Two permitted averages over groups of 4 and 5 have leaked one person's value.

1384. **B** — A may create the view because A has SELECT on emp. But A's rights on v come from its rights on emp, and A cannot grant on emp. Letting A grant on the view would leak emp data to B, so the grant is refused.

1385. **A** — Transparent encryption decrypts data as the DBMS reads it, so any query the DBMS accepts, including an injected one, gets plaintext. It defends against stolen disks and backups, not against abuse of the application's own connection.

1386. **C** — An audit trail is only useful if each account maps to one person or program. Individual accounts, with least privilege for each, also limit what any single compromised credential can do.

1387. **C** — Client-side checks run on the attacker's machine and can be skipped entirely. Validation must happen on the server, and the real fix is parameterised queries.

1388. **A** — A unique salt per user makes each hash different and defeats precomputed rainbow tables. SHA-256 is also fast, which helps guessing attacks; a slow function such as bcrypt or Argon2 is preferred.

1389. **A** — A view restricts access only if it is the user's sole route to the data. Any direct SELECT on emp, whether given to clerk, a role clerk holds or PUBLIC, gives the salary column straight away.

1390. **A** — Writing up is allowed because it cannot leak Secret data downwards, so appending is fine. Reading would be reading up, which the simple security property forbids. This blind write-up is why the strong *-property exists for integrity concerns.

1391. **A** — A loses its privilege. C's edge from A goes, but C still has an edge from D, and D has an edge from B, which comes from the DBA. The path DBA → B → D → C keeps both C and D. The C–D cycle alone would not, but here it is anchored through B.

1392. **C** — The parameter is bound safely, so the query's structure cannot change. But LIKE treats % and _ as pattern characters in the value itself, so a lone % matches every name. The fix is to escape wildcards (for example with an ESCAPE clause) or use = when an exact match is meant.

1393. **C** — Prepared statements protect only the values passed separately through placeholders. Here u becomes part of the SQL text that is prepared, so input such as ' OR '1'='1 changes the query. The fix is WHERE name = ? with u bound as a parameter.

1394. **B** — If each answer is true value + fresh independent noise, the average of many answers converges on the true value. Systems counter this by making noise consistent for repeated queries or by capping queries, as in a differential-privacy budget.

1395. **D** — A syntax error caused by one quote proves that input is concatenated into SQL, and the echoed fragment even shows the query's shape. Parameterised queries remove the flaw, and generic error pages stop the leak of query details that helps error-based injection.

1396. **A** — An audit trail must survive the attack it records. If the same credentials that run the app can delete log rows, an attacker can cover their tracks. Granting only INSERT on the log, or shipping logs to a separate store, keeps them trustworthy.

1397. **B** — The duplicate-key error itself is information flowing from Secret to Unclassified. Storing a second tuple with the same key at the lower level avoids the leak, at the cost of each level seeing its own version.

1398. **D** — Binary search finds one character out of 64 in log2 64 = 6 questions, so 8 characters need 8 × 6 = 48. Testing each candidate one at a time needs up to 8 × 64 = 512 queries.

1399. **C** — The WHERE clause becomes item = ? AND owner = 'x' OR '1'='1'. AND binds tighter than OR, so it means (item = 'pen' AND owner = 'x') OR '1'='1', which is true for every row. All 4 rows are deleted. One concatenated value is enough to make the whole statement injectable.

1400. **D** — A distributed database is one logical database spread across sites. A distributed DBMS manages it so that, ideally, users query it as if it were stored in one place. Unrelated databases or simple backups are not distributed databases.

1401. **A** — A horizontal fragment is a selection σcondition(r), so each fragment holds complete rows that satisfy its condition. Splitting by columns is vertical fragmentation.

1402. **C** — Vertical fragments are projections of the relation. Repeating the key in each fragment lets the DBMS rebuild r with a natural join on the key. Without it, the join can create spurious tuples. UNION is used for horizontal fragments, not vertical ones.

1403. **D** — Each horizontal fragment holds some rows of r, so r = r1 ∪ r2 ∪ ... ∪ rn. Natural join on the key rebuilds vertical fragments. Intersection would keep only rows in every fragment, which is empty when fragments are disjoint.

1404. **B** — With fragmentation transparency a user writes SELECT ... FROM customer, and the DBMS works out which fragments to read. Related forms are location transparency (users need not know where data is) and replication transparency (users need not know how many copies exist).

1405. **A** — CAP, stated by Brewer and proved by Gilbert and Lynch, concerns Consistency (every read sees the latest write), Availability (every request to a working node gets a response) and Partition tolerance (the system keeps working when the network splits).

1406. **C** — Full replication makes reads fast and local and keeps data available when a site fails. The cost is on updates: every copy must be updated and kept consistent, which adds messages and co-ordination.

1407. **A** — 2PC gives atomicity to distributed transactions: phase 1 asks every participant to prepare and vote, and phase 2 sends the global commit or abort. Two-phase locking (growing and shrinking phases) is a different protocol, for serialisability.

1408. **D** — In a homogeneous system all sites run the same DBMS and co-operate fully. In a heterogeneous (often federated or multidatabase) system, sites may use different DBMSs, schemas or data models, so translation and co-ordination are harder.

1409. **A** — Eager replication updates every copy as part of the transaction, so all replicas are consistent at commit. Asynchronous (lazy) replication commits first and propagates changes later, which is faster but allows stale reads.

1410. **B** — Completeness: every tuple or attribute appears in some fragment. Reconstruction: the original relation can be rebuilt (union or join). Disjointness: horizontal fragments do not overlap, and vertical fragments share only the key.

1411. **A** — For example, if customer is split by region, orders can be split as orders ⋉ customer_north and orders ⋉ customer_south. Each order then sits at the same site as its customer, which makes local joins possible. Mixing horizontal and vertical is called mixed (hybrid) fragmentation.

1412. **C** — After voting ready, a participant may not decide alone: the coordinator might have chosen commit or abort. If the coordinator fails and no other participant knows the decision, the participant stays blocked with its locks held. Three-phase commit reduces this under certain failure assumptions.

1413. **A** — When the network is healthy a system can be both consistent and available. During a partition, a node that cannot reach the others must either refuse requests (keeping consistency) or answer with possibly stale data (keeping availability). Disk failures are a separate issue.

1414. **D** — BASE systems put availability first, allow state to change without input as replicas converge (soft state), and promise only that replicas become consistent eventually if updates stop. Many AP-style NoSQL stores follow this approach.

1415. **B** — $r \ltimes s = \pi_R(r \bowtie s)$, the tuples of r that have a match in s. A site ships the join attribute values of s, filters r locally, and ships only matching r tuples back. That pays off when few r tuples match.

1416. **D** — $R + W > N$ gives $R > 5 - 3 = 2$, so $R = 3$ is the smallest. With $R = 2$, a read could hit two replicas that the last write missed ($R + W = N$). $R = 4$ also works but is not the smallest. Checked by enumerating every read and write set.

1417. **B** — Both fragments keep the key id, so a natural join on id puts each employee's attributes back together. UNION needs union-compatible fragments, and the Cartesian product would pair every row of F1 with every row of F2.

1418. **A** — Neither condition covers salary = 50,000, so such a row is lost, and the union of the fragments no longer equals staff. The two conditions never overlap, so disjointness holds. Testing on a sample table with one 50,000 row confirms one tuple goes missing.

1419. **C** — Both conditions include age = 30, so those rows are stored twice. Every row is still covered and the union still gives staff, so completeness and reconstruction hold. Overlap causes duplicate updates unless the overlap is deliberate replication.

1420. **A** — $R = 3$, $W = 5$: $8 > 7$ and $10 > 7$, so both rules hold. (4, 3) and (2, 5) sum to 7, so a read can miss the latest write. (5, 3) passes $R + W > N$, but $2W = 6$ is not > 7, so two writes could go to disjoint sets of replicas.

1421. **B** — Joining on name pairs each Ali row in F1 with each Ali row in F2 ($2 \times 2 = 4$) and Sara with Sara (1), giving 5 tuples. Two are spurious, such as (Ali, CS, 200). Because name is not a key, the fragments cannot rebuild emp.

1422. **D** — Sending the smaller relation to the site where the larger one lives and where the result is needed costs $100 \times 50 = 5{,}000$ bytes. Sending emp to Karachi costs 1,000,000 bytes, and the result would then have to come back to Lahore as well.

1423. **D** — S2 sends $\pi_B(s) = \{10, 30, 50\}$ to S1. $r \ltimes s$ keeps the r tuples with B in that set: (1, 10), (3, 30) and (4, 10), so 3 tuples go to S2 instead of 5. The answer 2 counts distinct matching B values, not tuples.

1424. **C** — Phase 1: 4 prepare messages and 4 votes. Phase 2: 4 commit messages and 4 acknowledgements. That is 4 × 4 = 16. Counting only one phase gives 8, and leaving out the acknowledgements gives 12.
1425. **D** — 2PC needs a unanimous ready vote to commit. One abort vote forces a global abort, so atomicity holds: the transaction takes effect nowhere. A participant that has voted abort can abort at once and does not change its vote.
1426. **D** — ROWA is the quorum case R = 1, W = N. A read needs any single live copy, so it still works. A write needs all 4 copies, so it cannot complete while one is down. This is why ROWA hurts write availability.
1427. **A** — Reconstruction undoes the steps in reverse: first natural-join each region's vertical pieces on cid to rebuild C_N and C_S, then union them. The four fragments are not union-compatible, and joining a north piece with a south piece on cid finds no matches.
1428. **B** — Qr + 4 > 7 needs Qr ≥ 4. With Qr = 3, the sites {2, 1} (weight 3) and {2, 1, 1} (weight 4) do not overlap, so a read can miss a write. Qw = 4 also meets 2Qw > S (8 > 7). Checked by enumerating all site subsets.
1429. **B** — With R = 1 and W = 1, R + W = 2 is not greater than 3, so a read can go to a replica the write did not reach. The other settings all give R + W > 3, so every read set meets every write set. Many Dynamo-style stores let you tune these values.
1430. **C** — Refusing requests on the minority side means some working nodes cannot answer, so availability is sacrificed to keep data consistent: CP. An AP system would accept writes on both sides and reconcile them later. CA is not a real choice once partitions can happen.
1431. **B** — Horizontal fragments by branch put each row where it is used, so most queries run locally. Vertical splitting would force joins across cities for nearly every query. A central copy makes almost every query remote, and full replication makes every update go to three sites.
1432. **C** — The coordinator may already have logged a commit, or an abort (for example on timeout), before crashing. If the participants guessed, they could disagree with that logged decision and break atomicity. With every survivor in the ready state and no decision known, 2PC blocks.
1433. **B** — The write may update replicas {1, 2, 3} and the read may go to {4, 5}. These sets are disjoint because R + W = N. Raising R to 3 (or W to 4) makes every read set overlap every write set.
1434. **C** — Two students with the same name cause every pairing of their dept and cgpa values to appear. Vertical fragments must share a key (roll_no here) so the join is lossless. Sharing the key is allowed by the disjointness rule for vertical fragments.
1435. **C** — Shipping emp directly costs 100,000 bytes. The semijoin ships πdept_id(dept) (50 × 4 = 200 bytes) to S1, but every emp row matches, so all 100,000 bytes of emp still travel: 100,200 in total. Semijoins pay off only when many tuples are filtered out.
1436. **D** — Combining the local graphs gives T1 → T2 → T1, a cycle. Neither transaction can proceed, yet no single site sees the cycle. Distributed systems detect this with a global (or centralised) wait-for graph, timeouts, or prevention schemes such as wait-die.
1437. **C** — CAP concerns behaviour during network partitions. Its consistency means every read sees the most recent write (linearisability), which differs from ACID consistency (integrity constraints hold). A single-site database has no network partition between replicas to tolerate.
1438. **A** — Asynchronous (lazy) replication acknowledges commits before replicas have the change. If the primary fails in that window and the replica takes over, the committed update is missing. Synchronous replication avoids this at the cost of commit latency.
1439. **B** — A semijoin keeps only orders whose cust_id matches a customer in that fragment. Order 104 matches none, so it vanishes from every fragment and the union no longer equals orders. Derived fragmentation is complete only when referential integrity holds.
1440. **A** — A local transaction touches data at a single site only. A global transaction accesses data at two or more sites, so it needs distributed concurrency control and an atomic commit protocol such as 2PC.
1441. **B** — With location transparency the query uses only the logical name, and the DDBMS looks up where the data lives in its catalogue. The user does not write site names into queries.
1442. **C** — Replication transparency lets users treat a data item as one logical object. The system decides which copy to read and makes sure updates reach the other copies. Hiding the split into pieces is fragmentation transparency.
1443. **D** — In a homogeneous distributed database every site runs the same DBMS product and the sites cooperate on queries and transactions. Storing the same data everywhere is full replication, which is a separate choice.

1444. **A** — Partial replication sits between no replication (one copy of each item) and full replication (every item everywhere). Frequently read items are copied to the sites that read them most.
1445. **B** — Lazy replication commits the update at one copy first and ships it to the other copies afterwards. Writes are fast, but a replica can serve stale data for a while, and a crash can lose changes that were not yet sent.
1446. **C** — One copy is named the primary. All writes go to it, and it then propagates them to the secondary copies. This avoids conflicting concurrent updates on different replicas, at the cost of a single point of control.
1447. **D** — Mixed fragmentation first splits a relation one way, for example horizontally by region, and then splits each piece the other way, for example vertically by column group. Rebuilding uses joins and unions in the reverse order.
1448. **A** — Local autonomy means a site controls its own data, users and administration and can keep working on local data even when other sites are down. Full autonomy is a feature of federated or multidatabase systems.
1449. **B** — After a relation is fragmented, the designer must place each fragment, and any copies, at sites. A good allocation keeps data near the queries that use it and limits network traffic and update cost.
1450. **C** — Over a wide-area network, shipping data between sites is far slower than local disk or CPU work, so textbook optimisers try to minimise the bytes sent. Techniques such as semijoins exist for this reason.
1451. **D** — Eventual consistency only promises convergence once updates stop. Between an update and that point, different replicas can return different values, which is why reads may be stale.
1452. **A** — In 3PC no participant commits until all have acknowledged the pre-commit. A participant that times out can therefore infer the outcome from its own state and the states of others, which removes 2PC's blocking under site failures. It still assumes no network partitions, and it costs more messages, not fewer.
1453. **B** — PACELC reads: if Partition, choose Availability or Consistency; Else, choose Latency or Consistency. Waiting for remote replicas keeps data consistent but makes every operation slower, even on a healthy network.
1454. **C** — A ready vote is a promise to commit if told to. To keep that promise after a crash, the participant must first write its log records and a ready record to stable storage. It keeps its locks until the decision arrives.
1455. **D** — 2PC commits only if every participant votes ready. Before the decision is made, the coordinator is free to treat silence as a no vote and abort, which is safe because no site can have committed yet.
1456. **A** — Each site sends its local wait-for edges to one coordinator, which builds the global wait-for graph. A cycle in that graph, possibly spanning several sites, means a deadlock, and a victim is chosen to roll back.
1457. **B** — Two sites can generate the same local value, so the site identifier is appended as the lower-order part. Comparing the local value first and the site id second gives a total order with no ties.
1458. **C** — Primary-copy locking treats the item as unreplicated for concurrency control: its lock manager lives at the primary site. It is simple, but if that site fails the item cannot be locked.
1459. **D** — The majority protocol needs locks on more than n/2 replicas. Two conflicting lock holders would need two majorities, and any two majorities overlap, so conflicts are detected while some replicas may be down.
1460. **A** — A federated system integrates pre-existing, autonomous local databases, which may differ in DBMS and schema, behind a common interface. Each local database keeps control of its own data.
1461. **B** — The biased protocol favours reads: a reader locks any one copy, but a writer must lock all copies. Reads become cheap, while writes cost more and fail if any replica is unavailable.
1462. **C** — A minterm combines all simple predicates with AND, each one either as written or negated. With two predicates there are 2^2 = 4 minterms, and each defines a candidate fragment. Minterms are disjoint and together cover all rows, which gives a correct fragmentation.
1463. **D** — If a release message reaches the detector after a new wait edge, the detector may see an edge that has already gone and report a cycle. It may then roll back a transaction that was never deadlocked.
1464. **D** — The rule is R + W > N, so R > 9 - 6 = 3, giving R = 4. With R = 3, a read could pick the 3 copies the write missed and see old data.
1465. **A** — 2W > 5 means W > 2.5, so W = 3. With W = 2, two writes could succeed on disjoint pairs of replicas and conflict.
1466. **B** — The query predicate matches the fragment's defining predicate, so only F_KHI can contain qualifying rows. This is how horizontal fragmentation lets the optimiser skip whole sites.
1467. **C** — Complete and disjoint horizontal fragments partition the rows, so their sizes add up to the relation's size: 1,000 - 420 - 310 = 270.

1468. **D** — The replicas work in parallel, and the commit waits for the last acknowledgement, so the slowest replica sets the time: 80 ms. Adding the times (135 ms) assumes they run one after another.
1469. **A** — name is only in F1 and salary is only in F2, so both are needed. Vertical fragments are combined with a join on the shared key id, not a union.
1470. **B** — S = 3 + 2 + 2 + 1 + 1 = 9. Qr + Qw > 9 gives Qw > 5, so Qw ≥ 6. 2Qw > 9 gives Qw ≥ 5. The stricter bound wins: Qw = 6.
1471. **C** — R = 3, W = 6 gives 9 > 8 and 12 > 8. R = 4, W = 4 and R = 5, W = 4 fail 2W > N because 8 is not greater than 8. R = 2, W = 6 gives R + W = 8, which is not greater than 8.
1472. **D** — A write needs 3 live replicas. With 5 replicas, up to 5 - 3 = 2 can fail. Quorums trade some availability for consistency, unlike write-all, which fails if even one replica is down.
1473. **A** — Send πemp_id(proj) to S1: 400 × 4 = 1,600 bytes. Send the matching rows, emp ⋉ proj, back to S2: 600 × 80 = 48,000 bytes. Total 49,600 bytes, against 400,000 bytes to ship all of emp.
1474. **B** — Shipping the smaller relation to the larger one and then sending the small result costs 200 + 50 = 250 KB. Sending only S to S3 cannot produce the join, because R is still at S1.
1475. **C** — The semijoin keeps each order whose cust_id appears in customer_north: the orders for customers 1, 3, 3, 5 and 1, which is 5 rows. Duplicate customers do not collapse, because each order is a separate row.
1476. **D** — Each predicate appears true or negated, so 2 predicates give 2^2 = 4 minterms: CS and low salary, CS and not low, non-CS and low, non-CS and not low.
1477. **A** — Every row is covered (CS rows by the first fragment, everything else by the third), so completeness holds and union rebuilds staff. EE rows satisfy both dept = 'EE' and dept ≠ 'CS', so they are stored twice, which breaks disjointness.
1478. **B** — After voting ready, the participant gave up its right to decide alone. The coordinator may have chosen either outcome, so the participant must find out the decision before it commits or aborts T.
1479. **C** — Merging the edges gives T1 → T2 → T3 → T4 → T1, one cycle through all four transactions. No single site sees it, which is why a global wait-for graph is needed.
1480. **D** — A majority of 6 is more than 3, so 4 sites. With 3, two transactions could each lock a disjoint half and both think they hold the lock.
1481. **A** — The counter is compared first, so 14 is older than 15 or 16 whatever the site ids are. The site id only breaks ties, as between (15, 2) and (15, 3).
1482. **B** — A write needs live weight of at least 6. Losing 2 + 1 leaves 9 - 3 = 6, which is enough. The other failure sets leave 5, 5 and 4, all below the quorum.
1483. **C** — Send πB(s) to S1: 500 × 4 = 2,000 bytes. Send r ⋉ s, the 300 matching rows, to S2: 300 × 60 = 18,000 bytes. Total 20,000 bytes, against 60,000 bytes to ship all of r.
1484. **D** — R + W > 7 needs W ≥ 6, and 2W > 7 needs W ≥ 4, so W = 6. A write needs 6 of 7 replicas, so only 1 can be down. Cheap reads have made writes fragile.
1485. **A** — Reads dominate and slight staleness is allowed, so local replicas give fast, available reads, and lazy propagation keeps the rare writes cheap. Synchronous replication would add cross-continent latency to every write for no business benefit.
1486. **B** — Replication helps when reads are frequent and writes rare. Here writes dominate and come from one site, so every extra copy would multiply write traffic for the sake of one monthly read.
1487. **C** — 3PC is non-blocking under site failures, but its termination rules assume that silence means a crash. In a partition, one side may abort while the other, having seen pre-commit, commits, which breaks atomicity.
1488. **D** — During a partition it stays available (PA). Else, with no partition, it chooses low latency over consistency by not waiting for other replicas (EL).
1489. **A** — Payroll needs one column group for all rows, and branches need another column group for some rows. Splitting by columns, then splitting the branch fragment by rows, puts each access pattern's data local to its site.
1490. **B** — Any site that knows the decision can share it. P1 received commit, so the global decision was commit, and P2 can commit. P2 blocks only if no reachable site knows the outcome.
1491. **C** — Until all votes are ready, the coordinator has not decided commit, so no site has committed. Choosing abort cannot clash with any earlier decision. P3, on recovery, finds no ready record and aborts locally.

1492. **D** — Once T1 released the lock, T2 no longer waited for T1, so there was never a real cycle. Message delays gave the detector an inconsistent snapshot, and it may roll back a transaction for no reason.
1493. **A** — Each operation needs 3 replicas. Only the group of 3 can form a quorum. The group of 2 cannot reach R or W, so it refuses requests. This is the CP behaviour quorums give.
1494. **B** — The problem is replication lag, seen only by the user who just wrote. Sending that user's reads to the primary briefly gives read-your-writes. Everyone else keeps using the fast replicas.
1495. **C** — R + W > N makes reads overlap writes, but 2W = 4 is not greater than N = 4. Writes to {1, 2} and {3, 4} can both succeed with no shared replica to order them, leaving conflicting values.
1496. **D** — When two relations share all their columns, a natural join keeps only tuples present in both, which is their intersection. Disjoint fragments share no tuple, so the result is empty. Horizontal fragments are rebuilt with union.
1497. **A** — A small timestamp makes S2's transactions look old. When they reach items already accessed by newer transactions, the protocol rejects them. A logical clock that jumps forward on seeing a larger timestamp keeps the sites roughly in step.
1498. **B** — Last-writer-wins keeps the value with the later timestamp and throws the other away. For a balance, one deposit or withdrawal disappears. This is the cost of staying available (AP) without a merge rule that suits the data.
1499. **C** — Copy at S1: remote reads 500 + 100 = 600, updates local. Copy at S2: 200 + 100 reads plus 400 updates = 700. Full replication: reads local, each update sent to 2 sites = 800. S3: 200 + 500 + 400 = 1,100. Writes pull the copy to S1.
1500. **C** — A document store keeps each record as a self-describing document (JSON, BSON or XML) and the database understands its internal fields, so you can filter and index on them. A key-value store treats the value as opaque and only looks it up by key.
1501. **D** — A key-value store is the simplest NoSQL model: put(key, value), get(key) and delete(key). The store does not interpret the value, which is why lookups by key are very fast but queries on the value's contents are not supported by the store itself.
1502. **D** — Graph databases store entities as nodes and relationships as edges, so traversing many hops (friends of friends, shortest paths, recommendations) follows edges directly instead of running repeated joins. Large aggregations suit a warehouse and exact-key lookups suit a key-value store.
1503. **A** — BASE is the counterpart of ACID used to describe many NoSQL systems: the system stays basically available, its state may change without new input as replicas sync (soft state), and replicas become consistent eventually rather than immediately.
1504. **C** — In the column-family model (BigTable, HBase and Cassandra style), a row key identifies a row, related columns are grouped into column families, and rows are sparse: a row stores only the columns that have values. This is different from simply storing a relational table column by column.
1505. **B** — Brewer's CAP theorem (proved by Gilbert and Lynch) says that during a partition a system cannot be both consistent and available. Many textbooks phrase it as 'pick two of three'; the more precise reading is that partitions cannot be ruled out, so the real choice is C or A while one lasts.
1506. **C** — CAP consistency is about replicas agreeing: a read sees the latest write (linearisability). ACID consistency is a different idea: a transaction moves the database between states that satisfy its integrity constraints. A non-error response is availability, and surviving lost messages is partition tolerance.
1507. **C** — Scaling out means adding machines to a cluster, which is the design goal of most NoSQL systems. Scaling up (vertical scaling) means buying a bigger single server. Splitting tables to remove redundancy is normalisation, not scaling.
1508. **A** — Sharding (horizontal partitioning) spreads the data so each node holds part of it, which scales storage and writes. Replication keeps copies of the same data, which improves availability and read throughput. Most real systems combine both.
1509. **C** — Schemaless means the database does not enforce a fixed schema at write time (schema-on-read). The structure has not vanished: the code that reads the documents still expects certain fields, so the schema lives in the application. Document stores can still index fields.
1510. **B** — An aggregate is a cluster of related data handled as one unit, such as an order with its line items. Key-value, document and column-family stores are built around such units. Graph databases instead break data into small nodes and relationships, so they are not aggregate-oriented.

1511. **D** — Eventual consistency is a weak guarantee: replicas may be out of date for a while, but once updates stop they all converge. Immediate latest-value reads describe strong consistency, and requiring every replica to acknowledge a write is a strict write quorum.

1512. **B** — A property graph has nodes, directed relationships (edges) and properties, and both nodes and relationships can carry properties. For example, a FRIENDS_WITH relationship can store a since: 2021 property.

1513. **A** — The names sound alike but the ideas differ. Columnar storage is a physical layout for analytics: reading one column across millions of rows is fast. The column-family model is a logical NoSQL data model where rows are found by row key and can have different columns. A common exam trap is treating the two as the same.

1514. **B** — NewSQL describes systems that aim to give relational features (SQL, ACID transactions) together with the horizontal scalability associated with NoSQL. It is a category of systems, not a version of the SQL standard.

1515. **A** — Polyglot persistence means using different data stores for different needs inside one application, each chosen for its access pattern. Sharding splits one dataset across nodes, denormalisation duplicates data to avoid joins, and eventual consistency is a replication guarantee.

1516. **D** — Every access is a whole-value get or put by one key, which is exactly what a key-value store is built for. It gives very fast lookups and scales out easily. No relationships, analytics or joins are needed.

1517. **A** — Mutual-friend suggestions are multi-hop traversals over relationships, which graph databases answer by following edges directly. A key-value or document store would need the application to stitch the links together, and a warehouse is built for aggregation, not traversal.

1518. **C** — Products with varying attributes map naturally to documents: each product document carries just the fields it has, and you can still index fields such as price or brand for filtering. A single wide relational table would be mostly NULLs, and a key-value store could not filter by attribute.

1519. **B** — Wide-column stores handle very high write rates and keep a row's data sorted, so a time-range scan for one sensor is cheap. One unbounded document per sensor is a known anti-pattern because documents have size limits, and a graph adds nothing when there are no relationships to traverse.

1520. **D** — Moving clockwise from 80 there is no node between 80 and 99, so the search wraps round to 0 and the first node reached is A at 10. Node C at 75 is behind the key, not ahead of it.

1521. **B** — Each key goes to the first node at or after its hash: 5 goes to A (10), 12 goes to B (40), 40 goes to B because a key equal to a node's position belongs to that node, and 60 goes to C (75). So B holds 12 and 40.

1522. **C** — D takes over the arc from just after B (40) up to 55. Before D arrived, keys in that arc belonged to C. Key 50 lies in the arc, so it moves from C to D. Key 60 is after 55 and stays on C, and keys 12 and 80 are unaffected. Consistent hashing moves only the keys in the new node's arc.

1523. **A** — Reads are guaranteed to see the latest write when R + W > N, because any read set must then share at least one replica with the write set. Only R = 3, W = 3 gives 6 > 5. The other choices all add up to 5, which is not greater than N.

1524. **B** — Embedding puts the orders inside their parent document, so the collection has one document per customer: 3. Seven would be the count if orders were stored as separate documents, and 10 counts customers and orders as separate documents.

1525. **C** — Column-family rows are sparse, so only present values are stored: 2 + 3 + 1 + 3 = 9 cells. A relational table with the 4 distinct columns would have 4 × 4 = 16 slots, with the missing ones as NULLs.

1526. **D** — Ali's friends are Sara, Omar and Bilal. Their friends are Ali and Hina (from Sara), Ali, Hina and Bilal (from Omar), and Omar and Ali (from Bilal). Removing Ali and his direct friends leaves only Hina, counted once even though two paths reach her. Zain is three hops away.

1527. **B** — 11 mod 3 = 2, because 11 = 3 × 3 + 2. So the key goes to node 2. There is no node 3 in a three-node cluster numbered from 0.

1528. **B** — Range partitioning keeps consecutive key values together, so a date-range query reads one or a few partitions. Hashing deliberately scatters neighbouring values, so a range query would have to ask every node. The trade-off is that range partitioning on time can create a write hot spot on the newest range.

1529. **D** — The overlap rule is R + W > N, so R > 5 - 1 = 4, giving R = 5. With W = 1 the newest value may sit on only one replica, so a read must contact all five to be sure of finding it.

1530. **A** — Only keys where key mod 4 equals key mod 5 stay put. Those are the first four keys in each block of 20 (0-3, 20-23 and so on), so 20 stay and 80 move. Consistent hashing exists to avoid this: adding a fifth node to a ring moves only about 1/5 of the keys.

1531. **C** — Refusing requests on the minority side keeps every accepted withdrawal consistent with the true balance, at the cost of being unavailable to some users. That is the CP choice. A quorum is the mechanism that decides which side may continue, not a way round CAP.

1532. **C** — Accepting writes everywhere during a partition is the AP choice: the service stays available but the replicas diverge. Once the partition heals, the versions must be reconciled, and for carts a union of items is a common merge rule. This is eventual consistency in practice.

1533. **D** — With R + W = 2, which is not greater than 3, the read can land on a replica the write has not reached yet, so a stale read is expected. Setting R = 2 and W = 2 (sum 4 > 3) forces every read set to overlap the write set. Partitioning scheme does not affect replica freshness.

1534. **A** — The requirements (multi-row transactions, joins, ad hoc queries, modest size) are what relational databases do best. NoSQL is a tool for specific scale or data-shape problems, not an automatic upgrade.

1535. **A** — Embedding suits small, bounded data that is always read with its parent. Comments here are unbounded and read in pages, so embedding would create huge documents that are rewritten on every new comment and may hit size limits. Separate documents with a post_id reference, indexed for paging, scale better.

1536. **D** — Monotonically increasing keys such as timestamps all land in the last range, creating a write hot spot. Hashing the sensor ID (or prefixing the key with a hash bucket) spreads writes across nodes while time ordering is kept inside each sensor's partition.

1537. **B** — In a relational schema every hop is another join (or a step of a recursive query) that probes the friendship table, and intermediate results grow quickly. A native graph database keeps each node's relationships directly accessible, so traversal cost depends on the neighbourhood visited rather than the whole table.

1538. **D** — Selling a seat twice is a correctness failure that cannot be fixed later by convergence, so that operation needs strong consistency, such as a conditional write or a transaction. Like counts, last-seen times and view counts can be slightly stale without real harm.

1539. **A** — B owned the arc from just after A (10) up to 40, which holds keys 12 and 40. With B gone, the next node clockwise from those keys is C at 75, so C takes both. Keys 5 and 80 stay on A and key 60 stays on C.

1540. **A** — MongoDB stores records as JSON-like documents (BSON) in collections and can query and index fields inside those documents, which is the defining feature of a document store.

1541. **B** — Redis keeps data in memory and looks values up by key. Its values can be strings, lists, sets, hashes or sorted sets, but access is still by key, which is why it is classed as a key-value store and widely used as a cache.

1542. **C** — Both organise data by row key into column families, and rows may hold different, sparse sets of columns. Their design comes from Google Bigtable (HBase) and Bigtable plus Dynamo (Cassandra).

1543. **D** — Cypher describes graph patterns such as (a)-[:FRIENDS]->(b) and is the query language of the Neo4j graph database. Cassandra uses CQL, and MongoDB uses its own query API.

1544. **A** — BSON (Binary JSON) adds types such as dates and binary data, and it is quicker to scan than plain JSON text. Users still see and write documents in JSON-like syntax.

1545. **B** — Soft state means replicas may differ for a while and the system's state keeps shifting as updates propagate, even when no new writes arrive. It goes hand in hand with eventual consistency.

1546. **C** — The store hashes the shard key, or compares it with key ranges, to place each record on a node. A poor choice can send most traffic to one node.

1547. **D** — Joins across machines are slow, so NoSQL models store what is read together in one place. The price is duplication: shared data copied into many records must be updated in many places.

1548. **A** — Scaling up makes one machine bigger and is limited by the largest machine you can buy. Scaling out (horizontal scaling) adds machines and is the approach most NoSQL systems are built for.

1549. **B** — A collection groups related documents, as a table groups rows. A document is closer to a row, and a field to a column.

1550. **C** — A TTL makes the store delete the key once the time runs out. It is a common way to keep sessions and cached results from growing without limit or going stale.

1551. **D** — A replica set has one primary that accepts writes and records them in its oplog. Secondaries copy the oplog. If the primary fails, the members elect a new primary.

1552. **A** — Each version carries a counter per node. If one clock is less than or equal to the other in every entry, that version is older and can be dropped. Otherwise the writes were concurrent and both versions must be kept and reconciled.

1553. **B** — Hinted handoff keeps writes available during temporary failures. The stand-in node stores the write with a note naming the intended replica and delivers it once that replica is reachable again.

1554. **C** — With gossip, a node picks a few peers every second or so and exchanges what it knows about cluster state. Information spreads quickly through the cluster, and no single node is a point of failure.

1555. **D** — With one position per node, arcs can be very uneven, and a failed node dumps all its keys on one neighbour. Many small arcs per node even out ownership and spread the moved keys across the cluster.

1556. **A** — Appending sequentially is much cheaper than random in-place updates. Memtables are flushed to SSTables, and background compaction merges them later.

1557. **B** — Like a relational index, a secondary index maps field values to the documents that contain them. Without it, a query on city has to scan every document. It is not unique unless declared so.

1558. **C** — Because every node points at its own relationships, one hop costs about the same however large the graph is. A relational join, by contrast, looks up an index or scans for each hop.

1559. **D** — The client sets how many replicas must answer for each read or write, trading latency and availability against freshness per operation. QUORUM reads with QUORUM writes give R + W > N.

1560. **A** — Each stage takes the documents from the previous stage and filters, groups, reshapes or sorts them. It plays the role of SQL's WHERE, GROUP BY and ORDER BY for document data.

1561. **B** — Read-your-writes is a session guarantee: it covers one client's view of its own updates. Linearisability is much stronger and covers every client. Plain eventual consistency does not promise even this.

1562. **C** — SSTables are immutable, so a delete is itself a write: a tombstone. Reads treat the data as gone. Compaction purges it after the grace period, which gives replicas that missed the delete time to learn of it, so deleted data is not resurrected.

1563. **D** — A Bloom filter has no false negatives: a 'no' answer lets the read skip that file. A 'maybe' can be wrong, so the read must still check the file. This saves many disk reads.

1564. **D** — Sorted sets keep members ordered by score in memory, so updates and top-N reads are very fast. A nightly warehouse cannot be live, and a graph store adds nothing for ranking.

1565. **A** — Stations are nodes, and connections are relationships. Shortest-path search over relationships is what graph databases are built for.

1566. **B** — The access pattern is a pure get or set by one key, with automatic expiry. That is exactly what key-value caches such as Redis or Memcached offer.

1567. **C** — Moving clockwise from 30, the first node reached is Z at 50. Y at 24 comes before 30, so it does not own it.

1568. **D** — Referencing keeps each entity as its own document: 5 author documents + 12 book documents = 17. Embedding books in authors would give 5.

1569. **A** — Each byte is stored 3 times: 1,200 × 3 = 3,600 GB. Spread over 6 nodes, that is 600 GB each.

1570. **B** — 50 → first position at or after it is 70 (A). 80 → 95 (B). 97 → nothing until the end, so wrap to 20 (A).

1571. **C** — R owns 31-70 (40 positions). P owns 71-99 and 0-5 (29 + 6 = 35). Q owns 6-30 (25). Unequal gaps between nodes give unequal load, which is why virtual nodes are used.

1572. **D** — The first node at or after 60 is D (80). The next two clockwise, wrapping round, are A (10) and B (30). C at 55 comes before 60, so it does not hold a replica.

1573. **A** — R + W > 5 needs R ≥ 2. A write needs 4 of 5 replicas, so only 1 can be down.

1574. **B** — Before: 5 → A(10), 30 → B, 50 → A(60), 80 → C. Without A, 5 goes to the next node, B at 35, and 50 goes to C at 85. A's load is shared between two nodes, which is the point of virtual nodes.

1575. **C** — A follows B and C. Their follows are D (from both) and E (from C). Neither is followed directly by A, so the answer is D and E. F is three hops away, via D.

1576. **D** — The query needs city 'Lahore' and age greater than 25. Ali (30) and Hina (27) match. Sara is 22. Omar is in Karachi. Zain has no age field, and a comparison like $gt does not match a missing field.

1577. **A** — Hashing user_id scatters new users across all shards, and a lookup by ID still goes to exactly one shard. Creation time sends all new writes to the newest range, and country is low-cardinality and skewed.

1578. **B** — SET makes x 5. INCR adds 1 (6), INCRBY adds 3 (9), DECR subtracts 1 (8). Each command is atomic on the server.

1579. **C** — sensor_id is the partition key, so giving it sends the query to one partition. ts is a clustering column, sorted within that partition, so a range on it is cheap. Queries without the partition key must visit every partition, which CQL refuses unless ALLOW FILTERING is added.

1580. **D** — The name lives in 1 customer document and 50 order documents, so 51 writes. This update cost is the trade-off for reading an order without a second lookup.

1581. **A** — Last-writer-wins keeps the version with the largest timestamp, 110, and discards the others. Keeping siblings is a different strategy, used with vector clocks.

1582. **B** — V1 has a larger A entry and V2 a larger B entry, so neither is ≤ the other in every position: concurrent. V1 ≤ V3, V2 ≤ V4 and V3 ≤ V4 in every entry, so each of those pairs is causally ordered.

1583. **C** — Key 20 is on B and C; key 50 is on C and A; key 80 is on A and B (wrapping). ALL needs every replica, so only key 80, which avoids C, can be read.

1584. **D** — QUORUM for RF 3 needs 2 replicas, but only 1 is up, so it fails. ONE needs a single replica, so it succeeds, at the cost of possible stale reads.

1585. **A** — Many-to-many data with frequently changing details is a case for references. Embedding would copy course details into many students, and every term's change would touch all of them.

1586. **B** — Large scan-and-aggregate queries are a poor match for graph databases, which shine at following relationships from a starting point. Columnar warehouses are built for this kind of query.

1587. **C** — A key-value store sees values as opaque and only looks up by key. Filtering on a field inside the value needs either a full scan or a model that understands fields, such as a document store with a secondary index.

1588. **D** — Schemaless means the database does not enforce structure, so the application must. Validation rules and a one-off migration to a single field name fix the cause, not just the symptom.

1589. **A** — NewSQL systems such as Google Spanner and CockroachDB keep SQL and ACID transactions while sharding and replicating data across nodes. The other options each drop one requirement.

1590. **B** — All traffic for one key lands on its owner and replicas, however many nodes exist. Salting the key spreads one celebrity's data over several partitions, at the cost of reading several buckets.

1591. **C** — Every entry in X's clock is ≤ the matching entry in Y's, so Y already includes X's history. There is no conflict, so Y simply wins. Merging is only needed for concurrent versions.

1592. **D** — R + W > N guarantees overlap only among the N designated replicas. A sloppy quorum may accept writes on stand-in nodes, so a read from the home replicas can miss the write until hinted handoff delivers it.

1593. **A** — Column-family modelling is query-first: each query needs a table whose partition key it can supply. The existing table forces a scan of every user's partition. A table keyed by day turns the dashboard into a single-partition read, at the cost of duplicate writes.

1594. **B** — With balanced ownership, the new server takes about 1/5 of the ring, and only those keys move. A simulation with 200 virtual nodes per server moved about 18.5% of 20,000 keys. Plain mod N hashing would move 80% when going from 4 to 5 servers.

1595. **C** — Each side wrote the full value 11, so LWW keeps one 11 and the other increment vanishes. A counter CRDT keeps each replica's increments and adds them, giving 12.

1596. **D** — w: 1 confirms only the primary's write. If that member fails before replication, the new primary never had the write, and the old primary rolls it back when it rejoins. 'majority' confirms only after most members hold the write, so it survives a failover.

1597. **A** — R + W = 4 > 3, so every read overlaps the latest write. ALL needs every replica, so one outage blocks reads. Writes need only one replica, so they stay available.

1598. **B** — Not every duplicate is an anomaly. The price paid is a fact about the order, fixed at the time of sale. Rewriting it, or reading the live price, would falsify past invoices.

1599. **C** — Overselling comes from concurrent read-then-write without coordination. A consensus-backed conditional write, such as Cassandra's IF clause, makes the decrement succeed only if the stock is still what was read, while other operations stay eventually consistent.

1600. **C** — OLAP (online analytical processing) serves analysts with large aggregating queries over historical, consolidated data. OLTP (online transaction processing) runs many small, concurrent read-write transactions on current data, such as withdrawals and order entry.

1601. **A** — Inmon's classic definition is a subject-oriented, integrated, time-variant and non-volatile collection of data that supports management decisions. Time-variant means data is kept with a time element, so history can be analysed.

1602. **D** — ETL extracts data from source systems, transforms it (cleans, standardises, integrates, aggregates) in a staging area, then loads it into the warehouse. ELT is a variant that loads raw data first and transforms inside the target system, but the acronym asked about here is ETL.

1603. **B** — A star schema has one fact table in the centre holding measures and foreign keys, with each dimension table linked directly to it. Dimensions are kept denormalised, so most queries need only one join per dimension.

1604. **B** — In a snowflake schema, a dimension such as product may be split into product, category and department tables. This saves some space and removes redundancy, but queries need more joins than in a star schema.

1605. **B** — When several fact tables share dimension tables, the design is called a fact constellation or galaxy schema. A snowflake schema is about normalising dimensions, not about having several fact tables.

1606. **A** — Fact rows record business events at a chosen grain, holding measures such as units sold and amount together with foreign keys that point to each dimension. Descriptive attributes live in the dimension tables.

1607. **A** — A data mart is a subset of warehouse-style data focused on one business area or department. It can be dependent (fed from the enterprise warehouse) or independent (fed directly from sources).

1608. **C** — Roll-up aggregates data by climbing a concept hierarchy (city to country) or by removing a dimension. Drill-down goes the other way, to finer detail. Slice fixes one dimension to a single value, and pivot rotates the view.

1609. **B** — A semi-additive measure can be summed over some dimensions but not others. Balances are snapshots, so summing them over time is meaningless; across time you take the last, average or minimum value. Units sold is fully additive, and ratios such as profit margin are non-additive.

1610. **C** — Han and Kamber list KDD as data cleaning, data integration, data selection, data transformation, data mining, pattern evaluation and knowledge presentation. Data mining is the core step that applies algorithms to extract patterns, but it depends on the preparation steps before it.

1611. **B** — Classification is supervised: it learns from labelled examples to predict the class of new records. Clustering is unsupervised: it groups similar records without predefined labels, and the number of clusters depends on the algorithm and its settings.

1612. **B** — Support(X) is the number of transactions containing X divided by the total number of transactions. Confidence is the conditional fraction for a rule, and lift compares confidence with the right-hand side's own support.

1613. **A** — A surrogate key is a meaningless integer assigned by the warehouse. It shields the warehouse from changes or clashes in source keys and lets one natural key have several versions, which is how slowly changing dimensions keep history.

1614. **C** — Type 2 preserves full history by inserting a new row for each change, with effective dates or a current flag. Type 1 overwrites the old value and loses history, Type 3 keeps only a limited previous value in an extra column, and Type 0 never changes the value.

1615. **D** — Relational OLAP (ROLAP) keeps data in relational tables and generates SQL, scaling to large data. Multidimensional OLAP (MOLAP) stores a cube in array structures with pre-computed aggregates, giving fast answers but less scalability. HOLAP combines both.

1616. **A** — Rolling up to region sums every quarter for each region. South has 50 (Q1, Pens) + 70 (Q2, Books) + 20 (Q2, Pens) = 140. 90 is South's Q2 total only and 330 is the grand total over all regions.

1617. **A** — The slice keeps only Q2 rows: North Pens 30, South Books 70, South Pens 20 and East Books 35. Totals by product are Books 70 + 35 = 105 and Pens 30 + 20 = 50. Books 190, Pens 140 are the totals before slicing.

1618. **B** — A dice selects on two or more dimensions. The matching rows are North Q1 Pens 40, North Q1 Books 60 and East Q1 Books 25, giving 125. 175 is all of Q1 including South, and 100 is North alone.

1619. **D** — Moving from a coarser level (year) to a finer level (quarter) of the time hierarchy is drill-down. Roll-up goes from quarter to year, and slice or dice select subsets rather than changing the level of detail.

1620. **C** — The WHERE clause keeps Lahore (store_key 10) facts: Pen 5, Notebook 3, Mouse 2 and Mouse 1. Grouped by category, Stationery = 5 + 3 = 8 and Electronics = 2 + 1 = 3. Electronics 9, Stationery 12 ignores the city filter. Checked in SQLite; the query is standard SQL.

1621. **D** — With n dimensions and no hierarchies, each dimension is either kept or aggregated away, so there are 2^n cuboids. For n = 3 that is 2^3 = 8, from the 3-D base cuboid down to the 0-D apex (grand total).

1622. **C** — Han and Kamber's formula is the product of (Li + 1), where Li is the number of levels in dimension i and the +1 is the virtual 'all' level. So (3 + 1) × (2 + 1) × (1 + 1) = 4 × 3 × 2 = 24. 8 ignores hierarchies, and 12 forgets the product dimension.

1623. **D** — Bread and milk appear together in T1, T2, T5, T7 and T10, which is 5 of 10 transactions, so support = 0.5. 0.7 is the support of bread alone, and 0.625 is 5/8, the fraction of milk transactions that contain bread.

1624. **A** — Confidence = support(butter and bread) / support(butter). Butter appears in 5 transactions (T2, T3, T6, T7, T10), and 4 of them also contain bread, so confidence = 4/5 = 0.8. 0.4 is the rule's support, and 0.571 is the reverse rule bread → butter (4/7).

1625. **D** — Eggs appear in 3 transactions and milk is in all 3, so confidence = 3/3 = 1.0. Milk's support is 8/10 = 0.8, so lift = 1.0 / 0.8 = 1.25. 0.375 is the confidence of the reverse rule milk → eggs (3/8).

1626. **A** — Min-max normalisation is v' = (v - min) / (max - min). So (500 - 200) / (1000 - 200) = 300 / 800 = 0.375. Dividing 500 by 1,000 gives 0.5, which forgets to subtract the minimum.

1627. **C** — The z-score is (v - mean) / standard deviation = (73,600 - 54,000) / 16,000 = 19,600 / 16,000 = 1.225. It says the value is 1.225 standard deviations above the mean.

1628. **D** — The first bin is 4, 8, 15, whose mean is 27 / 3 = 9, so every value in that bin, including 8, becomes 9. Replacing 8 with 4 would be smoothing by bin boundaries, because 4 is the closer boundary.

1629. **C** — A withdrawal is a short read-write transaction on current data that must be atomic and concurrent with many others, which is the definition of OLTP. Data warehouses, OLAP tools and ETL jobs work with copies of this data for analysis later.

1630. **D** — Lift = confidence / support(milk) = 0.714 / 0.8 ≈ 0.89. Milk is in 80% of all baskets but only 71.4% of bread baskets, so the items are slightly negatively correlated. High confidence alone can mislead when the right-hand item is very common.

1631. **A** — The Apriori property says every subset of a frequent itemset must be frequent. Since {butter, eggs} is infrequent, any superset containing it cannot reach the threshold, so it is pruned without scanning the data. This is what keeps candidate generation manageable.

1632. **C** — Frequent items (count of 4 or more) are bread 7, butter 5 and milk 8; eggs (3) is dropped. All three pairs are frequent: bread-butter 4, bread-milk 5 and butter-milk 4. The only 3-item candidate is {bread, butter, milk}, found in T2, T7 and T10, a count of 3, which is below 4.

1633. **B** — Snowflaking trades query speed for less redundancy, because every normalised dimension level adds a join. If storage is not the constraint, flattening dimensions into a star schema cuts joins and simplifies queries.

1634. **A** — A month-end balance is a snapshot. Adding 12 snapshots counts the same money 12 times. Balances can be summed across accounts on one date but over time you take the closing, average or minimum value. That is the definition of a semi-additive measure.

1635. **D** — Mixing OLAP and OLTP workloads on one system makes long analytical scans compete with short transactions for locks, I/O and CPU. A separate warehouse, loaded by ETL, isolates the analytics, keeps history and can be modelled for analysis with a star schema.

1636. **D** — Combining sources needs integration in the transform stage: standardising formats and resolving which records refer to the same real-world entity, then assigning one surrogate key. Without it, one customer becomes two dimension rows and reports are distorted.

1637. **B** — Lahore facts total 5 + 3 + 2 + 1 = 11 units. Because dim_store has two rows with key 10, each Lahore fact row joins twice, so the sum doubles to 22. This is why dimension keys must be unique: a duplicate silently inflates every measure. Checked in SQLite.

1638. **B** — Type 1 overwrites the attribute, so every past fact now points to a dimension row that says Karachi. Type 2 inserts a new row with a new surrogate key, so old facts keep pointing at the Lahore version and new facts at the Karachi version.

1639. **C** — Grouping records by similarity without predefined labels is clustering, an unsupervised task. Classification needs labelled examples, association rules find items that occur together in transactions, and regression predicts a numeric value.

1640. **B** — Pivot (also called rotate) changes the orientation of the cube view, for example swapping rows and columns, without changing the data or its level of detail. Slice fixes one dimension, roll-up aggregates to a coarser level, and drill-across combines facts from different fact tables.

1641. **B** — Dimension tables give context to the facts: textual and descriptive attributes that analysts filter, group and label reports by. The numeric measures and the foreign keys live in the fact table.

1642. **C** — Declaring the grain means stating precisely what a single fact row records, for example one line item on one sales receipt. Kimball makes it the first design decision because every measure and dimension must be true at that grain; mixing grains causes double counting.

1643. **B** — A conformed dimension means the same thing, with the same keys and attributes, in every fact table or mart that uses it. Kimball's bus architecture relies on conformed dimensions so that results from different marts can be compared and combined.

1644. **C** — A degenerate dimension is a dimension key, typically a transaction control number such as an invoice or order number, that sits in the fact table with no matching dimension table. It is still useful for grouping all lines of one invoice.

1645. **B** — A factless fact table records that an event happened (or a coverage relationship holds) using only dimension keys. You analyse it by counting rows, for example attendance per lecture.

1646. **A** — Inmon's corporate information factory is top-down: an integrated, normalised enterprise data warehouse comes first and dependent marts are built from it. Kimball's approach is bottom-up: business-process marts in star schemas, tied together by conformed dimensions through the bus architecture.

1647. **C** — Extracted source data lands in the staging area so that cleaning, de-duplication, key lookup and other transformations can run without touching the source systems or exposing half-processed data to users.

1648. **B** — Metadata describes the warehouse itself: schemas, where each column came from, what transformations were applied, and when it was last loaded. Users and ETL tools rely on it to understand and maintain the data.

1649. **D** — Han and Kamber call the 0-D cuboid, which aggregates over every dimension into one cell (the grand total), the apex cuboid. The n-D cuboid at the finest level is the base cuboid.

1650. **B** — Han and Kamber list three choices: no materialisation (compute everything on the fly), full materialisation (store every cuboid) and partial materialisation (store a selected subset). Partial is the usual compromise between storage and query speed.

1651. **D** — Hybrid OLAP combines ROLAP's scalability for large detailed data with MOLAP's speed for pre-computed summaries: detail stays relational, aggregates go into a multidimensional array store.

1652. **A** — A bitmap index stores one bit vector per distinct value. For low-cardinality attributes the vectors are small, and AND/OR of bit vectors answers multi-condition queries quickly. They suit read-mostly warehouses, not update-heavy OLTP.

1653. **A** — Type 3 adds a 'previous value' column, so the row holds both the current and the prior value, giving limited history. Type 1 overwrites, Type 2 adds a new row per version, and Type 0 never changes the original value.

1654. **A** — Outlier or anomaly detection finds objects that deviate strongly from the rest of the data, which is the basis of most fraud detection. Association rules find items that occur together, and regression predicts a numeric value.

1655. **C** — Both are supervised prediction tasks. Classification outputs a class such as 'approve' or 'reject'; regression outputs a number such as a house price or next month's sales.

1656. **A** — Support can only fall or stay the same as items are added, so if an itemset is frequent, each of its subsets is at least as frequent. The equivalent anti-monotone form is that any superset of an infrequent itemset is infrequent, which Apriori uses for pruning.

1657. **D** — FP-growth compresses the transactions into an FP-tree (one scan to count items, one to build the tree) and then mines it recursively by divide and conquer, avoiding Apriori's costly candidate generation and repeated scans.

1658. **B** — Following Han and Kamber, X is closed if no proper superset has the same support count, and maximal if no proper superset is frequent. Every maximal itemset is closed, but not every closed itemset is maximal. Closed itemsets keep full support information; maximal ones are a smaller summary.

1659. **A** — k-means is a partitioning method: you set k, assign each point to the nearest centroid, recompute centroids as means and repeat. It converges to a local optimum that depends on the starting centroids. Using a real point as the representative is k-medoids.

1660. **A** — ID3 picks the attribute that gives the largest information gain, the drop in entropy after the split. C4.5 uses gain ratio to correct ID3's bias towards many-valued attributes, and CART uses the Gini index.

1661. **C** — An overfitted model memorises the training set, including its noise. Training accuracy is high but test accuracy is low. Tree pruning, more data and cross-validation are standard remedies. A model too simple to fit the data is underfitting.

1662. **C** — PCA projects the data onto a smaller number of new axes that keep most of the variance, reducing the number of attributes. Han and Kamber list it under data reduction, alongside attribute subset selection and numerosity reduction.

1663. **B** — With no hierarchies each dimension is either present or aggregated away, so there are 2^n cuboids. For n = 4 that is 2^4 = 16, from the 4-D base cuboid down to the 0-D apex.

1664. **A** — The formula is the product of (L_i + 1), where the +1 is the virtual 'all' level. So (4 + 1) x (3 + 1) x (2 + 1) = 5 x 4 x 3 = 60. 24 forgets the 'all' level (4 x 3 x 2) and 8 is 2^3, which ignores hierarchies.

1665. **C** — A 2-D cuboid keeps exactly 2 of the 4 dimensions, and the number of ways to choose them is C(4, 2) = 6. The layers of the lattice have 1, 4, 6, 4 and 1 cuboids, which sum to 16.

1666. **D** — lift(A → B) = confidence(A → B) / support(B) = 0.6 / 0.4 = 1.5, so A makes B 1.5 times more likely than its base rate. 0.3 is the rule's support (0.6 x 0.5), and 1.2 wrongly divides by support(A).

1667. **D** — The CASE expressions put Q1 amounts in one column and Q2 amounts in another. South has Q1 = 30 and Q2 = 45 + 10 = 55. Taking only the first Q2 row gives 45, and swapping the columns gives 55, 30. This is a pivot written in SQL.

1668. **D** — Karachi covers store keys 2 and 3. In 2024 only (101, 2, 7) matches, giving 7. In 2025 the matching rows are 5, 8, 6 and 9, totalling 28. 17 and 32 include Lahore's units, and 14 counts only store 2.

1669. **B** — v' = (v - min) / (max - min) x (new_max - new_min) + new_min = (700 - 200) / 800 x 2 + (-1) = 0.625 x 2 - 1 = 0.25. 0.625 is the result for the [0, 1] range.

1670. **B** — Decimal scaling divides by 10^j, where j is the smallest integer that makes every |v'| < 1. The largest absolute value is 986, so j = 3 and -452 / 1000 = -0.452.

1671. **C** — The second bin is (13, 15, 35), whose boundaries are 13 and 35. Each value is replaced by the closer boundary: |15 - 13| = 2 and |35 - 15| = 20, so 15 becomes 13. 21 is the bin mean, which would be smoothing by bin means.

1672. **C** — Equal-width binning uses width (90 - 10) / 4 = 20, giving [10, 30), [30, 50), [50, 70) and [70, 90]. 63 lies in [50, 70), the third bin.

1673. **B** — Info(D) = -(9/14) log2(9/14) - (5/14) log2(5/14) = 0.410 + 0.530 = 0.940 bits. 1.000 would need an even 7/7 split, 0.459 is the Gini index of the same set, and 0.643 is just 9/14.

1674. **A** — Gini(D) = 1 - (9/14)^2 - (5/14)^2 = 1 - 0.413 - 0.128 = 0.459. 0.541 is the sum of squared proportions (forgetting to subtract from 1), 0.940 is entropy and 0.357 is 5/14.

1675. **A** — Euclidean distance = sqrt((5 - 2)^2 + (7 - 3)^2) = sqrt(9 + 16) = sqrt(25) = 5. 7 is the Manhattan distance (3 + 4) and 25 forgets the square root.

1676. **D** — Manhattan distance sums the absolute differences: |4 - 1| + |0 - 2| + |5 - 3| = 3 + 2 + 2 = 7. 4.12 is the Euclidean distance (sqrt 17) and 17 is the sum of squares.

1677. **D** — Each point joins its nearer centroid: 2, 3, 4 go to 3; 10 (distance 7 vs 2), 12 and 20 go to 12. The new means are (2 + 3 + 4) / 3 = 3 and (10 + 12 + 20) / 3 = 14. 11 leaves out 20, and 4.75 wrongly puts 10 in the first cluster.

1678. **C** — Each candidate 2-itemset is an unordered pair of distinct frequent items, so there are C(5, 2) = 5 x 4 / 2 = 10. 25 counts ordered pairs including an item with itself, and 20 counts ordered pairs.

1679. **C** — Each non-empty proper subset can be the left side, with the rest on the right. A 3-itemset has 2^3 - 2 = 6 such subsets: three rules with one item on the left and three with two. 8 counts all subsets, including the empty set and the full set.

1680. **D** — The join merges 2-itemsets that share their first item: {A,B} + {A,C} gives {A,B,C}, and {B,C} + {B,D} gives {B,C,D}. Pruning removes {B,C,D} because its subset {C,D} is not frequent. {A,B,C} survives because {A,B}, {A,C} and {B,C} are all frequent.

1681. **A** — Precision = TP / (TP + FP) = 40 / 45 = 0.889. Recall = TP / (TP + FN) = 40 / 50 = 0.800, and accuracy = (TP + TN) / 100 = 0.850.

1682. **D** — Dice selects on two or more dimensions at once (here city, category and year) to give a smaller sub-cube. Slice fixes a single value of one dimension, pivot only rotates the view and roll-up aggregates to a coarser level.

1683. **C** — The target is a continuous number learned from past sales with known prices, which is regression, a supervised prediction task. Clustering has no target, association rules find co-occurring items, and outlier detection flags unusual records.

1684. **D** — Header totals already equal the sum of their lines, so storing both grains in one table and summing counts every rupee twice. Kimball's rule is one grain per fact table; keep order lines in one fact and, if needed, headers in another.

1685. **D** — Support measures frequency, so items bought by few customers never reach 5% however strongly they are associated. Lowering the threshold, or using multiple minimum supports per item, lets such rules appear (at the cost of more candidates).

1686. **D** — Predicting the majority class gives 9,900 / 10,000 = 99% accuracy while catching none of the 100 frauds, so recall for fraud is 0. With imbalanced classes, use recall, precision, F1 or a confusion matrix instead of accuracy alone.

1687. **A** — A large train/test gap is the signature of overfitting: the tree memorised noise. Pre-pruning or post-pruning simplifies it and usually raises test accuracy.

1688. **B** — Distance-based methods weigh attributes by their numeric spread. A difference of 1,000,000 rupees swamps a 30-year age gap, so normalising puts the attributes on comparable scales.

1689. **B** — A mean is pulled strongly by extreme values, so one outlier drags a centroid away. k-medoids uses a real, central object as the representative and minimises absolute rather than squared distances, so it resists outliers. Removing or treating the outlier first also helps.

1690. **A** — Ratios cannot be summed or simply averaged. Profit is 100 + 30 = 130 on sales of 1,100, so the margin is 11.8%. Store the additive parts (profit and sales) as measures and compute the ratio after aggregation.

1691. **A** — Type 3 keeps a current-value column and a previous-value column in the same row, which is exactly the 'before and after one change' view. Type 2 keeps full history but in separate rows, so a side-by-side comparison needs extra work.

1692. **B** — Incremental extraction, using timestamps, logs or change data capture, moves only the changed 1% instead of everything, cutting the load window dramatically while keeping the warehouse current.

1693. **A** — ROLAP runs over relational star schemas and handles very large, detailed and changing data well. MOLAP is fastest for compact, pre-aggregated cubes but its arrays grow sparse and large with detailed, high-dimensional data.

1694. **B** — Independent 'stovepipe' marts drift apart in keys and definitions. Conforming the customer dimension and agreeing the business rules lets every mart report the same customers the same way, so their numbers can be compared and combined.

1695. **D** — With hierarchies the count is the product of (levels + 1) = 5^10 = 9,765,625 cuboids. Storing them all costs far too much space and load time, which is why warehouses materialise a selected subset.

1696. **C** — Books = 30 + 20 = 50 and Toys = 25 + 10 = 35, total 85. The row (9, 15) has no product 9 in dim_product, so the inner join discards it. Enforcing referential integrity in ETL, typically by pointing such rows to an 'Unknown' member, keeps totals consistent.

1697. **D** — Association rules measure how often items appear together. Lift above 1 shows positive correlation, but no association measure proves cause and effect; a controlled test such as an A/B promotion is needed.

1698. **C** — When historical records carry the class label and the goal is to predict that label for new records, the task is classification (supervised learning). Clustering ignores the labels and would not directly predict churn.

1699. **A** — An INNER JOIN keeps a pair of rows only when the join condition is TRUE for both. Outer joins also keep unmatched rows padded with NULLs, and a CROSS JOIN pairs every row with every row with no condition at all.

1700. **D** — A LEFT (outer) JOIN preserves every row of the left table. When no right row matches, the right side is padded with NULLs, not zeros or empty strings. Dropping the row is what an INNER JOIN would do.

1701. **D** — Under SQL's three-valued logic, any comparison with NULL, including NULL = NULL, gives UNKNOWN. A join condition must be TRUE to keep a pair, so NULL keys never match each other. The rows are not removed beforehand; they simply find no partner.

1702. **B** — A RIGHT JOIN preserves every row of the right table, B. Swapping the table order and using LEFT JOIN preserves B in the same way, so B LEFT JOIN A gives identical rows. A LEFT JOIN B would preserve A instead.

1703. **C** — A self join lists the same table twice in FROM. Aliases (e for the employee, m for the manager) let the ON clause tell the two copies apart, as in e.mgr = m.eid. No physical copy of the table is needed.

1704. **B** — This is an anti-join. The LEFT JOIN pads unmatched A rows with NULLs in B's columns, and WHERE B.k IS NULL keeps exactly those rows. Matched rows have a non-NULL B.k (when B.k is the join column), so they are removed.

1705. **B** — Enrolments 10 and 11 match course 1 (DB) and enrolment 12 matches course 2 (OS). Enrolment 13 points to course 4, which does not exist, so it is dropped. AI has no enrolments and is also absent. That gives 3 rows.

1706. **D** — Ali matches three orders (3 rows), Omar matches one (1 row) and Sara matches none but is kept with a NULL oid (1 row): 3 + 1 + 1 = 5. Order 104 belongs to a missing customer and is not preserved, because only the left table is.

1707. **A** — Matches: the two 3s in a each pair with the single 3 in b, giving 2 rows. Unmatched a rows: 1 and 2 (2 rows). Unmatched b rows: 4 and 5 (2 rows). Total 2 + 2 + 2 = 6.

1708. **A** — Each row pairs an employee e with their manager m. Faraz and Kiran report to Nadia (2), and Imran and Saba report to Kiran (2). Faraz manages nobody, so he never appears as m. Nadia has no manager, and the inner join drops her as an employee row rather than adding a NULL group.

1709. **B** — An inner join on a repeated key multiplies counts per value. Key 1 appears 2 times in r and 3 times in s: 2 × 3 = 6 rows. Key 2 appears 3 times in r and once in s: 3 rows. Keys 7 and 8 have no partner. Total 6 + 3 = 9.

1710. **D** — A RIGHT JOIN keeps every dept row. HR matches Ali and IT matches Omar and Sara. Finance (did 3) has no employee, so it appears once with e.name NULL. Hina's did 5 has no department, and as a left-side row she is not preserved by a RIGHT JOIN.

1711. **A** — A CROSS JOIN is the Cartesian product: every row of the first table pairs with every row of the second. 4 sizes × 3 colours = 12 rows. Adding the counts (7) is a common slip.

1712. **C** — Count the pairs with a < b. b = 2: a = 1 (1 pair). b = 3: a = 1, 2 (2 pairs). b = 5: a = 1, 2, 3, 4 (4 pairs). Total 1 + 2 + 4 = 7. A CROSS JOIN filtered by WHERE behaves like a theta join.

1713. **A** — In Q1 the filter is part of the join, so it only decides which orders match: Ali matches 200, while Sara and Omar keep their rows with NULL amounts (3 rows). In Q2 the LEFT JOIN first gives 4 rows, then WHERE removes every row whose amount is not above 100, including the NULL-padded ones. Only Ali, 200 survives.

1714. **C** — The LEFT JOIN keeps Omar as one row with o.oid NULL. COUNT(*) counts rows, not values, so Omar gets 1. COUNT(o.oid) skips NULLs and would correctly return 0. This is a classic outer-join-plus-aggregate bug.

1715. **D** — Omar's did 3 has no department and Sara's NULL did cannot match anything (NULL = 1 is UNKNOWN). Both rows are NULL-padded, so d.did IS NULL keeps them. Ali and Hina matched real departments.

1716. **B** — Joining two child tables on the same key creates 2 × 3 = 6 rows. Each order amount appears 3 times (300 × 3 = 900) and each payment appears twice (250 × 2 = 500). The fix is to aggregate each table in a subquery first, then join the totals.

1717. **C** — NATURAL JOIN equates every column with the same name, here both did and name. No employee's name equals a department name, so no row satisfies both equalities and the result is empty. Use JOIN ... USING (did) or an explicit ON to join on did only.

1718. **C** — The LEFT JOIN keeps all three students, but the following INNER JOIN needs e.cid = c.cid to be TRUE. Sara's course 200 does not exist and Omar's e.cid is NULL, so both are dropped. Only Ali (DB) survives. Making the second join a LEFT JOIN too would keep everyone.

1719. **A** — A candidate key is minimal. If E appears only on right-hand sides, it can always be derived from other attributes, so any set containing E stays a superkey after removing E. Such a set is never minimal, so E belongs to no candidate key.

1720. **C** — If $X^+ = R$, X functionally determines every attribute, which is the definition of a superkey. It is a candidate key only if no proper subset of X also has a closure equal to R.

1721. **A** — 3NF requires, for every non-trivial $X \to A$, that X is a superkey or A is prime. The prime-attribute escape clause is exactly what BCNF removes, which is why a relation can be in 3NF but not in BCNF.

1722. **B** — Splitting on a violating FD $X \to Y$ into $(X \cup Y)$ and $(R - Y)$ is always lossless, because the common part X determines $X \cup Y$. It can split a dependency across tables, though. The 3NF synthesis algorithm (Elmasri-Navathe, Silberschatz) guarantees both properties.

1723. **D** — 2NF forbids a non-prime attribute depending on only part of a candidate key, for example CourseTitle depending on CourseID alone when the key is (StudentID, CourseID). A dependency between non-prime attributes is transitive, which is 3NF's concern.

1724. **D** — Nothing determines A, so the only way a set's closure can contain A is if A is in the set already. Every superkey, and so every candidate key, must include A. This is the first shortcut in key-finding.

1725. **B** — Start with {A}. A → C adds C. CD → E needs D as well, which A cannot reach, so nothing more fires. A^+ = {A, C}.

1726. **B** — Start with {B, C}. C → D adds D. BD → E adds E. E → F adds F. Nothing produces A, so $(BC)^+$ = {B, C, D, E, F} and BC is not a superkey. $(AC)^+$ is the whole relation.

1727. **D** — A is never on a right-hand side, so it is in every key, and E is never on a left-hand side, so it is in none. A^+ = {A}. Try A plus one attribute: $(AB)^+$, $(AC)^+$ and $(AD)^+$ each reach all five attributes. So AB, AC and AD are all candidate keys.

1728. **C** — The FDs form a cycle, so each single attribute reaches all the others: $A^+ = B^+ = C^+ = D^+$ = {A, B, C, D}. Each of A, B, C and D is a candidate key, giving 4. Every left-hand side is a key, so R is also in BCNF.

1729. **D** — The only key is AB. Every non-prime attribute depends on the whole key, so there is no partial dependency and R is in 2NF. But C → D and D → EF have non-key left sides and non-prime right sides, which are transitive dependencies, so R is not in 3NF.

1730. **B** — A^+: A → BC, B → D, CD → E, so A is a key, and E → A makes E a key too. $(BC)^+$: B → D, CD → E, E → A, so BC is a key. $(CD)^+$: CD → E, E → A, A → BC, so CD is a key. Every other pair either contains A or E or fails to reach A (for example BD^+ = {B, D}).

1731. **A** — A binary decomposition is lossless when the common attributes determine all of R1 or all of R2. For (A, B) and (A, C, D) the common attribute is A, and A → B covers R1. (A, B) and (C, D) share nothing, and in the other options the shared attribute (B, or nothing) determines neither side.

1732. **C** — In AB → C, B is enough on its own (B → C), so AB → C reduces to B → C and duplicates it. In AC → D, C is extraneous because A^+ already contains C (A → B → C), giving A → D. No remaining FD is redundant: {A → B, B → C, A → D}.

1733. **C** — The candidate keys are AB, AC and AD, so A, B, C and D are prime and only E is non-prime. D → E makes E depend on D, a proper subset of the key AD. That is a partial dependency, so R is not even in 2NF.

1734. **A** — The common attribute D determines A, so D^+ covers R1 and the join is lossless. R1 has key D and R2 has key BD (BD → A → AB → C), so both are in BCNF. But no single part contains A, B, C and D together, so AB → CD can no longer be checked in one table.

1735. **C** — D appears in no FD, so it is in every key: the candidate keys are ABD and BCD, making A, B, C and D all prime. C → A violates BCNF (C^+ = {A, C}), but its right side A is prime, so 3NF holds.

1736. **B** — Saad appears with room R1 and with room R3, so teacher → room fails. The other three hold in this instance. Note that sample data can only disprove an FD; it can never prove that one holds for all future rows.

1737. **D** — Supersets of A: $2^3 = 8$. Supersets of BC: $2^2 = 4$. Sets containing both A and BC are counted twice: $2^1 = 2$. By inclusion-exclusion, 8 + 4 − 2 = 10.

1738. **A** — Test each FD with a closure. D^+ = {D}, which lacks A, so D → A does not follow. A^+ = {A, B, C, D} gives A → D and AC → BD, and B^+ = {B, C, D} gives B → CD.

1739. **D** — The transaction finished only half of its work and that half survived. Atomicity requires all of a transaction's effects or none of them, so the seat update should have been undone when the payment insert failed.

1740. **B** — Each transaction would have been correct if run alone. The double booking comes only from their interleaving, which isolation (enforced by concurrency control, such as locking) should prevent. With proper isolation the second clerk would see the seat as taken.

1741. **D** — Consistency means a transaction takes the database from one valid state to another, where 'valid' includes declared integrity constraints such as CHECK, keys and foreign keys. Rejecting the violating statement keeps the database in a valid state.

1742. **B** — The recovery manager undoes the effects of incomplete transactions (atomicity) and redoes committed work after a crash (durability), both from the log on stable storage. The concurrency-control manager, using locks or timestamps, provides isolation. Consistency is shared between integrity constraints and correct transaction code.

1743. **A** — The standard defines phantoms as allowed at READ UNCOMMITTED, READ COMMITTED and REPEATABLE READ; only SERIALIZABLE forbids them. Some products go further than the standard: MySQL InnoDB's REPEATABLE READ, for instance, blocks most phantoms with next-key locks.

1744. **C** — In ACID, consistency is about valid states: constraints and business rules are preserved by each transaction. In CAP (distributed systems), consistency means linearisability: any read on any node sees the most recent write. A single-node DBMS can be ACID-consistent without CAP ever arising.

1745. **C** — ROLLBACK TO s1 undoes only the work done after the savepoint (the 400 transfer). The 100 debit made before it stays, and the later 100 credit is added. COMMIT then makes 900 and 600 permanent. A savepoint gives partial rollback inside one transaction.

1746. **A** — A single SQL statement is atomic. When the CHECK fails on account 2, the whole UPDATE is undone, including the rows it had already changed, so no balance moves. SQLite's default ABORT behaviour, and every major DBMS, works this way. Verified in SQLite.

1747. **D** — T1's abort restores X to 500, but T2 has already committed a value computed from the dirty 800, so Y = 80 instead of 50. This is a dirty read, and because T2 committed first the schedule is also unrecoverable. READ COMMITTED or stricter prevents it.

1748. **C** — T2 reads A = 2000 before the transfer and B = 1500 after it, so it reports 3500. The true total is 3000 at every committed point. This inconsistent analysis (incorrect summary) happens because T2 was not isolated from T1.

1749. **A** — T1 never committed, so atomicity requires removing all its effects. Recovery undoes T1's updates using the old values in the log, in reverse order: B back to 2000, then A back to 1000. It does not matter that B's new value never reached disk.

1750. **A** — READ COMMITTED only guarantees that you never see uncommitted data. T2's 70 is committed by the second read, so T1 sees it. Getting 50 then 70 within one transaction is a non-repeatable read, which REPEATABLE READ prevents.

1751. **B** — T1's commit record is on disk, so durability requires redoing it: A = 150 even though the page never got there. T2 has no commit record, so atomicity requires undoing it: B goes back to 300. Durability comes from the log, not from forcing data pages.

1752. **D** — Only a matching debit and credit leave the total unchanged: 900 + 800 + 300 = 2000. The other three totals are 1900, 2600 and 1700. The DBMS cannot spot these errors unless the rule is declared, so consistency is partly the programmer's job.

1753. **B** — Serialisable means equivalent to some serial order. T1 then T2 gives 100 × 2 + 50 = 250; T2 then T1 gives (100 + 50) × 2 = 300. Interleavings such as r1 r2 w1 w2 give 150 (T1's write is lost) and r1 r2 w2 w1 give 200, so they are not serialisable.

1754. **A** — In autocommit mode each statement commits on its own, so the DBMS correctly keeps the committed debit. Atomicity holds per statement, but the business transfer was never declared as one transaction. Putting both statements inside one explicit transaction makes them all-or-nothing.

1755. **C** — Each lost transaction disappears entirely, so atomicity holds and the database is still a consistent earlier state. What is lost is the promise that an acknowledged commit survives a crash. PostgreSQL's synchronous_commit = off makes exactly this trade.

1756. **C** — SQLite undoes only the failing statement and leaves the transaction open, so COMMIT keeps rows 1 and 3 (verified). MySQL InnoDB behaves the same for a duplicate key. PostgreSQL differs: after an error it rejects every later statement and turns COMMIT into a rollback, so the table would be empty there.

1757. **D** — Both statements ran and committed, so atomicity held, and with no concurrent work isolation is irrelevant. The transaction itself turned a valid state into an invalid one. Silberschatz notes that consistency of an individual transaction is the application programmer's responsibility.

1758. **B** — Both sessions computed from the stale 1000, so the second write (1500) overwrote the first (800): a lost update. An in-place UPDATE locks the row and reads the current value, so the second change starts from 800 and gives 1300. SELECT ... FOR UPDATE or SERIALIZABLE would also work.

1759. **A** — GROUP BY treats all NULLs in the grouping column as one group, even though NULL = NULL is not TRUE in a WHERE clause. The groups are IT, HR, Ops and NULL, so 4 rows. B and D form the NULL group with a count of 2.

1760. **A** — MAX over zero rows returns NULL. Even a scalar subquery with no aggregate that returns no rows gives NULL rather than an error. An error occurs only when a scalar subquery returns more than one row.

1761. **C** — COUNT(column) counts non-NULL values, so it returns 0. SUM, AVG, MIN and MAX over no non-NULL values return NULL, not 0. Use COALESCE(SUM(x), 0) if you need a zero. COUNT(*) would give 2.

1762. **D** — An aggregate query without GROUP BY always returns exactly one row, even over an empty table. COUNT(*) is 0 and MAX(x) is NULL. With GROUP BY x, an empty table would give no rows at all.

1763. **D** — An integer in ORDER BY refers to a position in the SELECT list, not the table. Column 2 of the result is age, so descending order puts Sara (25) first, then Ali (22), then Omar (21).

1764. **B** — With no GROUP BY, the whole table is one group. SUM(x) = 30 fails HAVING SUM(x) > 100, so that single group is removed and the result is empty. Standard SQL allows HAVING without GROUP BY. With > 20 you would get one row, 30.

1765. **D** — COUNT(*) counts rows, including Omar's order with a NULL amount, so ISB has 2. KHI has orders 2, 5 and 6, and LHR has 1, 3 and 7. ORDER BY city sorts them alphabetically.

1766. **B** — WHERE drops the rows with scores below 50 (S1-OS 45 and S3-DB 39) before grouping. S1 is left with one row and fails COUNT(*) >= 2. S2 averages (88 + 91) / 2 = 89.5 and S3 averages (58 + 64) / 2 = 61.0.

1767. **C** — AVG ignores the NULL: 1450 / 7 ≈ 207.14, not 1450 / 8. The amounts above 207.14 are 300 (oid 1), 400 (oid 5) and 250 (oid 6). Dividing by 8 would give 181.25 and wrongly include 200 (oid 3).

1768. **C** — For each employee, the subquery finds the top salary in that employee's own department. IT's top is 120 (Sara), HR's is 70 (shared by Omar and Hina) and Ops's is 55 (Zain). Ties are all returned, giving 4 names.

1769. **D** — The subquery totals each city: LHR 600, KHI 800 and ISB 50 (SUM skips the NULL). Only KHI passes SUM(amount) > 650, so the outer query keeps customers with a KHI order: Hina and Sara. DISTINCT removes Sara's second row.

1770. **B** — The inner query finds the top salary, 80. The outer query keeps salaries strictly below 80 (50, 65, 40) and takes their maximum, 65. This is the standard 'second highest distinct salary' pattern; the duplicate 80s do not matter.

1771. **B** — Sorted by pts descending, then name: E 90, B 85, D 85, F 75, A 70, C 60. OFFSET 2 skips E and B, and LIMIT 2 takes the next two: D and F. The tie between B and D is broken by name.

1772. **B** — Present counts: S1 2 of 3, S2 1 of 4 and S3 1 of 1. HAVING keeps students who were present in more than half their records: S1 (4 > 3) and S3 (2 > 1). S2 fails (2 > 4 is false).

1773. **C** — AVG(x) divides by the number of non-NULL values: 60 / 3 = 20. COUNT(*) counts all 5 rows, so the second expression is 60 / 5 = 12. They agree only when the column has no NULLs, or if you use COUNT(x).

1774. **D** — Q1 filters rows first, so it counts only orders over 100. Ali keeps one (300), Omar one and Sara two, so only Sara has more than one. Q2 counts all orders and then checks for at least one big order, which Ali (2 orders, max 300) and Sara satisfy. Hina has 2 orders but none over 100.

1775. **A** — For each row, the subquery counts how many scores are strictly higher. E has 0 and A has 1, so both qualify (< 2). B and C each have 2 higher scores (95 and 90), so they fail. Here the result happens to be exactly 2 rows; with a tie at second place, both tied rows would appear.

1776. **A** — COUNT(*) counts all 6 rows. COUNT(DISTINCT a) ignores NULL, leaving {1, 2}: 2. SELECT DISTINCT treats NULLs as equal to each other, so the distinct pairs are (1,1), (1,2), (2,NULL) and (NULL,NULL): 4.

1777. **A** — The inner query totals each rep: A 400 and B 200 in N; C 600 and D 100 in S. The outer AVG averages those rep totals: N (400 + 200) / 2 = 300 and S (600 + 100) / 2 = 350. A plain AVG(amt) per region would average individual sales instead (200 and 233.33).

1778. **C** — Counts are IT 3, HR 1 and Ops 1. Ascending order puts the two 1s first, and LIMIT 1 cuts the result to one row, so a tied department silently disappears. Use HAVING COUNT(*) = (SELECT MIN(c) FROM (SELECT COUNT(*) AS c FROM emp GROUP BY dept)) to return every tied group.

1779. **B** — Projection keeps only the listed attributes, so it lowers the degree, and under set semantics it also removes duplicate tuples. Selection keeps all attributes and removes tuples, union keeps the degree, and a product adds the two degrees.

1780. **B** — R − S keeps tuples of R not in S, while S − R keeps tuples of S not in R, so they differ in general. Union and intersection are commutative, and so is natural join once you ignore the order of the columns.

1781. **C** — Division removes the divisor's attributes from the dividend: (sid, cid) − (cid) = (sid). Each sid in the result is paired in R with every cid in S, which is why division answers 'for all' queries.

1782. **D** — With every attribute in common, a natural join pairs tuples that agree on all attributes, which means identical tuples. The result is the tuples present in both relations: R ∩ S. If they share no attributes, the natural join becomes R × S instead.

1783. **D** — Selection keeps the tuples that satisfy p, so it can keep none of them (0) or all of them (12), but never adds any. 144 = 12 × 12 would be a self-product, not a selection.

1784. **A** — Projection distributes over union: a value appears in the projection of the union exactly when it appears in either projection. It does not distribute over difference or intersection. With R = {(1, x)} and S = {(1, y)}, π_A(R − S) = {1} but π_A(R) − π_A(S) = {}.

1785. **B** — Both conditions must hold. The IT employees are Asad (70), Cyrus (90) and Ehsan (90); only Cyrus and Ehsan have sal ≥ 80. So the result has 2 tuples.

1786. **C** — The sal values are 70, 50, 90, 50 and 90. Relational algebra uses set semantics, so duplicates are removed: {50, 70, 90}, 3 tuples. SQL's SELECT sal without DISTINCT would return 5 rows.

1787. **A** — The join is on eid. Asad has 2 projects, Bina 1, Cyrus 3, Ehsan 1 and Dua none: 2 + 1 + 3 + 1 = 7. Every Works tuple refers to an existing employee, so the join has as many tuples as Works.

1788. **B** — For each employee, count those who earn strictly more. Bina (50) and Dua (50) each have 3: Asad, Cyrus and Ehsan. Asad (70) has 2: Cyrus and Ehsan. Cyrus and Ehsan (90) have none. Total: 3 + 3 + 2 = 8.

1789. **C** — Works ÷ π_pid(Proj) returns the eids paired with every project: P1, P2 and P3. Only E3 works on all three, since E1 lacks P3. Joining with Emp and projecting the name gives {Cyrus}.

1790. **A** — π_eid(Emp) = {E1, E2, E3, E4, E5} and π_eid(Works) = {E1, E2, E3, E5}. Subtracting leaves {E4}: Dua, the employee who works on no project.

1791. **D** — The natural join gives 7 tuples. The left outer join also keeps each Emp tuple with no match, padded with NULL. Only Dua (E4) has no project, so there are 7 + 1 = 8 tuples, one of them with a NULL pid.

1792. **D** — The projects with budget > 200 are P2 (250) and P3 (400). P2 is worked on by E1, E3 and E5, and P3 by E3. Their names are Asad, Cyrus and Ehsan. Bina works only on P1 (100).

1793. **D** — Each Works tuple holds one pid, so 'both' needs two separate sets combined by intersection: {E1, E2, E3} ∩ {E1, E3, E5} = {E1, E3}. A single tuple can never have pid = 'P1' and pid = 'P2' at the same time.

1794. **C** — π_eid(Works) has 4 eids, so the product has 4 × 3 = 12 'ideal' pairs. Removing the 7 pairs that exist leaves 5 missing pairs: (E1, P3), (E2, P2), (E2, P3), (E5, P1) and (E5, P3). Their eids {E1, E2, E5} are removed from {E1, E2, E3, E5}, leaving E3.

1795. **C** — 'Every grade is A' means 'no grade that is not A'. Remove the students with at least one non-A grade (S2 has a B and S4 has a C) from all students: {S1, S2, S3, S4} − {S2, S4} = {S1, S3}. Selecting grade = 'A' alone finds students with at least one A.

1796. **B** — The theta join finds every employee who earns less than someone, which is Asad, Bina and Dua. Subtracting them from all names leaves those whom nobody out-earns, the top earners: Cyrus and Ehsan, tied at 90. Ties are all kept.

1797. **A** — If no B value is shared, nothing matches and the join is empty (0). If every tuple in both relations has the same B value, every R tuple matches every S tuple: 4 × 5 = 20. 9 is |R| + |S|, which applies to union, not join.

1798. **A** — For any employee, there is some Works tuple belonging to someone else, so the inequality is satisfied and every eid survives. 'Not related to any' needs difference: π_eid(Emp) − π_eid(Works) = {E4}. Inequality in a join means 'differs from at least one', not 'differs from all'.