DATABASE ENGINEERING

PYQ Answers

Previous Year Question Papers — Paper-wise Solved

Paper 1 — R25, Third Semester Regular Examination 2025, CS242

Paper 2 — B26, Fourth Semester Backlog Examination 2026, CSE23306

Paper 3 — R24, Fourth Semester Regular Examination 2025, CS242

Part A questions (2 marks) and Part B questions (7 marks) are answered separately for each paper

The Green Chemistry paper (R26, CY201) in the same PDF is not a Database Engineering paper and is excluded

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.

Paper 1 — R25, Third Semester Regular Examination 2025, CS242

Part A — 2-Mark Answers

Q1. Explain exhaustive relationship in an ER model with an example.

The wording of this question is unclear in the paper. The two accepted readings are: (1) an **exhaustiveness (total) constraint** on a specialization, meaning every entity of the superclass must belong to some subclass, as when every EMPLOYEE must be a secretary, engineer or technician; or (2) an **exclusive (disjoint) relationship**, meaning an entity may belong to at most one subclass, so the subclasses do not overlap. Some question banks use the phrase loosely for a **recursive relationship**, in which an entity set is related to itself, such as an employee supervising another employee. State the reading you adopt and give one example of it.

Q2. Differentiate between strong entity sets and weak entity sets with characteristics, key dependencies and examples.

Strong Entity Set	Weak Entity Set
Has its own key attribute	Has only a partial key, no key of its own
Exists independently	Existence depends on an owner entity set
Single rectangle in the diagram	Double rectangle in the diagram
Participation may be partial	Total participation in the identifying relationship
Example: EMPLOYEE, STUDENT	Example: DEPENDENT of EMPLOYEE, PAYMENT of LOAN

The key dependency is the essential difference: a weak entity's full key is the owner's primary key plus its own partial key, so (SSN, Dependent_name) identifies a dependent.

Q3. Highlight the important tasks handled by a Database Administrator.

The DBA defines the conceptual, external and internal schemas and their mappings; decides storage structures, indexes and access paths; grants and revokes user privileges and defines views; declares and monitors integrity constraints; monitors and tunes performance; plans backup and performs recovery; identifies user requirements and chooses the DBMS; and maintains the data dictionary and system documentation.

Q4. Explain the key terminologies of the Relational Model — relation, tuple, domain and cardinality.

A **relation** is a table of values, formally a subset of the Cartesian product of a set of domains. A **tuple** is one row of that table. A **domain** is the set of atomic values an attribute may take, such as integers from 0 to 120 for Age. **Cardinality** is the number of tuples, that is the number of rows. The number of attributes is called the **degree** or arity.

Q5. List any two Relational Model Constraints and describe their significance.

Entity integrity — the primary key may not be NULL, so every tuple can be identified. **Referential integrity** — a foreign key value must either match an existing primary key value in the referenced relation or be wholly NULL, so relationships never point to non-existent rows. The other two are domain constraints and key constraints.

Q6. State the difference between single-valued, multivalued, stored and derived attributes.

Type	Meaning	Example
Single-valued	One value per entity	Date_of_birth
Multivalued	A set of values per entity	Phone_no
Stored	Physically stored in the database	Date_of_birth, Salary
Derived	Computed from stored attributes	Age from DOB, Total from Qty and Rate

Q7. Outline the Selection and Projection operations with syntax.

Selection returns the rows satisfying a condition and does not change the degree: $\sigma_{Salary > 50000}(EMPLOYEE)$. **Projection** returns only the named columns and may reduce the degree: $\pi_{Name, Salary}(EMPLOYEE)$. Selection works horizontally, projection vertically, and duplicate tuples produced by projection are removed.

Q8. Write the SQL query for students whose name is 7 characters long and starts with M.

```
SELECT sname FROM Student WHERE sname LIKE 'M_____';
```

The pattern is the letter M followed by exactly six underscores, giving seven characters in total.

Q9. Compare and contrast the RENAME operator and the SQL SORT command.

RENAME, written $\rho_S(R)$ or $\rho_{S(B1,B2)}(R)$, changes the name of a relation or its attributes so that the same relation can be used twice in one expression. The SQL SORT command, that is ORDER BY, does not change names at all — it arranges the output rows in ascending or descending order of a column. RENAME changes the schema; ORDER BY changes only the presentation.

Q10. For R(STUVW) with F = {ST → UW, V → W}, find the minimum cover and the candidate key.

Split the right-hand sides: ST → U, ST → W, V → W. Neither ST → W nor V → W is redundant, because W cannot be derived from the other dependencies. **Minimal cover = {ST → U, ST → W, V → W}.** The attributes never appearing on a right-hand side are S, T and V, and (STV)+ = {S, T, U, V, W}, so **candidate key = {S, T, V}**. If the paper's symbols instead read F = {ST → UVW, V → TW}, the minimal cover is {ST → U, ST → V, ST → W, V → T, V → W} and the candidate key is {S, T}. Show both readings if the question is unclear.

Q11. Explain the difference between partial and transitive dependency.

A **partial dependency** is a non-prime attribute depending on only part of a composite candidate key — StudentID → StudentName when the key is {StudentID, CourseID}. A **transitive dependency** is a non-prime attribute depending on another non-prime attribute — EmpID → DeptID → DeptName. Partial dependencies are removed by 2NF, transitive dependencies by 3NF.

Q12. Explain Armstrong's Axioms and their role in inferring functional dependencies.

Reflexivity — if $Y \subseteq X$ then $X \rightarrow Y$; **augmentation** — if $X \rightarrow Y$ then $XZ \rightarrow YZ$; **transitivity** — if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$. They are sound, producing only true dependencies, and complete, being able to produce every dependency implied by F. Their practical use is that they justify the attribute-closure algorithm, which is how implied dependencies and super keys are actually tested.

Q13. Compare conflict serializability and view serializability with suitable examples.

Conflict Serializability	View Serializability
Based on the order of conflicting operations	Based on read-from relations and final writes
Tested with a precedence graph	Tested by three view-equivalence conditions
Rejects some valid blind-write schedules	Accepts certain blind-write schedules
Every conflict-serializable schedule is view serializable	The converse is not true

Example: W1(X), W2(X), W1(X) has a cyclic precedence graph, so it is not conflict serializable, but it is view serializable because T1 performs the final write in every equivalent order.

Q14. Compare conflict operations — read-write, write-read and write-write.

Two operations conflict when they belong to different transactions, access the same data item, and at least one of them is a write. R1(X) and W2(X) is a **read-write** conflict; W1(X) and R2(X) is a **write-read** conflict; W1(X) and W2(X) is a **write-write** conflict. Two reads never conflict, and operations on different data items never conflict.

Q15. Illustrate the various types of database failures.

Type	Example	Recovery need
Transaction failure	Logical error, constraint violation, deadlock victim	Undo the incomplete transaction
System failure	Power loss, OS or DBMS crash	Redo committed work, undo incomplete work
Media failure	Disk crash, corrupted file	Restore from backup and apply logs
Communication failure	Network break in a distributed system	Resolve incomplete distributed work

Part B — 7-Mark Answers

Q1. Q2(a) Explain the disadvantages of the traditional file processing system and the need for a DBMS.

Disadvantages

Data redundancy and inconsistency — the same fact is stored in several files, so space is wasted and copies disagree after an update. **Program–data dependence** — the file layout is written into every program, so any change forces rewrites. **Difficulty of access** — only pre-written reports are available; a new question needs a new program. **No concurrency control** — simultaneous updates can lose one another. **Weak atomicity and recovery** — a half-completed update leaves the data inconsistent. **Poor security and limited sharing** — access control is per file, not per record or field.

The need for a DBMS

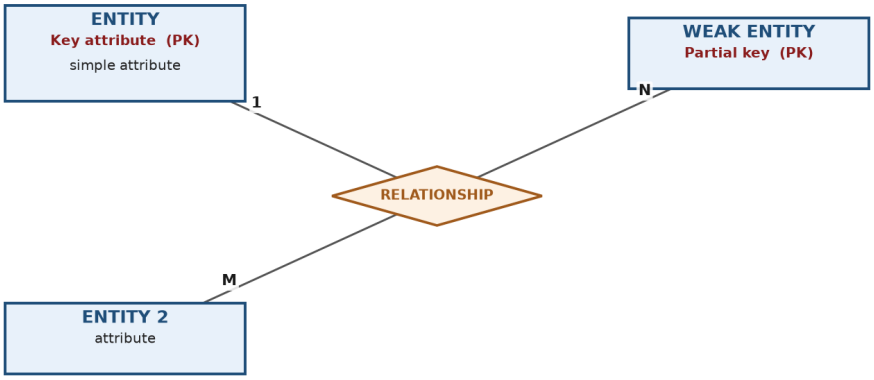
A DBMS stores one integrated database described by a schema held in the data dictionary, so each fact is stored once. It separates programs from data through the three-schema architecture and data independence, offers a high-level query language, enforces integrity constraints centrally, provides concurrency control and recovery through ACID transactions, and supports fine-grained authorisation through views and privileges.

Q2. Q2(b) Explain the major components of the ER model with relationship types.

Components

Entity — a distinguishable real-world object; **entity set** — a collection of entities of the same type, drawn as a rectangle. **Attribute** — a property of an entity, which may be simple, composite, single-valued, multivalued, stored, derived or a key. **Relationship** — an association among entities; **relationship set** — all associations of one type, drawn as a diamond. **Key** — the attribute that identifies an entity uniquely, underlined in the diagram.

ER Model — Basic Notation Used in These Notes



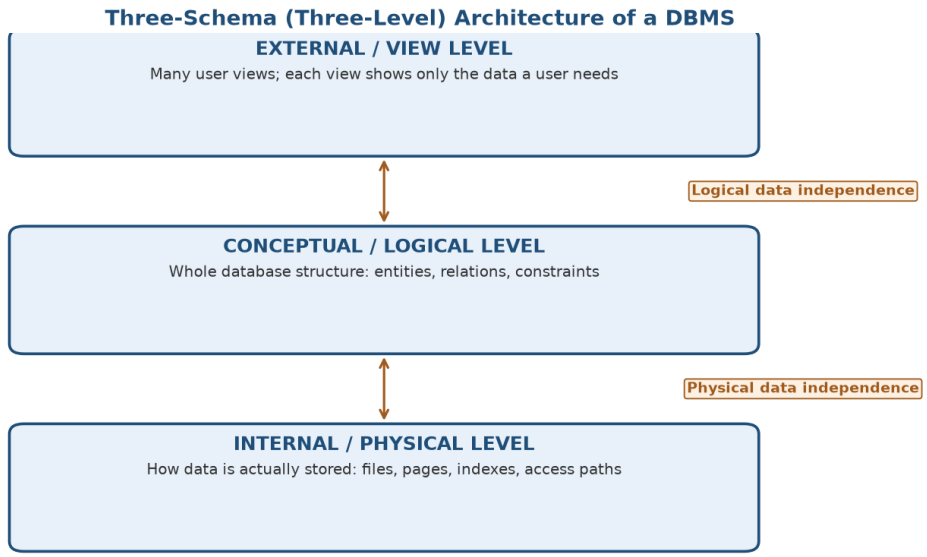
Bold attribute with (PK) = key attribute. Weak entity is drawn in a double-bordered box in a full ER diagram.

Relationship types

Basis	Types
Degree	Unary (recursive), binary, ternary, n-ary
Cardinality ratio	1:1, 1:N, N:1, M:N
Participation	Total (double line), partial (single line)
Attributes on the relationship	Descriptive, such as Grade or Hours

A recursive relationship needs role names, because the same entity set participates twice — as in EMPLOYEE supervises EMPLOYEE.

Q3. Q2(c) Illustrate the three-schema levels with a diagram and explain how data abstraction helps.



External/Conceptual mapping and Conceptual/Internal mapping isolate each level from changes in the level below.

The three levels

External level — many user views, each showing only the data that user needs. **Conceptual level** — the complete logical description of the database: entity sets, attributes, relationships and constraints. **Internal level** — the physical description: file organisations, record layouts, indexes and access paths.

Mappings

The external/conceptual mapping derives each view from the conceptual schema; the conceptual/internal mapping realises the conceptual schema on disk. Because of these mappings, a change at one level need not affect the others, which gives logical and physical data independence.

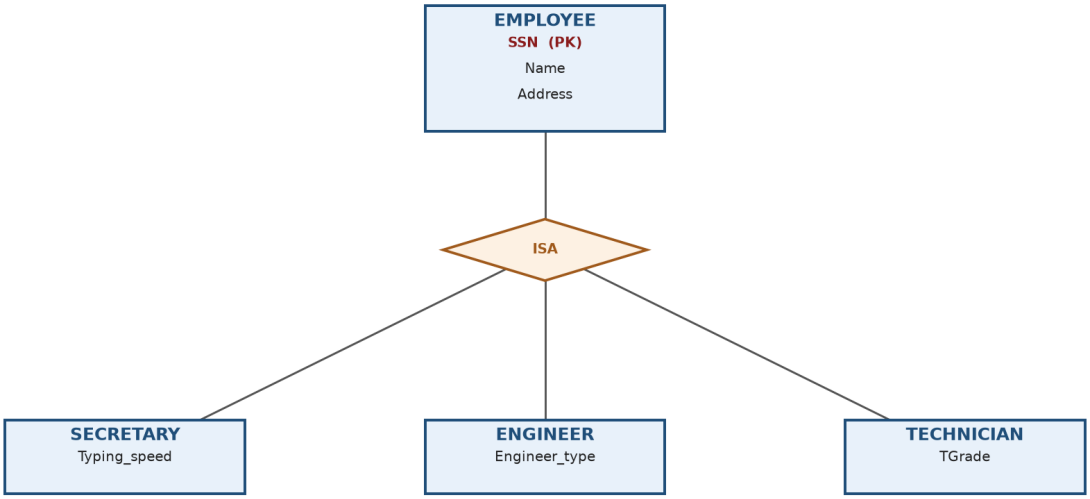
How data abstraction helps

It hides storage detail from users and application programmers, so a naive user sees only a form and a programmer sees only the conceptual schema. This makes the system easier to use, protects programs from storage changes, allows the same data to be presented differently to different users, and improves security because each user can be restricted to a view.

Q4. Q3(a) Explain Generalization and Specialization in the EER model with diagrams.

Generalization is bottom-up: entity sets sharing common attributes are combined into one superclass, and the common attributes move up to it. **Specialization** is top-down: a superclass is divided into subclasses on the basis of a distinguishing characteristic, and subclass-specific attributes move down.

Generalization / Specialization — EER Notation

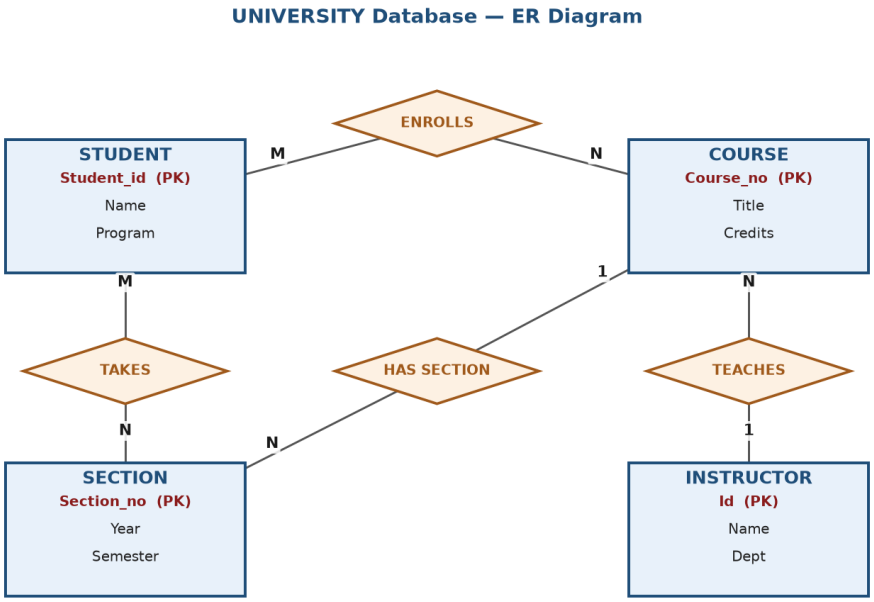


ISA is the specialization/generalization relationship. Attributes are inherited downward; the constraint may be disjoint or overlapping, total or partial.

Generalization	Specialization
Bottom-up	Top-down
Combines entity sets into a superclass	Divides a superclass into subclasses
Removes duplication	Adds specific attributes
Starts from specific entity sets	Starts from one general entity set

The specialization may be **disjoint** (at most one subclass) or **overlapping**, and **total** (every superclass entity belongs to some subclass) or **partial**. Subclasses inherit all superclass attributes and relationships.

Q5. Q3(b) Design an ER diagram for a university database and map it to relational tables.



SECTION is a weak entity of COURSE; its partial key is Section_no. Grade is a descriptive attribute of TAKES.

Design

STUDENT (key Student_id), COURSE (key Course_no), INSTRUCTOR (key Id) and SECTION, a weak entity of COURSE with partial key Section_no. ENROLLS is M:N between STUDENT and COURSE. TEACHES is N:1 from COURSE to INSTRUCTOR. HAS_SECTION is 1:N from COURSE to SECTION with total participation on the SECTION side. TAKES is M:N between STUDENT and SECTION, carrying Grade.

Table	Attributes	Key and constraints
STUDENT	Student_id, Name, Program	PK: Student_id
INSTRUCTOR	Id, Name, Dept	PK: Id
COURSE	Course_no, Title, Credits, Id	PK: Course_no; FK: Id to INSTRUCTOR
SECTION	Course_no, Section_no, Year, Semester	PK: (Course_no, Section_no); FK: Course_no
ENROLLS	Student_id, Course_no	PK: (Student_id, Course_no); both FKs
TAKES	Student_id, Course_no, Section_no, Grade	PK: three columns; FKs to STUDENT and SECTION

The M:N relationships become their own tables, the 1:N relationship is merged into the many side, and the weak entity receives the owner's key.

Q6. Q3(c) Discuss Super Key, Candidate Key, Primary Key, Alternate Key and Foreign Key with examples and their role in data integrity.

Key	Definition	Example in STUDENT(SID, RollNo, Email, Name)
Super key	Any set that uniquely identifies a tuple	{SID}, {SID, Name}, {RollNo, Email}
Candidate key	A minimal super key	{SID}, {RollNo}, {Email}
Primary key	The chosen candidate key	{SID}

Key	Definition	Example in STUDENT(SID, RollNo, Email, Name)
Alternate key	A candidate key not chosen	{RollNo}, {Email}
Foreign key	An attribute referencing another relation's key	DeptID referencing DEPARTMENT
Composite key	A key of two or more attributes	(OrderID, ProductID) in ORDER_ITEM

Role in data integrity

The primary key enforces **entity integrity** — unique and never NULL, so every tuple is identifiable. The foreign key enforces **referential integrity** — every value exists in the parent or is NULL, so no relationship points to a missing row. Alternate keys give additional unique access paths and prevent duplicate entities. Candidate keys guarantee that at least one minimal identifier exists, which is the precondition for normalisation.

Q7. Q4(a) Explain Inner Join and differentiate natural, theta and equi join with examples.

An **inner join** returns only the tuples that satisfy the join condition; unmatched tuples from either side are discarded.

Natural Join	Equi Join	Theta Join
Joins on all same-named attributes	Equality condition written explicitly	Any comparison condition
Condition implicit	Condition uses =	May use <, >, ≤, ≥, ≠
Common column appears once	Join column appears twice	Join column appears twice
$R \bowtie S$	$R \bowtie_{R.A=S.B} S$	$R \bowtie_{\theta} S$

Example

With EMPLOYEE(EmpID, EmpName, DeptID) and DEPARTMENT(DeptID, DeptName), the natural join EMPLOYEE ⋈ DEPARTMENT matches on DeptID and keeps one copy, giving Amit–CSE, Priya–AIDS and so on. The equi join EMPLOYEE $\bowtie_{\text{EMPLOYEE.DeptID} = \text{DEPARTMENT.DeptID}}$ DEPARTMENT gives the same rows but with DeptID appearing twice. A theta join is needed when the match is a range: EMPLOYEE $\bowtie_{\text{Salary} \geq \text{MinSalary} \wedge \text{Salary} \leq \text{MaxSalary}}$ SALARY_GRADE assigns each employee a salary grade.

Q8. Q4(b) Write the SQL queries on the University_Faculty_Details table.

ID	name	salary	dept_name	building	budget
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Taylor	100000
45346	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58883	Calvineri	52000	History	Painter	50000
85821	Brandt	92000	Comp. Sci.	Taylor	100000
13151	Mozart	40000	Music	Packard	80000

ID	name	salary	dept_name	building	budget
35456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

1. Salary between 70,000 and 90,000

SELECT name, dept_name, salary FROM University_Faculty_Details WHERE salary BETWEEN 70000 AND 90000;

Result: Wu 90000 Finance, Katz 75000 Comp. Sci., Kim 80000 Elec. Eng., Gold 87000 Physics, Singh 80000 Finance.

2. Name, department and salary whose name starts with S

SELECT name, dept_name, salary FROM University_Faculty_Details WHERE name LIKE 'S%';

Result: Srinivasan 65000 Comp. Sci., Singh 80000 Finance.

3. All faculty names in ascending order of salary

SELECT name FROM University_Faculty_Details ORDER BY salary ASC;

Result: Mozart, Calvineri, El Said, Srinivasan, Crick, Katz, Kim, Singh, Gold, Wu, Brandt, Einstein.

4. Update all salaries by 10 percent

UPDATE University_Faculty_Details SET salary = salary * 1.10;

5. Display ProductName and Price sorted by ProductName

SELECT ProductName, Price FROM Product ORDER BY ProductName ASC;

The table Product is not part of the given schema, so state the assumption and write the statement against that table.

6. Add a phone_number column

ALTER TABLE University_Faculty_Details ADD (phone_number VARCHAR(15));

7. Insert three new records

INSERT INTO University_Faculty_Details (ID, name, salary, dept_name, building, budget) VALUES (11111,'Raman',71000,'Physics','Watson',70000), (22223,'Bose',68000,'Comp. Sci.','Taylor',100000), (33333,'Saha',59000,'Music','Packard',80000);

Q9. Q4(c) Solve the relational algebra queries on product, client and salesman.

Schema: product(product_id, product_name, manufacturer_name, product_rate, sell_price, product_description, client_id); client(client_id, client_name, address, city, pin, state, bal_due); salesman(salesman_id, salesman_name, address, city, pin, state, salary, product_id).

(i) Names of salesmen and clients from the same city

$\pi_{\text{salesman_name, client_name}}(\sigma_{\text{salesman.city = client.city}}(\text{salesman} \times \text{client}))$

(ii) Salesman name, address and phone for products selling above ₹10,000

$\pi_{\text{salesman_name, address}}(\sigma_{\text{sell_price > 10000}}(\text{salesman} \bowtie \text{product}))$ The attribute phone does not exist in the given schema, so either omit it or state the assumption that a contact column exists.

(iii) Names of salesmen selling Samsung products

$\pi_{\text{salesman_name}}(\sigma_{\text{manufacturer_name = 'Samsung'}}(\text{salesman} \bowtie \text{product}))$

(iv) Client details for Odisha clients with balance due over 20,000

$\sigma_{\text{state = 'Odisha'} \wedge \text{bal_due > 20000}}(\text{client})$

(v) client_id and names of clients who buy washing machines

$\pi_{client_id,\ client_name}(\sigma_{product_name\ =\ 'Washing\ Machine'}(client \bowtie product))$

Q10. Q5(a) State when two FD sets are equivalent, and test R(WXYZ) with FD1 = {W→X, X→Y, YW→Z} and FD2 = {W→X, X→Y, W→Z}.

The condition

F ≡ G if and only if F+ = G+, that is, every FD of G is implied by F and every FD of F is implied by G. In practice each side is tested with attribute closure, or the minimal covers of the two sets are compared.

Test

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

Q11. Q5(b) Explain normalization up to 3NF and lossless join, and normalise R(SSN, ENAME, PNUMBER, PNAME, PLOCATION, HOURS, CITY, STATUS) to BCNF.

The concepts

Normalization decomposes a relation so that each fact is stored once. **1NF** requires atomic values. **2NF** removes partial dependencies on a composite key. **3NF** removes transitive dependencies of non-prime attributes. A decomposition is **lossless** when the natural join of the fragments reproduces the original relation exactly.

Candidate key

SSN+ = {SSN, ENAME, CITY, STATUS}; PNUMBER+ = {PNUMBER, PNAME, PLOCATION}; {SSN, PNUMBER}+ = all eight attributes. **Candidate key = {SSN, PNUMBER}**, so all other attributes are non-prime.

Violations

SSN → ENAME, CITY violates 3NF and BCNF, because SSN is not a super key and the dependents are non-prime. PNUMBER → PNAME, PLOCATION violates similarly. CITY → STATUS violates as well, since STATUS is non-prime.

Decomposition to BCNF

Step	Action	Result
1	Remove CITY → STATUS	R1(CITY, STATUS) and R2(SSN, ENAME, PNUMBER, PNAME, PLOCATION, HOURS, CITY)
2	In R2, remove SSN → ENAME, CITY	R3(SSN, ENAME, CITY) and R4(SSN, PNUMBER, PNAME, PLOCATION, HOURS)
3	In R4, remove PNUMBER → PNAME, PLOCATION	R5(PNUMBER, PNAME, PLOCATION) and R6(SSN, PNUMBER, HOURS)

Final design

Relation	Attributes	Key
CITY_STATUS	CITY, STATUS	CITY
EMPLOYEE	SSN, ENAME, CITY	SSN
PART	PNUMBER, PNAME, PLOCATION	PNUMBER
WORKS	SSN, PNUMBER, HOURS	(SSN, PNUMBER)

Every determinant is now a super key, so the design is in BCNF. Each split was on a violating FD, so it is lossless, and every original FD is preserved inside one fragment, so it is dependency preserving.

Q12. Q6(a) Explain transactions and ACID properties with an example.

A **transaction** is a logical unit of work made of one or more database 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	How it is ensured
Atomicity	All operations happen or none do	Undo of incomplete work, rollback
Consistency	The database moves from one valid state to another	Constraint checking
Isolation	Concurrent transactions behave as if run alone	Concurrency control, locking, serializability
Durability	Committed changes survive failure	Logging, checkpoints, redo

If the transfer fails after debiting A, atomicity requires the debit to be undone; otherwise the total money in the bank would no longer be conserved.

Q13. Q6(b) Check the conflict serializability of S1, S2 and S3.

S1: r1(x); r3(x); w3(x); w1(x); r2(x)

Conflicts on x: r1(x) before w3(x) gives T1 → T3; w3(x) before w1(x) gives T3 → T1. The precedence graph has the cycle T1 ↔ T3, so **S1 is not conflict serializable**.

S2: r3(x); r2(x); r1(x); w3(x); w1(x)

Conflicts: r2(x) before w3(x) gives T2 → T3; r1(x) before w3(x) gives T1 → T3; r2(x) before w1(x) gives T2 → T1. The graph T2 → T1 → T3 is acyclic, so **S2 is conflict serializable with the equivalent serial order T2, T1, T3**.

S3: r1(x); w2(x); r3(x); w1(x); r2(x); w3(x)

Conflicts: r1(x) before w2(x) gives T1 → T2; w2(x) before w1(x) gives T2 → T1; w2(x) before r3(x) gives T2 → T3; r1(x) before w3(x) gives T1 → T3. The cycle T1 ↔ T2 means **S3 is not conflict serializable**.

Paper 2 — B26, Fourth Semester Backlog Examination 2026, CSE23306

Part A — 2-Mark Answers

Q1. Discuss primary key and foreign key of a relation with an example.

The **primary key** uniquely identifies each tuple in its own relation, cannot be NULL and is unique — SSN in EMPLOYEE. The **foreign key** is an attribute in one relation whose values must match a primary key value in another relation or be NULL — Dno in EMPLOYEE referencing DEPARTMENT(Dnumber). The primary key enforces entity integrity, the foreign key referential integrity.

Q2. Write the role of a Database Administrator.

The DBA defines the schemas and mappings, chooses storage structures and indexes, grants and revokes privileges, declares and monitors integrity constraints, tunes performance, plans backup and performs recovery, identifies user requirements, chooses the DBMS and maintains the data dictionary.

Q3. Define data independence in a database.

Data independence is the ability to change the schema at one level without changing the schema at the level above. **Logical** data independence allows the conceptual schema to change without altering external schemas or programs; **physical** data independence allows the internal schema to change without altering the conceptual schema.

Q4. Describe composite attributes.

A composite attribute can be divided into smaller sub-parts, each with independent meaning — Address = (House_no, Street, City, PIN), Name = (First_name, Last_name). In the relational model only the simple components are stored, and a strong entity set with composite attributes still needs only one table.

Q5. Compare strong entity with weak entity set, with an example.

A strong entity set has its own key and exists independently; a weak entity set has only a partial key, depends on an owner entity set, is drawn with a double rectangle, always participates totally in the identifying relationship, and maps to a table whose primary key is the owner's key plus the partial key. Example: LOAN is strong; PAYMENT, identified by Loan_no plus Payment_no, is weak.

Q6. List different types of degrees of relationships.

Unary (degree 1, recursive) — an employee supervises another employee; **binary** (degree 2) — a student enrolls in a course; **ternary** (degree 3) — a supplier supplies a part to a project; and generally **n-ary**, where n entity sets participate.

Q7. Define inner join and outer join.

An **inner join** keeps only the tuples that satisfy the join condition. An **outer join** additionally keeps the unmatched tuples, filling the missing attributes with NULL, and is of three kinds — left, right and full — depending on which side is preserved.

Q8. Explain the division operation with an example.

Division answers "for all" queries. R(X, Y) ÷ S(Y) gives every X value in R that is paired with every Y value in S. Example: WORKS_ON(Essn, Pno) ÷ ALL_PROJ(Pno) returns the employees who work on every project. It can be expressed with projection, Cartesian product and difference.

Q9. Difference between DROP and TRUNCATE in SQL.

DROP	TRUNCATE
Removes the table structure and its data	Removes all rows, keeps the structure
Table ceases to exist	Empty table remains
Removes indexes, constraints, privileges	Keeps indexes and constraints
Slower	Very fast, deallocates pages

Q10. R(A,B,C,D,E), F = {A→BC, CD→E}, decomposed into R1(A,B,C) and R2(C,D,E). Is the decomposition lossless?

No. The common attribute is C, and C+ = {C} under F, so C determines neither R1 nor R2. CD → E needs D as well, which lies only in R2. The natural join can therefore create spurious tuples, so the decomposition is lossy. A lossless split would need the common attribute to be a key of one fragment, for example R1(A,B,C) and R2(A,C,D,E).

Q11. Discuss different types of functional dependencies with examples.

Trivial — Y ⊆ X, as AB → A. **Non-trivial** — A → B. **Completely non-trivial** — AB → CD, disjoint sides. **Full** — {SID, CID} → Grade. **Partial** — SID → SName when the key is {SID, CID}. **Transitive** — EmpID → DeptID → DeptName. **Multivalued** — SID ↠ Hobby. **Join** — R = πR1(R) ⋈ πR2(R). **Inclusion** — R.A ⊆ S.B.

Q12. For R(A,B,C,D,E) with F = {A→B, BC→D, D→E}, compute the closures of A and BC.

A+ = {A, B}, because A → B and BC → D cannot fire without C. (BC)+ = {B, C, D, E}, because BC → D and then D → E. Two further closures: D+ = {D, E} and (ABC)+ = {A, B, C, D, E}, so ABC is a super key. Since A and C never appear on a right-hand side, {A, C} is the candidate key.

Q13. Differentiate conflict and view serializability.

Conflict Serializability	View Serializability
Order of conflicting operations must be preserved	Read-from relations and final writes must be preserved
Tested by precedence graph	Tested by three view-equivalence conditions
A subset of view-serializable schedules	A superset that also admits some blind-write schedules

Q14. Distinguish serial and non-serial schedules.

A **serial schedule** completes one transaction before the next starts, so it is always correct but allows no concurrency. A **non-serial schedule** interleaves the operations of several transactions, giving better throughput and response time but requiring serializability checking, because some interleavings produce lost updates or dirty reads.

Q15. Explain the requirements of recovery in a database environment.

Recovery must guarantee **atomicity** — the effects of uncommitted transactions are undone — and **durability** — the effects of committed transactions are redone. It must restore the database to the most recent consistent state after a transaction, system or media failure, and it must do so using information that survives the failure: the transaction log, checkpoints and backups.

Part B — 7-Mark Answers

Q1. Q2(a) Compare Relational and Hierarchical models with examples.

Relational Model	Hierarchical Model
Data as tables of rows and columns	Data as a tree of records
Relationships by common values (foreign keys)	Relationships by parent–child pointers
Any M:N relationship possible with an extra table	M:N relationships cannot be represented directly
No pointer maintenance	Pointer structure must be navigated by the programmer
Declarative querying with SQL	Navigational access, one record at a time
Normalization theory applies	Rigid structure causes insertion, deletion and update anomalies
Example: STUDENT(SID, Name, Dept)	Example: DEPARTMENT at the root, EMPLOYEE as children, DEPENDENT as grandchildren

The hierarchical model is fast for one-to-many traversal along the tree, but a child cannot exist without its parent and a record cannot have two parents, which is why it fails for real-world many-to-many data.

Q2. Q2(b) Illustrate the advantages of a DBMS over a file-management system, and justify why data independence is necessary.

Advantages

Advantage	How the DBMS provides it
Controls redundancy	One integrated database, each fact stored once
Ensures consistency	Constraints enforced centrally
Shares data	Many applications read the same data
Enforces integrity	Keys, domain and referential constraints
Provides security	Views and grant/revoke privileges
Supports concurrent access	Transactions with isolation and locking
Provides recovery	Logs, checkpoints, undo and redo
Reduces program–data dependence	Schema held separately in the data dictionary

Why data independence is necessary

Without it, every change in the way data is stored or organised forces every application program to be rewritten, because the file layout is embedded in the code. Data independence lets the DBA reorganise files, add indexes or move to new storage without touching the programs that use the data, and lets the logical schema grow — a new attribute or a new relation — without breaking existing views and reports. It is what makes a database maintainable over time.

Q3. Q2(c) Explain data atomicity and isolation with data inconsistency, with examples.

Atomicity

A transaction is all-or-nothing. If it fails part-way, every change it made is undone. Example: transferring ₹1,000 from A to B debits A and then fails before crediting B. Without atomicity, ₹1,000 would vanish; with it, the debit is rolled back and the total balance is unchanged.

Isolation

Concurrent transactions must not see each other's uncommitted work; each must behave as if it were running alone. Example: T1 reads X and is about to update it while T2 also reads the same X — without isolation both compute from the old value and one update is lost.

Data inconsistency

Inconsistency means two stored copies of the same fact disagree, or a constraint that should hold across rows no longer does. It arises from redundancy without control, from partial updates, and from unisolated concurrency.

Cause	Inconsistency produced	Prevented by
Redundant storage, one copy updated	Two different values for one fact	Normalization and controlled update
Failure after a partial update	Money debited but not credited	Atomicity and rollback
Lost update	One transaction's write overwrites another's	Isolation, locking
Dirty read	Reading a value later rolled back	Isolation, no uncommitted reads

Q4. Q3(a) Illustrate the different types of keys with examples.

Key	Definition	Example
Super key	Any set that identifies a tuple uniquely	{SID}, {SID, Name}
Candidate key	A minimal super key	{SID}, {RollNo}, {Email}
Primary key	The chosen candidate key	{SID}
Alternate key	Candidate keys not chosen	{RollNo}, {Email}
Foreign key	References another relation's primary key	DeptID in STUDENT
Composite key	Two or more attributes together	(OrderID, ProductID) in ORDER_ITEM
Partial key	Discriminator of a weak entity	Dependent_name

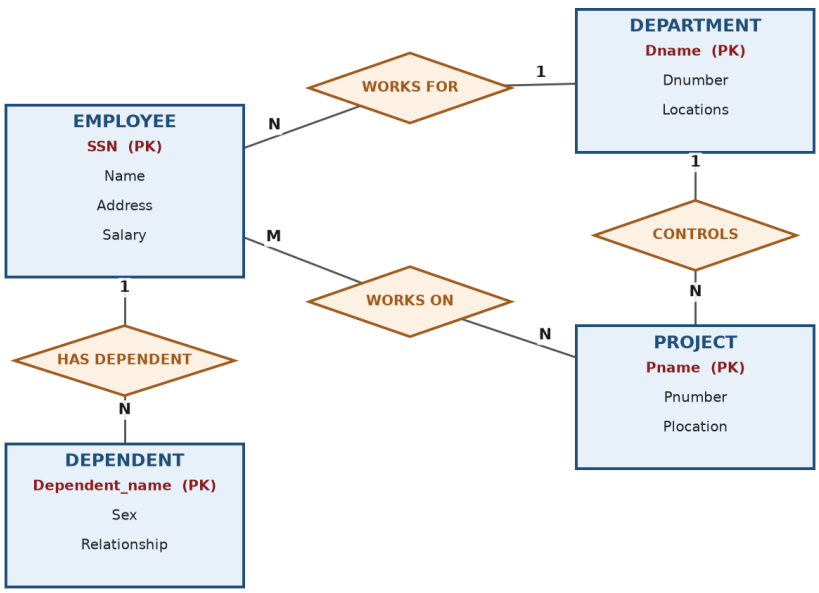
The primary key enforces entity integrity, the foreign key referential integrity, and the alternate keys prevent duplicate entities while giving extra unique access paths.

Q5. Q3(b) Describe cardinality ratios and participation constraints with an ER diagram.

Ratio	Meaning	Example
1:1	Each entity relates to at most one of the other	EMPLOYEE MANAGES DEPARTMENT
1:N	One A relates to many B	DEPARTMENT has many EMPLOYEEs
N:1	Many A relate to one B	Many EMPLOYEEs work for one DEPARTMENT
M:N	Many relate to many	STUDENT enrolls in many COURSEs

The **participation constraint** fixes the minimum: **total** participation, drawn with a double line, means every entity must take part; **partial**, drawn with a single line, means only some need to. Every employee must belong to a department (total on the EMPLOYEE side), but only one employee manages a department (partial on the DEPARTMENT side).

COMPANY Database — ER Diagram

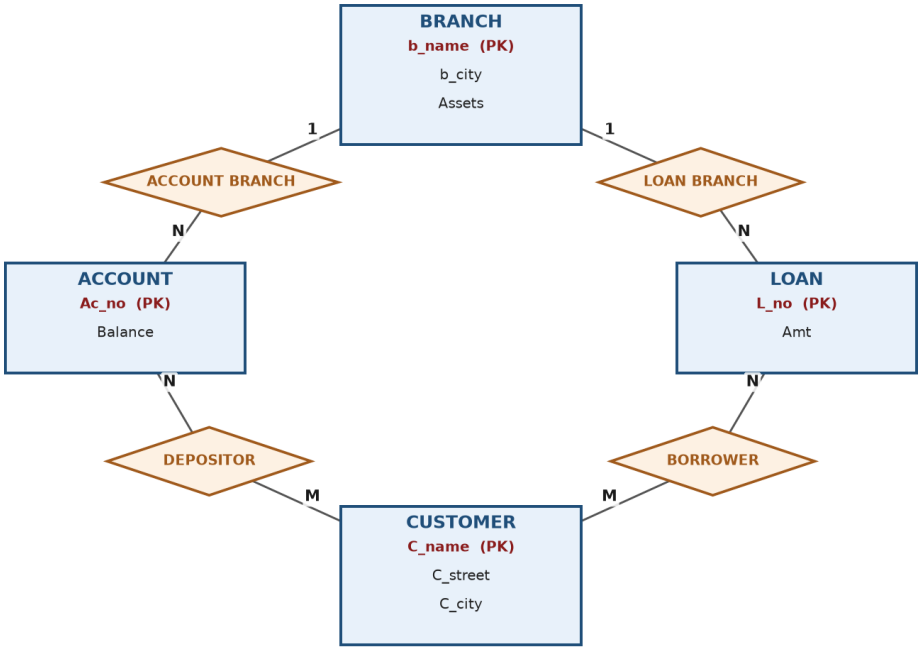


WORKS_ON carries the descriptive attribute Hours. DEPENDENT is a weak entity (partial key Dependent_name).

Q6. Q3(c) Map the given bank ER model to the relational model.

Entities and attributes: Account(Ac_no, Balance), Branch(b_name, b_city, Assets), Customer(C_name, C_street, C_city), Loan(L_no, Amt). Relationships: Account branch, Depositor, Loan branch, Borrower.

BANKING Database — ER Diagram (repeated in PYQ 2025 and 2026)



Maps to 6 tables: Account, Branch, Loan, Borrower, Customer, Depositor.

Table	Attributes	Key and constraints
Branch	b_name, b_city, Assets	PK: b_name
Account	Ac_no, Balance, b_name	PK: Ac_no; FK: b_name to Branch
Loan	L_no, Amt, b_name	PK: L_no; FK: b_name to Branch
Customer	C_name, C_street, C_city	PK: C_name
Depositor	C_name, Ac_no	PK: (C_name, Ac_no); FKs to Customer and Account
Borrower	C_name, L_no	PK: (C_name, L_no); FKs to Customer and Loan

Six tables in total. The two 1:N relationships are merged into the many side, and the two M:N relationships each become a separate table with a composite key.

Q7. Q4(a) Retrieve the tuples returned by the given natural join and outer join expressions.

Relations: P(X, Y, Z) = {(X1,Y1,Z1), (X1,Y1,Z2), (X2,Y2,Z2), (X2,Y4,Z4)}; Q(X, Y, T) = {(X2,Y1,2), (X1,Y2,5), (X1,Y1,6), (X3,Y1,1)}; R(Y, V) = {(Y1,V1), (Y3,V2), (Y2,V3), (Y2,V2)}.

(1) $\sigma_{P.Y = Q.Y \wedge Q.X = P.X}(P \bowtie Q)$

The natural join matches on both X and Y, so only (X1, Y1) survives; the repeated selection removes nothing further.

X	Y	Z	T
X1	Y1	Z1	6
X1	Y1	Z2	6

(2) $\sigma_{R.Y = Q.Y}(R \bowtie Q)$

The natural join on Y pairs every R tuple with every Q tuple of the same Y.

Y	V	X	T
Y1	V1	X2	2
Y1	V1	X1	6
Y1	V1	X3	1
Y2	V3	X1	5
Y2	V2	X1	5

(3) Left outer join of Q and R

Q LEFT OUTER JOIN R preserves every Q tuple. All Q values of Y — Y1 and Y2 — exist in R, so no Q row is lost and no NULL appears; the unmatched R row (Y3, V2) is dropped because R is not the preserved side.

X	Y	T	V
X2	Y1	2	V1
X1	Y2	5	V3
X1	Y2	5	V2
X1	Y1	6	V1
X3	Y1	1	V1

If R were the preserved side, R LEFT OUTER JOIN Q would keep (Y3, V2) with NULL for X and T.

Q8. Q4(b) State and explain the DDL, DML and DQL commands with examples.

DDL

Statement	Syntax	Purpose
CREATE TABLE	CREATE TABLE Emp (EmpID INT PRIMARY KEY, Name VARCHAR(30), Salary NUMBER(10,2), DeptID INT REFERENCES Dept(DeptID));	Creates a table with constraints
ALTER TABLE	ALTER TABLE Emp ADD (Phone VARCHAR(15));	Adds or drops a column
DROP TABLE	DROP TABLE Emp;	Removes structure and data
TRUNCATE	TRUNCATE TABLE Emp;	Removes all rows, keeps structure
CREATE INDEX	CREATE INDEX idx ON Emp(Name);	Builds an index
CREATE VIEW	CREATE VIEW CSE AS SELECT * FROM Emp WHERE DeptID = 10;	Defines a virtual table

DML

Statement	Syntax	Purpose
INSERT	INSERT INTO Dept VALUES (10,'HR');	Adds rows
UPDATE	UPDATE Emp SET Salary = Salary * 1.10;	Modifies rows
DELETE	DELETE FROM Emp WHERE Salary < 50000;	Removes rows

DQL

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

Example: SELECT Name, Salary FROM Emp WHERE Salary > 50000 ORDER BY Salary DESC;

DDL changes the schema and updates the data dictionary; DML changes the instance and can be rolled back; DQL only reads.

Q9. Q4(c) Solve the relational algebra queries on Employee and Department.

Schema: Employee(eid, ename, salary, dept_id); Department(dept_id, dept_name, location).

(i) Primary and foreign keys

Employee: PK eid, FK dept_id referencing Department(dept_id). Department: PK dept_id.

(ii) Employees with salary greater than 40,000

$\sigma_{salary > 40000}(Employee)$

(iii) Employees working in the HR department

$\pi_{ename}(\sigma_{dept_name = 'HR'}(Employee \bowtie Department))$

(iv) Departments located in Delhi

$\pi_{dept_name}(\sigma_{location = 'Delhi'}(Department))$ Or simply $\sigma_{location = 'Delhi'}(Department)$ if all department attributes are wanted.

Q10. Q5(a) Check whether F1 = {A→BC, B→DE, E→AB} and F2 = {A→BCF, B→DE, E→AB} on R(A,B,C,D,E,F) are equivalent.

Under F1, A+ = {A, B, C, D, E} — A → BC, B → DE, and E → AB adds nothing new. F is not in A+, so A → F is not implied and F1 does not imply F2. Under F2, A+ contains F, and A → BCF implies A → BC, so F2 does imply F1. **The sets are not equivalent; F2 is strictly stronger than F1.**

Q11. Q5(b) Explain the need for normalization and the normal forms up to BCNF with examples.

The need

An un-normalized relation repeats facts, which causes three anomalies: insertion is blocked because a new fact needs an unrelated fact to exist, update must touch many rows and can leave copies disagreeing, and deletion can destroy unrelated facts. Normalization stores each fact once, in the relation determined by its determinant.

The forms

Form	Condition	Removes
1NF	Atomic values, no repeating groups	Multivalued cells
2NF	1NF plus no partial dependency	Partial dependency on part of a key
3NF	2NF plus no transitive dependency of a non-prime attribute	Transitive dependency
BCNF	Every determinant of a non-trivial FD is a super key	Remaining key anomalies

Example

ENROLL(StudentID, CourseID, StudentName, CourseName, Grade) with SID → SName, CID → CName and {SID, CID} → Grade. The partial dependencies SID → SName and CID → CName violate 2NF; decomposing gives STUDENT(SID, SName), COURSE(CID, CName) and ENROLL(SID, CID, Grade). If DeptID → DeptName were present, EmpID → DeptID → DeptName would violate 3NF and require a DEPARTMENT relation. A relation in 3NF but not BCNF is R(Student, Course, Instructor) with Instructor → Course, decomposed into (Instructor, Course) and (Student, Instructor).

Q12. Q6(a) Describe a transaction and explain transaction states with a state diagram.

A transaction is a logical unit of database work consisting of one or more operations that must be executed as a whole.

State	Meaning
Active	Executing read, write and computation operations
Partially committed	The final statement has executed but the changes are not yet secured
Committed	Completed successfully; changes are permanent
Failed	Cannot proceed because of an error, deadlock or crash
Aborted	Rolled back; incomplete effects removed
Terminated	Processing finished after commit or abort

The life cycle is Active → Partially committed → Committed, or Active → Failed → Aborted. An aborted transaction may be restarted if the failure was not caused by a logic error. A diagram is drawn with Active at the left, Partially committed and Committed above, Failed and Aborted below, and Terminated at the right.

Q13. Q6(b) Check conflict serializability using a precedence graph.

S1: r1(x); r1(y); w1(x); w2(x); r2(y)

Conflicts: r1(x) before w2(x) gives T1 → T2; w1(x) before w2(x) gives T1 → T2; r1(y) and r2(y) are two reads and do not conflict. The graph has only T1 → T2 and is acyclic, so **S1 is conflict serializable with the serial order T1, T2**.

S2: r2(x); r1(y); w1(x); w2(y)

Conflicts: r2(x) before w1(x) gives T2 → T1; r1(y) before w2(y) gives T1 → T2. The cycle T1 ↔ T2 means **S2 is not conflict serializable**. Note: the scanned page is rotated and the subscripts are small, so verify the schedule against the original before finalising.

Paper 3 — R24, Fourth Semester Regular Examination 2025, CS242

Part A — 2-Mark Answers

Q1. Explain the purpose and role of a data dictionary.

A data dictionary, or system catalog, is the repository of metadata: relation names, attributes, data types, sizes, keys, constraints, indexes, users and privileges. It lets the DBMS parse and validate queries, enforce constraints and check authorisation without reading the data, and it is the DBA's documentation of the schema.

Q2. Explain the meaning of a composite attribute with an example.

An attribute that can be divided into sub-parts with independent meaning — Address = (House_no, Street, City, PIN), or Name = (First_name, Last_name). Only the simple components are stored in the relational model.

Q3. Give an example of a one-to-many relationship.

DEPARTMENT — WORKS_FOR — EMPLOYEE: one department can have many employees, but each employee works for exactly one department. The cardinality is 1:N from DEPARTMENT to EMPLOYEE, with total participation on the EMPLOYEE side.

Q4. Compare and contrast Generalization, Specialization and Aggregation.

Generalization is bottom-up — common attributes of several entity sets are combined into a superclass. **Specialization** is top-down — a superclass is divided into subclasses with their own attributes. **Aggregation** treats a relationship set together with its entity sets as a single higher-level entity, so that it can participate in another relationship, as when an employee manages the works-on of another employee on a project.

Q5. Illustrate ER and EER models in the context of a database.

The ER model describes data as entity sets, attributes, relationships, cardinality ratios and participation constraints, and is drawn with rectangles, ellipses and diamonds. The EER model extends it with subclass and superclass, generalization and specialization, attribute inheritance, category or union types, aggregation, and the disjoint/overlapping and total/partial constraints on specialization. EER is used when the application has a genuine class hierarchy.

Q6. Explain degree of a relationship with an example.

The degree is the number of entity sets participating in a relationship. Unary (degree 1) — an employee supervises another employee; binary (degree 2) — a student enrolls in a course; ternary (degree 3) — a supplier supplies a part to a project.

Q7. Analyse how relational algebra differs from SQL in expressiveness and execution.

Relational algebra is procedural and fixes the order of operations; SQL is declarative and adds aggregation, grouping, ordering, NULL handling, string matching and subqueries. In execution, the DBMS parses the SQL, translates it into an internal relational-algebra tree and then optimises that tree, so algebra is the execution language while SQL is the user language.

Q8. Explain the difference between JOIN and Cartesian product.

JOIN	Cartesian Product
Combines only related tuples	Combines every tuple with every tuple
Output usually smaller than the product	Output size is the product of cardinalities
Produces meaningful pairs	Most pairs are meaningless
Written R ⋈ S	Written R × S

Q9. Explain union compatibility in relational algebra operations.

Two relations are union compatible when they have the same degree and corresponding attributes come from the same domains. Union, intersection and difference are defined only for union-compatible relations; attribute names need not match, only types and order.

Q10. Identify anomalies that occur in an un-normalized relation.

Insertion anomaly — a fact cannot be recorded without an unrelated fact, such as a new course needing an enrolled student. **Update anomaly** — the same fact appears in many rows, so one change needs many updates and a missed row leaves inconsistency. **Deletion anomaly** — deleting one fact destroys another, such as deleting the last enrollment of a course removing the course itself.

Q11. Explain minimizing the left-hand side of a functional dependency.

To test whether A is extraneous in XA → Y, remove A and compute the closure of X under F. If Y is still in X+, then A was extraneous and the FD reduces to X → Y. Example: with AB → C and A → B, removing B leaves A, and A+ contains B and hence C, so AB → C becomes A → C.

Q12. Identify and eliminate the redundant FD from F = {A→B, B→C, A→C}.

Remove A → C temporarily and compute A+ with the remaining dependencies: A → B gives B, and B → C gives C, so A+ = {A, B, C} contains C. Therefore A → C is redundant and the cover reduces to **{A → B, B → C}**.

Q13. Explain view serializability with a suitable example.

A schedule is view serializable if it is view equivalent to some serial schedule — that is, the same transaction reads the initial value of each item, the same read-from relationships hold, and the same transaction performs the final write. Example: W1(X), W2(X), W1(X) is not conflict serializable because its precedence graph has a cycle, but it is view serializable because T1 performs the final write in both the schedule and the serial order T1, T2.

Q14. Analyse S1: R1(A), R2(B), W1(B), W2(A) for conflict serializability using a precedence graph.

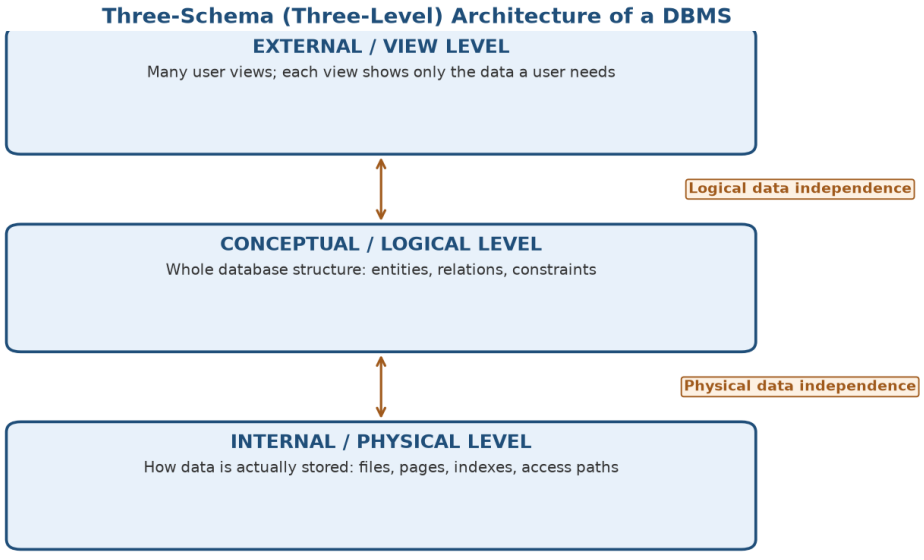
Conflicts: R2(B) before W1(B) gives T2 → T1; R1(A) before W2(A) gives T1 → T2. The precedence graph contains the cycle T1 ↔ T2, so **S1 is not conflict serializable**.

Q15. List types of failures that can occur in a database system.

Transaction failure — logical error, constraint violation or a transaction chosen as the deadlock victim; **system failure** — power loss, operating-system crash or DBMS crash; **media failure** — disk crash or corrupted database file; and **communication failure** in a distributed system. The first needs undo, the second needs undo and redo, the third needs restore from backup plus logs.

Part B — 7-Mark Answers

Q1. Q2(a) Draw and explain the three-schema architecture and state the importance of data abstraction.



External/Conceptual mapping and Conceptual/Internal mapping isolate each level from changes in the level below.

The three levels

External — many user views, each showing only the data that user needs. **Conceptual** — the complete logical description of entities, attributes, relationships and constraints. **Internal** — the physical description of files, record layouts, indexes and access paths. The external/conceptual and

conceptual/internal mappings connect them.

Importance of data abstraction

Data abstraction hides the complexity of the lower levels from the higher ones, so users deal with a simple view instead of storage detail. This makes the database easier to understand and to use, protects application programs from storage changes, allows the same data to be shown differently to different users, and improves security because a user can be restricted to a view that exposes only permitted data.

Q2. Q2(b) Explain the file-processing system and its limitations, and how a DBMS addresses them.

What it is

In a file-processing system each application keeps its own files and its own programs to read and update them; there is no central control over the data.

Limitations

Limitation	Consequence
Data redundancy	Same fact stored in several files, wasting space
Data inconsistency	Copies disagree after a partial update
Program–data dependence	Layout embedded in code; any change forces rewrites
Difficulty of access	Only fixed reports; new questions need new programs
No concurrency control	Lost updates and dirty reads
Weak atomicity	A half-done update leaves inconsistent data
Poor security	Access control per file, not per record or field
Limited sharing	Data isolated by department

How a DBMS addresses them

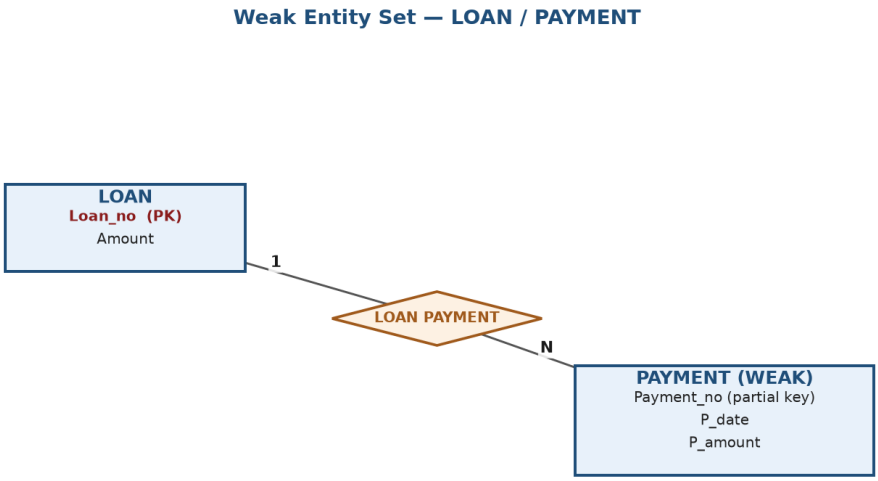
A single integrated database described by a schema in the data dictionary stores each fact once. The three-schema architecture and data independence separate programs from storage. SQL provides ad hoc querying. Constraints enforce integrity centrally. Transactions with ACID properties provide concurrency control and recovery. Views and privileges give fine-grained security, and the same data can be shared by all authorised applications.

Q3. Q2(c) Explain weak entity sets, with an example such as Student–Dependent.

Definition

A weak entity set has no key attribute of its own, exists only in association with an owner entity set, participates totally in the identifying relationship, and is identified by the owner's primary key together with its own partial key.

Strong Entity Set	Weak Entity Set
Own key attribute	Only a partial key
Independent existence	Depends on the owner
Single rectangle	Double rectangle
Participation may be partial	Total participation in the identifying relationship
Maps to a table with its own primary key	Maps to a table keyed by owner key plus partial key



PAYMENT has no key of its own; it is identified by Loan_no + Payment_no and exists only with total participation in LOAN_PAYMENT.

Identifying relationship and partial key

The relationship linking the weak set to its owner is the **identifying relationship**, drawn with a double diamond, and it always has total participation on the weak side. The **partial key** distinguishes entities belonging to the same owner: Payment_no distinguishes the payments of one loan, but repeats across loans.

Student–Dependent example

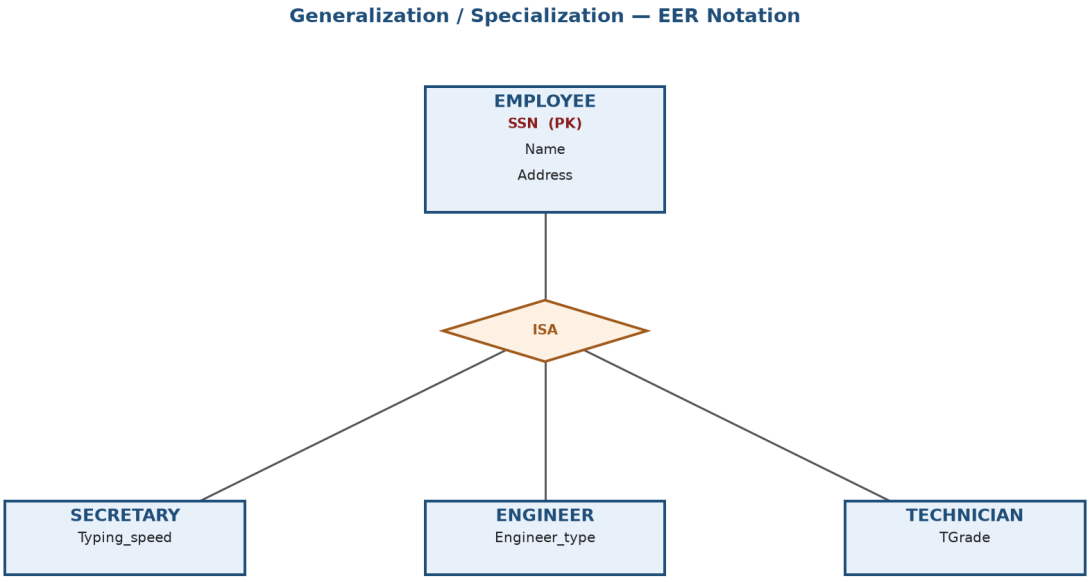
DEPENDENT(Dependent_name, Sex, Birthdate, Relationship) is weak with respect to EMPLOYEE. Dependent_name is only a partial key, because two employees may each have a dependent named Rahul. The full key is (SSN, Dependent_name), and the mapping gives EMPLOYEE(SSN, Name, …) and DEPENDENT(SSN, Dependent_name, Sex, Birthdate, Relationship). The Building–Apartment example works the same way, with Apartment_no as the partial key within a Building.

Q4. Q3(a) Classify database keys and explain how each contributes to data integrity.

Key	Definition	Contribution to integrity
Super key	Any identifying attribute set	Provides the pool of identifiers
Candidate key	A minimal super key	Guarantees a minimal identifier exists
Primary key	The chosen candidate key	Entity integrity — unique and never NULL
Alternate key	Candidate keys not chosen	Prevents duplicate entities, gives extra access paths
Foreign key	References another relation's key	Referential integrity — no dangling references
Composite key	Two or more attributes together	Identifies rows where no single attribute is unique
Partial key	Discriminator of a weak entity	Identifies the weak entity within its owner

Together they ensure that every row is identifiable, that no two rows represent the same entity, and that every relationship points to a row that actually exists.

Q5. Q3(b) Design relational tables from an ER diagram involving specialization.



ISA is the specialization/generalization relationship. Attributes are inherited downward; the constraint may be disjoint or overlapping, total or partial.

Strategy 1 — single table

EMPLOYEE(SSN, Name, Address, Typing_speed, Engineer_type, TGrade, Emp_type). The type attribute records which subclass the row belongs to. This is compact and suits an overlapping specialization, but it produces many NULLs and cannot enforce disjointness without a CHECK constraint.

Strategy 2 — one table per subclass, superclass key only

EMPLOYEE(SSN, Name, Address); SECRETARY(SSN, Typing_speed); ENGINEER(SSN, Engineer_type); TECHNICIAN(SSN, TGrade). Each subclass key is also a foreign key to EMPLOYEE. This works for any constraint and avoids NULLs, but rebuilding a full entity needs a join.

Strategy 3 — one table per subclass with all attributes

SECRETARY(SSN, Name, Address, Typing_speed), and similarly for the other subclasses. No join is needed, but the common attributes are repeated and a total constraint cannot be enforced declaratively.

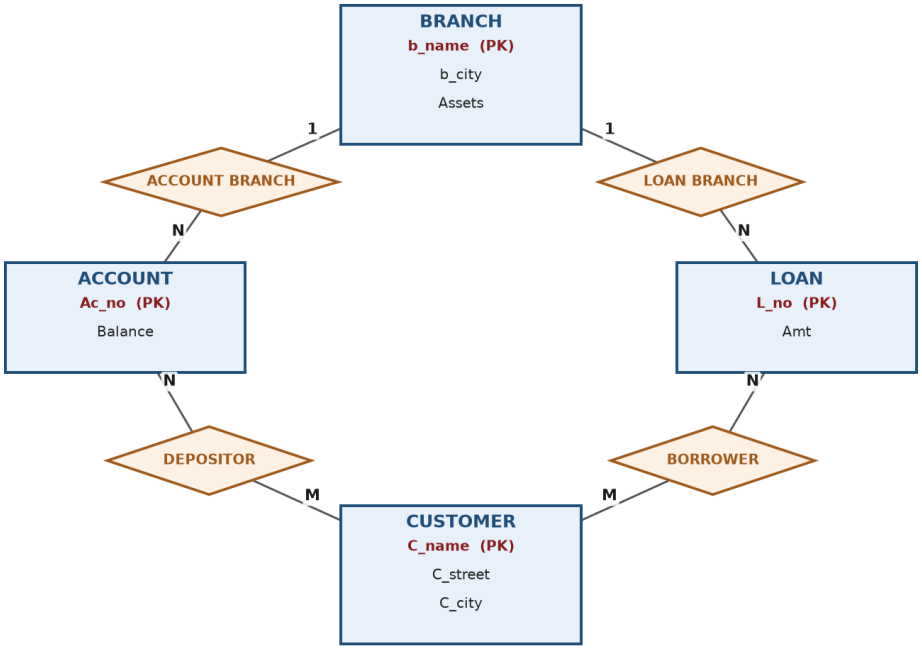
Choosing

Situation	Best strategy
Total and disjoint, few subclass attributes	Single table
Many subclass attributes, any constraint	Table per subclass with superclass key
Queries read one subclass only	Table per subclass with all attributes
Overlapping specialization	Single table, or subclass tables with the key

Q6. Q3(c) Convert Figure-1 ER diagram into relational tables, and draw the company ER diagram.

Figure-1 — the banking ER diagram

BANKING Database — ER Diagram (repeated in PYQ 2025 and 2026)

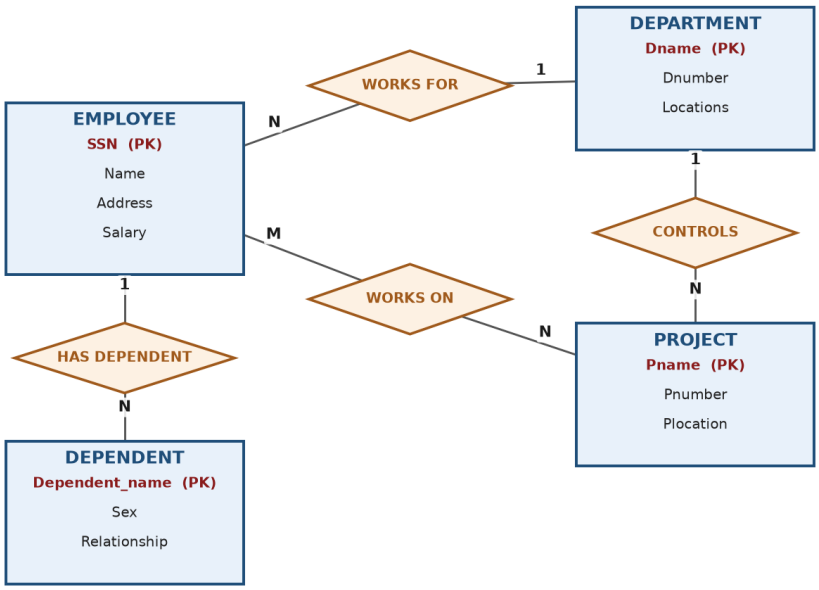


Maps to 6 tables: Account, Branch, Loan, Borrower, Customer, Depositor.

Table	Attributes	Key and constraints
Branch	b_name, b_city, Assets	PK: b_name
Account	Ac_no, Balance, b_name	PK: Ac_no; FK: b_name to Branch
Loan	L_no, Amt, b_name	PK: L_no; FK: b_name to Branch
Customer	C_name, C_street, C_city	PK: C_name
Depositor	C_name, Ac_no	PK: (C_name, Ac_no); both FKs
Borrower	C_name, L_no	PK: (C_name, L_no); both FKs

The company scenario — ER diagram

COMPANY Database — ER Diagram



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

Design decisions

DEPARTMENT has a multivalued attribute Locations, which becomes a separate table. MANAGES is 1:1 and carries Start_date. WORKS_FOR is N:1 with total participation on the EMPLOYEE side. CONTROLS is 1:N from DEPARTMENT to PROJECT. WORKS_ON is M:N and carries Hours. DEPENDENT is weak, identified by HAS_DEPENDENT with Dependent_name as partial key.

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

Q7. Q4(a) Define join and compare natural, theta and equi join with examples.

A **join** combines tuples from two relations that satisfy a matching condition, and it is the operation that reconstructs information distributed across normalized tables.

Natural Join	Equi Join	Theta Join
Joins on all same-named attributes	Equality condition written explicitly	Any comparison condition
Condition implicit	Uses =	May use <, >, ≤, ≥, ≠
Common column appears once	Join column appears twice	Join column appears twice
R ⋈ S	R ⋈R.A=S.B S	R ⋈θ S

With EMPLOYEE(EmpID, EmpName, DeptID) and DEPARTMENT(DeptID, DeptName), the natural join matches on DeptID and keeps one copy. The equi join produces the same rows with DeptID twice. A theta join is required for range matching, as in EMPLOYEE $\bowtie_{\text{Salary} \geq \text{MinSalary} \wedge \text{Salary} \leq \text{MaxSalary}}$ SALARY_GRADE, which assigns each employee a salary grade.

Q8. Q4(b) Write the SQL queries on the Department and Employee tables.

DeptID	DeptName
10	HR
20	IT
30	Finance

EmpID	Name	Age	Salary	DeptID
1	John	28	50000	10
2	Maria	32	62000	20
3	Ahmed	24	45000	10
4	Priya	29	67000	30

(i) Create the tables and insert the data

CREATE TABLE Department (DeptID INT PRIMARY KEY, DeptName VARCHAR(30) NOT NULL);

CREATE TABLE Employee (EmpID INT PRIMARY KEY, Name VARCHAR(30) NOT NULL, Age INT, Salary NUMBER(10,2), DeptID INT REFERENCES Department(DeptID));

INSERT INTO Department VALUES (10,'HR'), (20,'IT'), (30,'Finance');

INSERT INTO Employee VALUES (1,'John',28,50000,10), (2,'Maria',32,62000,20), (3,'Ahmed',24,45000,10), (4,'Priya',29,67000,30);

(ii) Names and salaries of employees in IT

SELECT E.Name, E.Salary FROM Employee E JOIN Department D ON E.DeptID = D.DeptID WHERE D.DeptName = 'IT';

Result: Maria, 62000.

(iii) Increase every salary by 10 percent

UPDATE Employee SET Salary = Salary * 1.10;

(iv) Delete employees with salary below 50,000

DELETE FROM Employee WHERE Salary < 50000;

This removes Ahmed.

(v) Average employee salary

SELECT AVG(Salary) AS Average_Salary FROM Employee;

Result: 56,000 before the update.

(vi) Employees whose name starts with M

SELECT * FROM Employee WHERE Name LIKE 'M%';

Result: Maria.

(vii) Employees ordered by salary, highest first

SELECT * FROM Employee ORDER BY Salary DESC;

Result: Priya 67000, Maria 62000, John 50000, Ahmed 45000.

Q9. Q4(c) Solve the relational algebra queries on the passenger, agency, flight and booking schema.

Schema: passenger(pid, pname, pgender, pcity); agency(aid, aname, acity); flight(fid, fname, fdate, ftime, source, destination); booking(pid, aid, fid, fdate).

(i) Flight IDs travelling to New Delhi

$\pi_{fid}(\sigma_{destination = 'New\ Delhi'}(flight))$

(ii) Flight IDs and names from Chennai to New Delhi

$\pi_{fid,\ fname}(\sigma_{source = 'Chennai' \wedge destination = 'New\ Delhi'}(flight))$

(iii) Agency IDs that booked flight UA2321

$\pi_{aid}(\sigma_{fid = 'UA2321'}(booking))$

(iv) Passenger names with bookings on UA2321

$\pi_{pname}(passenger \bowtie \sigma_{fid = 'UA2321'}(booking))$

(v) Passenger names and cities who booked through JET Agency

$\pi_{pname,\ pcity}(passenger \bowtie booking \bowtie \sigma_{aname = 'JET\ Agency'}(agency))$ The selection is pushed down to booking and agency first, which is cheaper than joining the full relations.

Q10. Q5(a) For R(A,B,C,D,E,F,G,H,I,J,K) with F = {A→BC, A→DE, B→F, F→GH, D→IJ, K→B}, find all candidate keys and normalise to BCNF.

Candidate keys

The attributes never appearing on a right-hand side are A, E and K, so each must be in every candidate key.

Set	Closure	Reason
A+	A, B, C, D, E, F, G, H, I, J	A → BC, A → DE, B → F, F → GH, D → IJ
K+	K, B, F, G, H	K → B, B → F, F → GH
(A,K)+	All eleven attributes	A gives B to J, K gives B

E is already determined by A, so it need not be added. **The only candidate key is {A, K}.**

Is R in BCNF?

No. A → B is non-trivial and A is not a super key, so the relation violates BCNF; so do B → F, F → GH, D → IJ and K → B.

Decomposition

Relation	Attributes	Key	FD held
R1	A, B, C, D, E	A	A → BC, A → DE
R2	B, F	B	B → F
R3	F, G, H	F	F → GH
R4	D, I, J	D	D → IJ
R5	K, B	K	K → B
R6	A, K	(A, K)	The global key

Verification

Every determinant in R1 to R6 is a super key of its own relation, so the design is in BCNF. Each split was made on a violating FD, so the decomposition is lossless. Every FD of F appears in exactly one fragment, so it is dependency preserving. R6 is added because the candidate key {A, K} is not contained in any other fragment, and preserving the key is what guarantees losslessness for the

whole decomposition.