DATABASE ENGINEERING

Most Expected Topics

Module-wise High-Chance Topics, Answered

Built from the handwritten module list shared, cross-checked against PYQ R25, B26 and R24

Module 1 — Minimal cover, candidate key, lossless decomposition, dependency preserving, normalization theory, equivalence

Module 2 — ER diagram, DDL and DML

Module 3 — Relational algebra, DML/DCL/DQL, join theory

Module 4 — Armstrong's axioms

Module 5 — View and conflict serializability, transaction, ACID

Compiled from the official five-unit syllabus, the previous-year question papers (R25, B26, R24), the class slide decks, the model question banks and the handwritten notes. Every question is answered in full; differences are shown in two-column tables and ER diagrams are drawn wherever a question needs one.

How this list maps to the paper

The handwritten note groups the syllabus into five revision modules rather than the five teaching units. Every topic on it also appears in the question papers, which is why it is worth revising in this order.

Note module	Topics listed	Syllabus unit	How often it appears in PYQ
Module 1	Minimal cover, candidate key, lossless decomposition, dependency preserving, normalization theory, equivalence	Unit IV	Every paper — Q5 is always a 7-mark FD or normalization question, plus 2-mark parts
Module 2	ER diagram, DDL and DML	Unit I and Unit III	ER diagram in every paper; DDL/DML in B26 Q4(b) and R25 Q4(b)
Module 3	Relational algebra, DML/DCL/DQL, join theory	Unit III	Join theory in all three papers; algebra queries in all three
Module 4	Armstrong's axioms	Unit IV	R25 Q1(l), R24 Q1(k) — and it is the basis of every closure question
Module 5	View serializability, conflict serializability, transaction, ACID	Unit V	Conflict serializability in all three; view in R25 Q1(m), R24 Q1(m)

The note also carries the line **"if Y ⊆ X, then X → Y"** next to Reflexivity — that is the reflexivity axiom, and it is covered in Module 4 below.

Module 1 — Functional Dependencies and Normalization

Q1. What is a minimal cover? Give the algorithm and find one for F = {A → B, C → B, D → ABC, AC → D}.

Long answer — 7 marks

Definition

A **minimal cover** (canonical cover) Fc is a set of functional dependencies equivalent to F that has no unnecessary part. It must satisfy three conditions: every right-hand side is a single attribute; no left-hand-side attribute is extraneous; and no whole dependency is redundant.

Algorithm

1. **Split** every right-hand side into single attributes. 2. **Remove extraneous left-hand attributes** — for each XA → Y, compute X+ under the current set; if Y ∈ X+, drop A. 3. **Remove redundant dependencies** — for each X → Y, compute X+ without that FD; if Y is still in X+, drop the FD. 4. **Verify** equivalence with the original set by checking closures in both directions.

Worked example

F = {A → B, C → B, D → ABC, AC → D}. Step 1 — split D → ABC into D → A, D → B, D → C. The set is now {A → B, C → B, D → A, D → B, D → C, AC → D}. Step 2 — test the left-hand side of AC → D. Removing A leaves C, and C+ = {C, B}, which does not contain D. Removing C leaves A, and A+ = {A, B}, which also does not contain D. Neither attribute is extraneous, so AC → D stays. Step 3 — test redundancy. Remove D → B: D+ becomes {D, A, C, B} through D → A, D → C and A → B, which contains B, so **D → B is redundant** and is dropped. Test the others: A → B is needed (A+ alone is {A}), C → B is needed (C+ alone is {C}), D → A and D → C are needed, and AC → D is needed. **Minimal cover = {A → B, C → B, D → A, D → C, AC → D}.**

Two shorter standard examples

F = {A → BC, B → C}: split gives {A → B, A → C, B → C}; A → C is redundant because A → B and B → C, so **Fc = {A → B, B → C}**. F = {AB → C, A → B}: B is extraneous in AB → C because A+ = {A, B} and AB → C then applies, so the FD becomes A → C and **Fc = {A → B, A → C}**.

Exam tip

Always show the split first, then the left-hand-side reduction, then the redundancy removal, and finally state the equivalence check. Examiners give marks for each stage, even if the final set has a small error.

Q2. How do you find the candidate key of a relation? Use the closure method and the shortcut.

Long answer — 7 marks

Method A — the shortcut

Any attribute that **never appears on the right-hand side** of any FD must belong to every candidate key. Start with those attributes, compute the closure, and add attributes only while the closure is not the whole relation. Then check minimality by removing each attribute in turn.

Method B — the closure method

Compute X+ for single attributes, then pairs, and stop as soon as X+ equals all attributes of R. The smallest sets whose closure is the whole relation are the candidate keys.

Worked example 1 — R(A, B, C, D, E), F = {A → B, B → C, CD → E}

The attributes never on a right-hand side are A and D, so start with AD.

Step	Closure	Reason
Start	{A, D}	Mandatory attributes
1	{A, B, D}	A → B
2	{A, B, C, D}	B → C
3	{A, B, C, D, E}	CD → E

(AD)+ is the whole relation, so **candidate key = {A, D}**.

Worked example 2 — R(A, B, C, D, E), F = {A → B, BC → D, D → E}

A+ = {A, B} and C+ = {C}; neither is the whole relation. Testing pairs, (AC)+ = {A, C, B, D, E} = all attributes, so **candidate key = {A, C}**. The single closures are A+ = {A, B}, C+ = {C}, D+ = {D, E}, and (BC)+ = {B, C, D, E}.

Worked example 3 — R(A,B,C,D,E,F,G,H,I,J,K), F = {A → BC, A → DE, B → F, F → GH, D → IJ, K → B}

Never on a right-hand side: A, E, K. But E is already determined by A, so it is not needed separately.

Set	Closure
A+	A, B, C, D, E, F, G, H, I, J (missing K)
K+	K, B, F, G, H
(A,K)+	All eleven attributes

The only candidate key is {A, K}.

Why this matters

Every normalization question starts by finding the key. Without it you cannot decide whether a dependency is partial (2NF), transitive (3NF) or a BCNF violation, because all three tests are defined relative to the key and the prime attributes.

Q3. Explain lossless join decomposition with the test, and check the given examples.

Long answer — 7 marks

Definition

A decomposition of R into fragments R1, R2, …, Rn is **lossless** if the natural join of the fragments reproduces exactly the original relation. It is **lossy** if the join creates tuples that were never in R — these are called **spurious tuples**, and information has been destroyed.

Binary test

For R split into R1 and R2, the decomposition is lossless with respect to F if at least one of these holds: (R1 ∩ R2) → R1 or (R1 ∩ R2) → R2 In words, the common attributes must functionally determine all attributes of at least one fragment.

Worked example — lossless

R(A, B, C, D) with F = {A → B, B → C, C → D}, decomposed into R1(A, B, C) and R2(C, D). The common attribute is C, and C → D holds, so C → R2. **The decomposition is lossless.**

Worked example — lossy

R(A, B, C, D, E) with F = {A → BC, CD → E}, decomposed into R1(A, B, C) and R2(C, D, E). The common attribute is C, and C+ = {C} under F, so C determines neither R1 nor R2; CD → E needs D as well. **The decomposition is lossy.**

Standard checks to memorise

Relation and F	Decomposition	Common attribute	Result
R(ABCD), A→B, B→C, C→D	R1(ABC), R2(CD)	C, and C → D	Lossless
R(ABCD), A→B, B→C, C→D	R1(ACD), R2(BC)	C, determines neither	Lossy
R(ABC), A→B	R1(AB), R2(AC)	A, and A → AB	Lossless
R(ABC), A→B	R1(AB), R2(BC)	B, determines neither	Lossy
R(ABCD), A→B, C→D	R1(AB), R2(CD)	None	Lossy
R(UVWXY), U→VW, V→X, W→Y	R1(UVW), R2(VX), R3(WY)	V and W are keys	Lossless

Why every split on a violating FD is lossless

If X → Y is a non-trivial FD and the relation is split into XY and (R − Y), the common attributes are X, and X determines XY, so the binary test is satisfied immediately. This is why BCNF decomposition algorithms are lossless by construction.

Q4. Explain dependency preservation with the test and standard examples.

Long answer — 7 marks

Definition

A decomposition is **dependency preserving** if every FD of the original set can be enforced by checking the fragments individually, without joining them. Formally, the union of the projections of F onto the fragments must imply F.

Test

For each FD X → Y in F, compute the closure of X using only the FDs that lie inside each fragment — that is, only those FDs whose attributes all appear in the same Ri. If Y appears in that closure for every FD, the decomposition preserves all dependencies.

Worked example — preserved

R(A, B, C, D) with F = {A → B, B → C, C → D}, decomposed into R1(A, B, C) and R2(C, D). Inside R1 we have A → B and B → C; inside R2 we have C → D. Every FD lives inside one fragment, so **the**

decomposition is dependency preserving — and by the earlier check it is also lossless.

Worked example — not preserved

The same relation decomposed into R1(A, C, D) and R2(B, C). R2 keeps B → C, and R1 keeps C → D, but **A → B cannot be checked anywhere**, because A and B never appear in the same fragment. The decomposition is not dependency preserving.

The important distinction

Losslessness and dependency preservation are **independent** properties. The decomposition R(A,B,C,D,E) with F = {A → BC, CD → E} into R1(A,B,C) and R2(C,D,E) is dependency preserving but lossy. A BCNF decomposition is always lossless but may lose dependencies; a 3NF decomposition can always be made both.

The synthesis algorithm for 3NF

Compute a minimal cover; make one relation for each FD in it; add a relation for a candidate key if none of the fragments contains one; remove any relation contained in another. The result is lossless, dependency preserving and in 3NF.

Q5. Explain normalization theory up to BCNF with one running example.

Long answer — 7 marks

The need

An un-normalized relation stores the same fact many times, so it suffers from three anomalies: an **insertion anomaly** (a new fact cannot be recorded without an unrelated fact), an **update anomaly** (one change needs many row updates) and a **deletion anomaly** (deleting one fact destroys another).

The running relation

ENROLL(StudentID, CourseID, StudentName, CourseName, Grade) with SID → SName, CID → CName and {SID, CID} → Grade. The candidate key is {SID, CID}.

The four normal forms

Normal form	Condition	What it removes	Our relation
1NF	Atomic values, no repeating groups	Multivalued cells	Already in 1NF
2NF	1NF plus no partial dependency	Dependency on part of a composite key	Violated: SID → SName, CID → CName
3NF	2NF plus no transitive dependency of a non-prime attribute	Dependency through another non-prime attribute	Violated once department data is added
BCNF	Every determinant of a non-trivial FD is a super key	Remaining anomalies from overlapping keys	Checked after 3NF

Decomposition

Step	Relations produced
2NF	STUDENT(SID, SName), COURSE(CID, CName), ENROLL(SID, CID, Grade)
3NF	Adding DeptID → DeptName gives STUDENT(SID, SName, DeptID) and DEPARTMENT(DeptID, DeptName)

The classic 3NF-but-not-BCNF relation

R(Student, Course, Instructor) with {Student, Course} → Instructor and Instructor → Course. The candidate keys are {Student, Course} and {Student, Instructor}. Instructor → Course violates BCNF

because Instructor is not a super key, but it satisfies 3NF because Course is a prime attribute. The BCNF decomposition is R1(Instructor, Course) and R2(Student, Instructor).

One-line summary

1NF is about atomic values, 2NF about partial dependency, 3NF about transitive dependency, BCNF about every determinant being a key, 4NF about multivalued dependency and 5NF about join dependency.

Q6. When are two sets of functional dependencies equivalent? Test the standard pairs.

Long answer — 7 marks

Condition

F ≡ G if and only if F+ = G+, which holds exactly when every FD of G is implied by F and every FD of F is implied by G. In practice, check each FD with attribute closure, or compare the minimal covers of the two sets.

Test 1 — R(W, X, Y, Z), FD1 = {W → X, X → Y, YW → Z}, FD2 = {W → X, X → Y, W → Z}

Under FD1, W+ = {W, X, Y, Z}: W → X, X → Y, then YW → Z gives Z. So W → Z is implied. Under FD2, (YW)+ = {Y, W, X, Z}: W → X and W → Z. So YW → Z is implied. **The sets are equivalent.**

Test 2 — R(A, B, C, D, E, F), F1 = {A → BC, B → DE, E → AB}, F2 = {A → BCF, B → DE, E → AB}

Under F1, A+ = {A, B, C, D, E}; F is missing, so A → F is not implied and F1 does not imply F2. Under F2, A → BCF implies A → BC, so F2 implies F1. **Not equivalent — F2 is strictly stronger.**

Test 3 — R(A, B, C, D, E), F = {A → BC, AB → C, D → AC, D → E}, G = {A → BC, D → AE}

Under F, A+ = {A, B, C} and D+ = {D, A, C, E, B} = all attributes, so D → AE is implied. Under G, D+ = {D, A, E, B, C} = all, so D → AC is implied, and AB → C follows from A → BC. **The sets are equivalent.**

Shortcut for the exam

Compute the minimal cover of each set. If the two minimal covers are the same, the sets are equivalent. If one contains a dependency the other cannot derive, they are not.

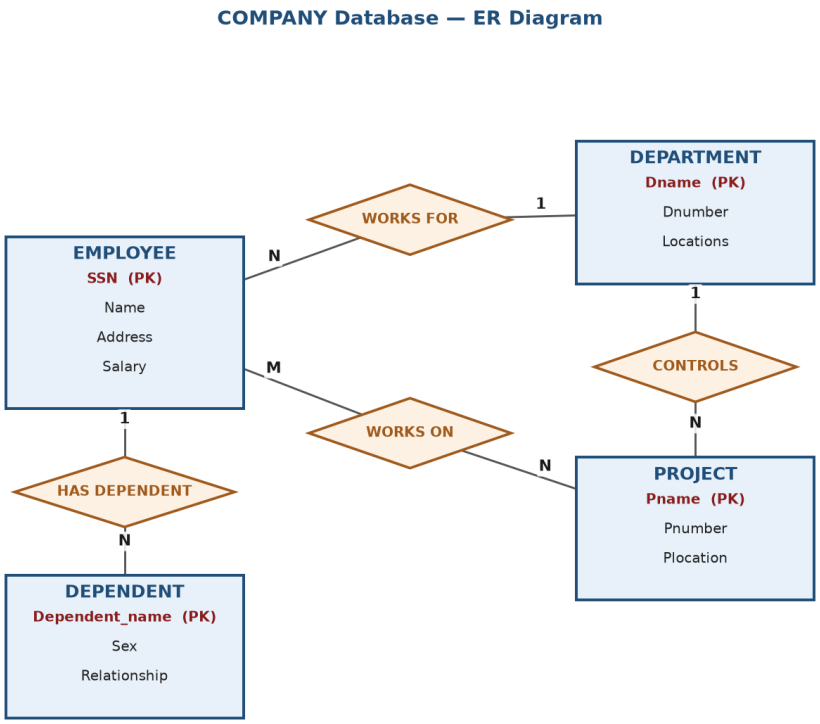
Module 2 — ER Diagram, DDL and DML

Q1. Draw the ER diagram for a company database and map it to tables.

Long answer — 7 marks. This is the standard ER question; the bank and hospital versions work the same way.

Problem

A company has departments with a name, number and a manager (start date stored), and several locations. Departments control projects, each with a unique name, unique number and one location. Employees have name, SSN, address, salary, sex and birthdate, work for exactly one department, and may work on several projects with hours per week recorded. Employees may have dependents with name, sex, birthdate and relationship.



WORKS_ON carries the descriptive attribute Hours. DEPENDENT is a weak entity (partial key Dependent_name).

The four decisions that earn marks

1. **Cardinality** — WORKS_FOR is N:1 (many employees to one department) with total participation on the employee side, because every employee must belong to a department. 2. **Descriptive attributes** — Hours belongs to WORKS_ON, not to EMPLOYEE or PROJECT; Start_date belongs to MANAGES. 3. **Multivalued attribute** — Locations becomes its own table when mapped. 4. **Weak entity** — DEPENDENT has only the partial key Dependent_name and must be identified through its owner, so its key becomes (SSN, Dependent_name).

Mapping

Table	Attributes	Key and constraints
EMPLOYEE	SSN, Name, Address, Salary, Sex, Birthdate, Dnumber	PK: SSN; FK: Dnumber to DEPARTMENT
DEPARTMENT	Dname, Dnumber, Mgr_ssn, Mgr_start_date	PK: Dnumber; FK: Mgr_ssn to EMPLOYEE
DEPT_LOC	Dnumber, Dlocation	PK: (Dnumber, Dlocation)
PROJECT	Pname, Pnumber, Plocation, Dnum	PK: Pnumber; FK: Dnum to DEPARTMENT
WORKS_ON	Essn, Pno, Hours	PK: (Essn, Pno); both FKs
DEPENDENT	Essn, Dependent_name, Sex, Birthdate, Relationship	PK: (Essn, Dependent_name); FK: Essn

If the paper gives a different scenario

The steps never change: find the entity sets and their keys, add attributes with the correct type, place the relationships with cardinality and participation, mark weak entities and descriptive attributes, then map. Always write one line of assumptions — examiners award marks for stated assumptions.

Q2. Explain DDL and DML with syntax and examples.

Long answer — 7 marks

DDL — Data Definition Language

DDL defines and modifies the **structure** of database objects. It writes to the data dictionary and is normally auto-committed.

Statement	Syntax	Purpose
CREATE TABLE	CREATE TABLE Emp (EmpID INT PRIMARY KEY, Name VARCHAR(30) NOT NULL, Salary NUMBER(10,2), DeptID INT REFERENCES Dept(DeptID));	Creates a table with constraints
ALTER TABLE	ALTER TABLE Emp ADD (Phone VARCHAR(15));	Adds, modifies or drops a column
DROP TABLE	DROP TABLE Emp;	Removes the structure and the data
TRUNCATE	TRUNCATE TABLE Emp;	Removes all rows, keeps the structure
CREATE INDEX	CREATE INDEX idx_name ON Emp(Name);	Builds an index
CREATE VIEW	CREATE VIEW CSE AS SELECT * FROM Emp WHERE DeptID = 10;	Defines a virtual table

DML — Data Manipulation Language

DML changes the **data** inside the tables and participates in transactions, so it can be rolled back.

Statement	Syntax	Purpose
INSERT	INSERT INTO Dept VALUES (10, 'HR');	Adds one row
INSERT multi-row	INSERT INTO Dept VALUES (20,'IT'), (30,'Finance');	Adds several rows at once
UPDATE	UPDATE Emp SET Salary = Salary * 1.10;	Modifies existing rows
DELETE	DELETE FROM Emp WHERE Salary < 50000;	Removes selected rows

How to remember the difference

DDL changes the **shape**, DML changes the **content**. DROP removes the table itself, whereas DELETE removes rows and leaves the table standing — that distinction is a favourite 2-mark question.

Module 3 — Relational Algebra, DML/DCL/DQL and Join Theory

Q1. Explain the relational algebra operations with examples.

Long answer — 7 marks

Set theoretic operators

Operator	Symbol	Meaning	Example
Union	R ∪ S	Tuples in R or S or both	CSE students ∪ AIDS students
Intersection	R ∩ S	Tuples in both	Students in both the cricket and chess teams
Difference	R − S	Tuples in R but not S	Students who registered but did not appear
Cartesian product	R × S	Every pairing of tuples	STUDENT × COURSE before selection

Union, intersection and difference require the two relations to be **union compatible** — same degree and corresponding attributes from the same domains.

Division

R ÷ S returns the values of R's other attributes that are paired with **every** value of S. It is the only operator that expresses "for all" directly: WORKS_ON ÷ ALL_PROJ gives the employees who work on every project.

Relational operators

Operator	Symbol	Meaning
Selection	$\sigma_{cond}(R)$	Rows satisfying the condition
Projection	$\pi_{attrs}(R)$	Only the listed columns
Join	$R \bowtie S$ or $R \bowtie_\theta S$	Related tuples from two relations
Rename	$\rho_S(R)$	Renames a relation or its attributes

A worked query

Names of CSE students enrolled in DBMS:

$\pi_{Name}(\sigma_{Dept='CSE' \wedge Title='DBMS'}((STUDENT \bowtie ENROLLMENT) \bowtie COURSE))$ The joins reconstruct the M:N relationship through its associative table, the selection applies both conditions, and the projection keeps the required column. Pushing $\sigma_{Dept='CSE'}$ down to STUDENT first is the optimised form.

The three-step method for any algebra question

Find the relations holding the required attributes, join them on their common keys, then apply σ for conditions and π for the output columns.

Q2. Explain DML, DCL and DQL.

Long answer — 7 marks

DML

DML manipulates the data: INSERT adds rows, UPDATE modifies them, DELETE removes them. DML statements are transactional and can be rolled back.

DCL — Data Control Language

DCL controls **who may do what**. GRANT gives a privilege and REVOKE takes it away.

GRANT SELECT, INSERT ON Emp TO user1;

REVOKE SELECT ON Emp FROM user1;

Privileges may be given on a table, a view or a column, and may be passed on with the WITH GRANT OPTION clause.

DQL

DQL retrieves data and has one statement, SELECT, with the general form:

SELECT columns FROM tables WHERE row-condition GROUP BY columns HAVING group-condition ORDER BY columns;

Example: SELECT DeptID, AVG(Salary) FROM Emp GROUP BY DeptID HAVING AVG(Salary) > 50000 ORDER BY DeptID;

Comparison

Category	Purpose	Statements	Rollback
DDL	Define structure	CREATE, ALTER, DROP, TRUNCATE	Usually not
DML	Change data	INSERT, UPDATE, DELETE	Yes
DQL	Retrieve data	SELECT	Not applicable
DCL	Control access	GRANT, REVOKE	Yes
TCL	Control transactions	COMMIT, ROLLBACK, SAVEPOINT	Not applicable

Q3. Explain the join operations — the theory.

Long answer — 7 marks. Appears in all three papers.

What a join does

A join combines tuples from two relations that satisfy a matching condition. It is the operation that reconstructs information that normalization has distributed across tables, and it is the reason the relational model needs no pointers.

Join	Condition	Duplicate join column	Unmatched tuples
Theta	Any comparison (=, ≠, <, >, ≤, ≥)	Kept	Dropped
Equi	Equality only, written explicitly	Kept	Dropped
Natural	Equality on all same-named attributes	Removed	Dropped
Left outer	Any	Kept	Left side kept
Right outer	Any	Kept	Right side kept
Full outer	Any	Kept	Both sides kept
Self	Any, with the relation renamed	Kept	Depends on the type

Natural versus equi join

In an equi join the condition is written out and the join column appears twice in the result. In a natural join the condition is implicit — the DBMS joins on all identically named attributes and keeps one copy. A natural join is therefore an equi join with the duplicate column removed.

Theta join example

EMPLOYEE $\bowtie_{\text{Salary} \geq \text{MinSalary} \wedge \text{Salary} \leq \text{MaxSalary}}$ SALARY_GRADE assigns each employee the grade band his salary falls into. A theta join is required whenever the match is a range rather than an exact value.

Outer join example

CUSTOMER LEFT OUTER JOIN ORDERS keeps every customer, including those who never placed an order, with NULL in the order columns. This is why outer joins exist — an inner join would silently drop those customers and give a wrong report.

Self join example

$\pi_{\text{E.EmpName} \to \text{Employee, M.EmpName} \to \text{Manager}}(\rho_{E}(\text{EMPLOYEE}) \bowtie_{\text{E.ManagerID} = \text{M.EmpID}} \rho_{M}(\text{EMPLOYEE}))$ lists each employee with his manager. The top-level manager disappears because the join drops the NULL, so a left outer self join is needed to retain him.

Multi-attribute join

RESULT $\bowtie_{\text{RESULT.StudentID} = \text{ATTENDANCE.StudentID} \wedge \text{RESULT.CourseID} = \text{ATTENDANCE.CourseID}}$ ATTENDANCE matches on both key attributes; matching on StudentID alone would pair marks from one course with attendance from another.

Module 4 — Armstrong's Axioms

Q1. State and prove Armstrong's axioms and the derived rules.

Long answer — 7 marks. Also asked as a 2-mark definition.

The three axioms

Axiom	Statement	Example
IR1 Reflexivity	If $Y \subseteq X$ then $X \to Y$	{A, B} → A
IR2 Augmentation	If $X \to Y$ then $XZ \to YZ$	A → B implies AC → BC
IR3 Transitivity	If $X \to Y$ and $Y \to Z$ then $X \to Z$	A → B and B → C imply A → C

Reflexivity — the statement in the note

If Y ⊆ X, then X → Y. Take any two tuples t1 and t2 with t1[X] = t2[X]. Because Y is a subset of X, they also agree on every attribute of Y, so t1[Y] = t2[Y]. Therefore X → Y holds. This is why a set of attributes always determines its own subsets, and why trivial dependencies such as AB → A need no checking.

Augmentation — proof

Assume X → Y. Take tuples with t1[XZ] = t2[XZ]; then t1[X] = t2[X] and t1[Z] = t2[Z]. From X → Y, t1[Y] = t2[Y]. Combining gives t1[YZ] = t2[YZ], so XZ → YZ.

Transitivity — proof

Assume X → Y and Y → Z. If t1[X] = t2[X], then by X → Y, t1[Y] = t2[Y], and then by Y → Z, t1[Z] = t2[Z]. Hence X → Z.

Derived rules

Rule	Statement	How it follows
Union	If X → Y and X → Z then X → YZ	Augment X → Y with X and X → Z with Y, then use transitivity
Decomposition	If X → YZ then X → Y and X → Z	Use reflexivity YZ → Y and then transitivity
Pseudotransitivity	If X → Y and WY → Z then WX → Z	Augment X → Y with W, then use transitivity
Composition	If X → Y and Z → W then XZ → YW	Augment each, then combine

Why the axioms matter

They are **sound** — every dependency they produce is true in every relation satisfying F — and **complete** — every dependency implied by F can be derived from F using them. Completeness is what justifies using attribute closure as the test for implication, and it is the theoretical basis of minimal cover and of 3NF synthesis.

Module 5 — Transactions, ACID and Serializability

Q1. What is a transaction? Explain ACID properties.

Long answer — 7 marks

Definition

A **transaction** is a logical unit of database work made of one or more operations that must succeed or fail as a whole. Example: transferring ₹1,000 from A to B — Read(A), A = A − 1000, Write(A), Read(B), B = B + 1000, Write(B), COMMIT.

Property	Meaning	Ensured by
Atomicity	All operations happen or none happen	Rollback and undo of incomplete work
Consistency	The database moves from one valid state to another	Constraint checking
Isolation	Concurrent transactions behave as if run alone	Concurrency control and serializability
Durability	Committed changes survive failure	Logging, checkpoints, redo

Why each matters

If the transfer fails after debiting A, **atomicity** undoes the debit, otherwise money disappears. **Consistency** guarantees the total of all balances is unchanged. **Isolation** stops another transaction from reading the half-finished state. **Durability** guarantees that once the customer is told the transfer succeeded, a power cut cannot undo it.

Transaction states

Active → Partially committed → Committed, or Active → Failed → Aborted, ending in Terminated. An aborted transaction may be restarted unless the failure was a logic error.

Q2. Check conflict serializability using a precedence graph.

Long answer — 7 marks. Appears in all three papers.

The conflict rule

Two operations conflict when they belong to **different transactions**, access the **same data item**, and at least one is a **write**.

Pair	Conflict	Reason
R1(X), R2(X)	No	Read and read
R1(X), W2(X)	Yes	Read and write
W1(X), R2(X)	Yes	Write and read
W1(X), W2(X)	Yes	Write and write
R1(X), W2(Y)	No	Different data items

Method

Create one node per transaction; for every conflicting pair, draw an edge from the earlier transaction to the later one; if the graph has a directed cycle the schedule is **not** conflict serializable; if it is acyclic, any topological order gives the equivalent serial schedule.

Worked example 1 — serializable

S = R1(X), W1(X), R2(X), W2(X). Conflicts: W1(X) before R2(X) gives T1 → T2, and W1(X) before W2(X) gives T1 → T2. Acyclic, so **conflict serializable** with serial order T1 → T2.

Worked example 2 — not serializable

S = R1(X), R2(Y), W1(Y), W2(X). Conflicts: R1(X) before W2(X) gives T1 → T2; R2(Y) before W1(Y) gives T2 → T1. The cycle T1 ↔ T2 means **not conflict serializable**.

Worked example 3 — three transactions

S = R1(X), W1(X), R2(X), W2(Y), R3(Y), W3(Y). Edges: W1(X) before R2(X) gives T1 → T2; W2(Y) before R3(Y) and W3(Y) gives T2 → T3. Acyclic, so **conflict serializable** with order T1 → T2 → T3.

Worked example 4 — the PYQ schedule

S1: R1(A), R2(B), W1(B), W2(A). Conflicts: R2(B) before W1(B) gives T2 → T1; R1(A) before W2(A) gives T1 → T2. Cycle, so **not conflict serializable**.

Common mistakes

Treating two reads as a conflict; drawing an edge for operations on different data items; ignoring write-write conflicts; drawing the edge in the wrong direction; and assuming an acyclic graph has only one serial order — it may have several valid topological orders.

Q3. Explain view serializability with an example.

Long answer — 7 marks. Asked as 2 marks in R25 and R24.

Definition

A schedule is **view serializable** if it is view equivalent to some serial schedule. Two schedules are view equivalent when all three conditions hold:

Condition	Meaning
Initial read	If Ti reads the initial value of X in one schedule, it must read the initial value in the other
Read-from	If Ti reads X written by Tj in one schedule, the same must hold in the other

Condition	Meaning
Final write	The transaction that performs the final write of X must be the same in both

Relationship with conflict serializability

Every conflict-serializable schedule is view serializable, but not every view-serializable schedule is conflict serializable. View serializability is the **larger** class; the extra schedules are those involving **blind writes**.

The blind write example

A **blind write** is a write performed without first reading the item. Consider S = W1(X), W2(X), W1(X). On X, W1(X) before W2(X) gives T1 → T2, and W2(X) before W1(X) gives T2 → T1, so the precedence graph has a cycle and the schedule is **not conflict serializable**. But it is **view serializable**: no transaction reads X, so the read-from conditions are vacuous, and T1 performs the final write in both the schedule and the serial order T1 → T2.

Comparison

Aspect	Conflict Serializability	View Serializability
Basis	Order of conflicting operations	Read-from relations and final writes
Testing	Precedence graph, simple	Three conditions, harder
Blind writes	May reject valid schedules	Accepts some blind-write schedules
Relationship	A subset of view-serializable schedules	A superset of conflict-serializable schedules
Exam technique	Draw the graph, check for a cycle	Check the three conditions

How to answer in the exam

Test conflict serializability first with the graph, because it is quicker. If the graph has a cycle, check for blind writes — only then is it worth testing view serializability, and only a schedule with blind writes can be view- but not conflict-serializable.

Final revision order

Priority	Topic	Why
1	Types of keys and candidate-key finding	Feeds every normalization question and is a 7-mark question itself
2	Minimal cover and equivalence	Module 1 core; two of the three papers ask it
3	Lossless join and dependency preservation	Always a 2-mark part and often inside the 7-mark
4	Normalization to BCNF	Q5 in every paper
5	Conflict serializability by precedence graph	Q6 in every paper
6	ER diagram and mapping	Q3 in every paper
7	Joins — natural, theta, equi, outer, self	Q4 in every paper
8	Relational algebra and SQL queries	Q4 in every paper
9	Armstrong's axioms	Short question, and the theory behind closure
10	ACID and transaction states	Q6 short part
11	View serializability	2-mark definition plus one 7-mark chance

Read the four unit files for the full treatment of each topic, and this file in the last two days before the paper.