DATABASE ENGINEERING

Unit 3

Relational Algebra, Relational Calculus and SQL

Section A — 34 short-answer questions (2 marks each)

Section B — 14 long-answer questions (7 marks each), with worked queries

Merges the Unit III syllabus topics, the model question bank and every Unit III question from PYQ R25, B26 and R24

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.

Section A — 2-Mark Questions

Q1. Define relational algebra.

Relational algebra is a procedural query language in which each operation takes one or two relations as input and produces a new relation as output. It provides the formal foundation of SQL, and because every operator is closed over relations, operations can be nested to build complex queries.

Q2. Outline the Selection operation with its syntax.

PYQ R25 Q1(g)

Selection returns the tuples of a relation that satisfy a given condition. It works on rows and does not change the degree of the relation.

$\sigma_{condition}(R)$ — for example $\sigma_{Salary\ >\ 50000}(EMPLOYEE)$, or $\sigma_{Dept='CSE'\ \wedge\ CGPA>8}(STUDENT)$. The condition may use $=, \neq, <, >, \leq, \geq$ combined with AND, OR and NOT.

Q3. Outline the Projection operation with its syntax.

PYQ R25 Q1(g)

Projection returns only the specified attributes of a relation, discarding the rest. It works on columns and may reduce the degree.

$\pi_{A1, A2, ..., An}(R)$ — for example $\pi_{Name, Salary}(EMPLOYEE)$. Duplicate tuples created by the projection are removed, because a relation is a set.

Q4. What is the RENAME operator? Compare it with the SQL SORT command.

PYQ R25 Q1(i)

The **RENAME** operator ρ renames a relation or its attributes, so that the same relation can be used twice in one expression without ambiguity.

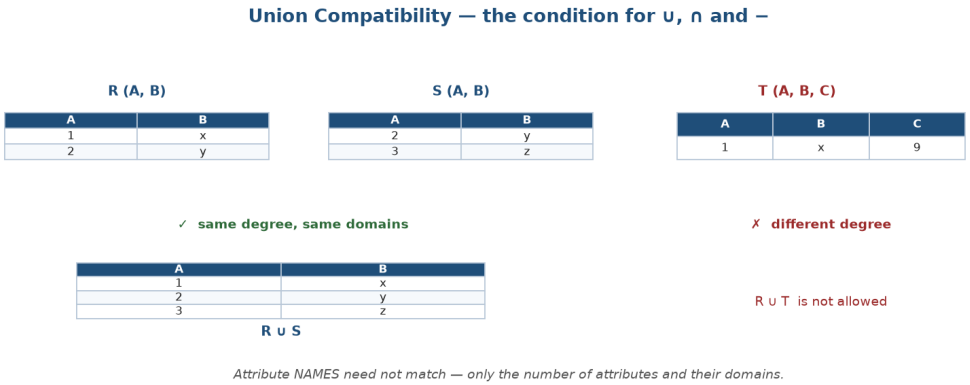
$\rho_S(R)$ renames the relation, and $\rho_{S(B1,B2)}(R)$ renames both the relation and its attributes. The SQL **SORT** command — that is ORDER BY — does something completely different: it does not change names but arranges the output rows in ascending or descending order of a column, as in SELECT * FROM Employee ORDER BY Salary DESC. RENAME changes the schema; ORDER BY changes only the presentation of the result.

Q5. Differentiate JOIN from Cartesian product.

PYQ R24 Q1(h)

JOIN	Cartesian Product
Combines related tuples only, using a condition	Combines every tuple of one relation with every tuple of the other
Output is usually smaller than the product	Output size is the product of the two cardinalities
Meaningful pairs are produced	Most pairs are meaningless
Written R ⋈ S or R ⋈θ S	Written R × S
Degree is less than the sum if common attributes are merged	Degree is the sum of the two degrees

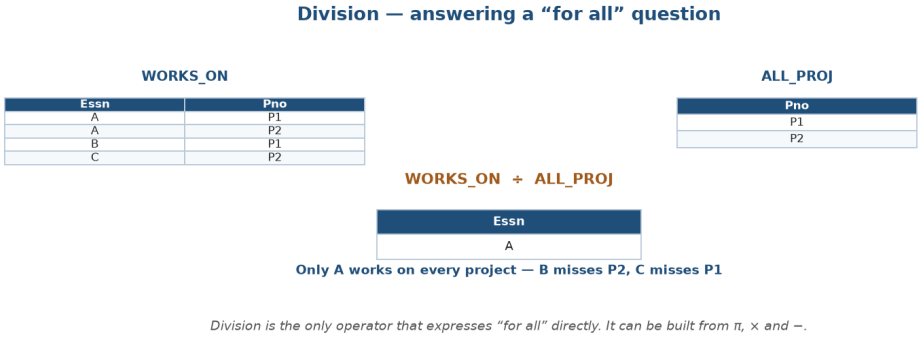
Q6. What is union compatibility?



PYQ R24 Q1(i)

Two relations are **union compatible** when they have the same number of attributes (same degree) and the corresponding attributes are drawn from the same domains. Union, intersection and difference are defined only for union-compatible relations. The names of the attributes need not match, only their types and order.

Q7. Explain the division operation with an example.



PYQ B26 Q1(h)

Division answers "for all" queries. If R(X, Y) ÷ S(Y) is performed, the result contains every value of X in R that is paired with **every** value of Y in S. Example: if WORKS_ON(Essn, Pno) and ALL_PROJ(Pno) contain all project numbers, then WORKS_ON ÷ ALL_PROJ gives the employees who work on every project. Division can be expressed using projection, Cartesian product and difference.

Q8. Define inner join and outer join.

PYQ B26 Q1(g)

An **inner join** returns only the tuples that satisfy the join condition, so unmatched tuples from either relation are discarded. An **outer join** additionally keeps the unmatched tuples, filling the missing attributes with NULL. Outer joins are of three kinds — left, right and full — depending on which side's unmatched tuples are preserved.

Q9. Differentiate natural join, theta join and equi join.

PYQ R25 Q4(a), R24 Q4(a)

Natural Join	Equi Join	Theta Join
Joins on all attributes having the same name	Joins on an explicit equality condition	Joins on any comparison condition
Condition is implicit	Condition is written explicitly with =	Condition may use <, >, ≤, ≥, ≠
Common attributes appear once	Join attribute appears twice	Join attribute appears twice
Notation R ⋈ S	R ⋈R.A = S.B S	R ⋈θ S

Q10. How is equi join different from natural join?

PYQ model questions

In an **equi join** the equality condition is written explicitly and the join attribute appears twice in the result, once from each relation. In a **natural join** the condition is implicit — the DBMS joins on all identically named attributes and keeps only one copy of each. Therefore a natural join is a special equi join with the duplicate column removed, and it is used only when the attribute names match.

Q11. What are the advantages of outer joins? Name the three types.

PYQ model questions

An outer join preserves tuples that have no match, which an inner join would silently drop. This is essential for reports such as "all customers and their orders, including customers with no orders", where losing the unmatched customers would give a wrong answer. The three types are **left outer join** (preserves the left relation), **right outer join** (preserves the right relation) and **full outer join** (preserves both).

Q12. What is a self join?

A **self join** is a join of a relation with itself. Because the same relation appears twice, both copies must be renamed — for example $\rho_E(\text{EMPLOYEE}) \bowtie_{\text{E.ManagerID = M.EmpID}} \rho_M(\text{EMPLOYEE})$ to list each employee with the name of the manager. Self joins are used whenever a recursive relationship has been stored in one table.

Q13. Define relational calculus. Why is it called non-procedural?

Relational calculus is a declarative query language based on predicate logic, in which a query describes the properties that the required tuples or values must satisfy rather than the sequence of operations to obtain them. It is called **non-procedural** (or declarative) because the user states only *what* is wanted; the DBMS decides *how* to evaluate it.

Q14. Differentiate TRC from DRC.

Tuple Relational Calculus	Domain Relational Calculus
Variables stand for whole tuples	Variables stand for individual attribute values
Form: { t such that P(t) }	Form: { <x1, x2, ...> such that P(...) }
Attributes accessed as t.Name	Values accessed positionally
Relation membership: t ∈ STUDENT	Relation as predicate: STUDENT(sid, name, dept, cgpa)
Reasoning is about rows	Reasoning is about values

Q15. What is a safe relational calculus expression?

A **safe** expression is one whose result contains only values drawn from the **active domain** of the database, so the answer is always finite. An unsafe expression such as { t such that t is not in STUDENT } would try to return every tuple in the universe that is not in the relation, which is infinite and meaningless. Every practical calculus query must be range-restricted to be safe.

Q16. Explain the existential and universal quantifiers.

The **existential quantifier** ∃ means "there exists at least one" and is used for queries containing "any" or "at least one" — for example students enrolled in at least one course. The **universal quantifier** ∀ means "for all" and is used for queries containing "every" or "all" — for example students enrolled in every required course. In practice ∀ is usually written with implication, as ∀r (r ∈ REQUIRED ⇒ ∃e (...)).

Q17. Compare relational algebra with relational calculus.

PYQ R24 Q1(g)

Relational Algebra	Relational Calculus
Procedural — says how to compute	Declarative — says what is required
Built from operators σ, π, ⋈, ∪, −, ×	Built from predicates, ∧, ∨, ¬, ∃, ∀
Variables are not central	Tuple or domain variables are central
Sequence of operations matters	Order of evaluation is decided by the DBMS
Equivalent expressive power for safe queries	Equivalent expressive power for safe queries

Q18. How does relational algebra differ from SQL in expressiveness and execution?

PYQ R24 Q1(g)

Relational algebra is a **procedural** notation: the expression fixes the order of operations and therefore hints at an execution plan. SQL is **declarative** and adds features that pure algebra lacks — aggregation, grouping, ordering, NULL handling, string matching, nested subqueries and duplicate retention. For execution, the DBMS parses the SQL, converts it into an internal relational-algebra tree, and then optimises that tree, so the algebra is the execution language while SQL is the user language.

Q19. What is SQL? Name its main categories of commands.

SQL (Structured Query Language) is the standard relational database language. Its categories are **DDL** (CREATE, ALTER, DROP, TRUNCATE), **DML** (INSERT, UPDATE, DELETE), **DQL** (SELECT), **DCL** (GRANT, REVOKE) and **TCL** (COMMIT, ROLLBACK, SAVEPOINT).

Q20. Differentiate DROP and TRUNCATE.

PYQ B26 Q1(i)

DROP	TRUNCATE
Removes the table structure and its data	Removes all rows but keeps the structure
Table no longer exists	Table exists and is empty
Belongs to DDL	Usually classified as DDL
Cannot be rolled back in most systems	Can often be rolled back
Removes indexes, constraints and privileges	Keeps indexes and constraints
Slower	Very fast, because it deallocates pages

Q21. What is DDL? Give examples.

DDL (Data Definition Language) is the set of SQL statements used to define and modify the structure of database objects. Examples are CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX, CREATE VIEW and TRUNCATE TABLE. DDL statements update the data dictionary and are usually auto-committed.

Q22. What is DML? Give examples.

DML (Data Manipulation Language) is the set of statements used to insert, update, delete and retrieve the data stored in the tables. Examples are INSERT, UPDATE, DELETE and SELECT. DML statements change the instance, not the schema, and they participate in transactions.

Q23. What is DCL? Give examples.

DCL (Data Control Language) is the set of statements used to control access to the database. Examples are GRANT (give a privilege), REVOKE (take it back), and in some systems DENY. DCL enforces security by deciding which user may perform which operation on which object.

Q24. What is QBE?

QBE (Query By Example) is a visual, form-based query language in which the user fills in a skeleton table on the screen: entering a value in a column means "match this", entering a condition means "satisfy this", and entering a variable with P. means "print this column". It was designed for non-programmers and is still found as a graphical front end in some desktop database products.

Q25. What is the difference between a unary and a binary relational algebra operator?

A **unary** operator acts on a single relation — selection, projection and rename. A **binary** operator acts on two relations — union, intersection, difference, Cartesian product and join. Binary operators other than the product usually require union compatibility.

Q26. Why is the Cartesian product rarely used directly?

Because it pairs every tuple of one relation with every tuple of the other, producing mostly meaningless combinations and a result whose cardinality is the product of the two inputs. It is used only as a building block, for example to express a join or a division, and is normally followed by a selection that removes the meaningless pairs.

Q27. What is a view? State its advantages.

Model question — Unit 3

A **view** is a stored query that behaves like a table but holds no data of its own; the DBMS evaluates the query each time the view is used. Its advantages are **security**, because a user can be restricted to a view that hides sensitive rows or columns; **simplicity**, because a complex join is presented as one simple table; **independence**, because a view hides a change in the base tables; and **reusability**, because the same definition is shared by many queries. A view on a single table can usually be updated; a view involving joins, aggregates or DISTINCT is normally read-only.

Q28. What is a subquery? Distinguish a simple subquery from a correlated subquery.

Model question — Unit 3

A **subquery** is a SELECT nested inside another statement; it may return a single value, a single column or a whole table. A **simple** subquery does not reference the outer query, so it is evaluated

once and its result reused. A **correlated** subquery references a column of the outer query, so it is evaluated once per outer row — more expensive, but it is the natural way to express "exists" and "for all". Example: WHERE Salary > (SELECT AVG(Salary) FROM Emp) is simple, while WHERE EXISTS (SELECT 1 FROM Emp e WHERE e.DeptID = d.DeptID) is correlated.

Q29. Differentiate WHERE from HAVING.

Model question — Unit 3

WHERE	HAVING
Filters rows before grouping	Filters groups after grouping
Cannot contain an aggregate function	Usually contains one
Applied to individual tuples	Applied to groups formed by GROUP BY
Written before GROUP BY	Written after GROUP BY
Example: WHERE Salary > 30000	Example: HAVING AVG(Salary) > 50000

Q30. Explain the aggregate functions and the rule for NULL values.

Model question — Unit 3

Function	Returns
COUNT(*)	Number of rows, including those with NULL
COUNT(column)	Number of non-NULL values in that column
SUM, AVG	Total and average of the non-NULL values
MIN, MAX	Smallest and largest non-NULL value

Every aggregate except COUNT(*) **ignores NULLs**, so AVG(Salary) divides by the number of employees whose salary is known, not by the total number of employees. This changes the answer whenever a column is nullable, which is why IS NULL and COALESCE are used deliberately in aggregate queries.

Q31. What is the purpose of GROUP BY and ORDER BY?

Model question — Unit 3

GROUP BY partitions the rows into groups that share a value, so that an aggregate is computed per group rather than over the whole table — for example the average salary per department. **ORDER BY** sorts the final result in ascending or descending order of one or more columns and does not change which rows are returned. GROUP BY works before the SELECT list is evaluated, while ORDER BY works last of all.

Q32. Differentiate CHAR and VARCHAR.

Model question — Unit 3

CHAR(n)	VARCHAR(n)
Fixed length — always n characters	Variable length — up to n characters
Padded with blanks when shorter	Stores only the characters supplied
Faster for fixed-size data	Saves space for variable-length data
Suits codes such as PIN or roll number	Suits names and addresses

Q33. What is the difference between DELETE and TRUNCATE?

Model question — Unit 3

DELETE	TRUNCATE
Removes selected rows, or all rows	Removes all rows
WHERE clause allowed	WHERE clause not allowed
DML, can be rolled back	DDL, usually cannot be rolled back
Slow, row by row	Very fast, deallocates pages
Fires triggers	Does not fire triggers
Keeps identity counters	Resets identity counters in most DBMSs

Q34. What is the order of evaluation of a SELECT statement?

Model question — Unit 3

FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY. The tables are read first, rows are filtered, groups are formed, groups are filtered, the output columns are computed, and finally the result is sorted. This order explains two common errors: a column alias defined in SELECT cannot be used in WHERE, because WHERE runs before SELECT, but it can be used in ORDER BY, because ORDER BY runs afterwards.

Section B — 7-Mark Questions

Q1. Explain the join operations: natural, theta, equi, outer and self join, with examples.

Natural, Equi and Theta Join on the same two relations

EMPLOYEE (EmpID, Name, DeptID)

EmpID	Name	DeptID
E1	Amit	D1
E2	Priya	D2
E3	Rahul	D1

DEPARTMENT (DeptID, DeptName)

DeptID	DeptName
D1	CSE
D2	AIDS

NATURAL JOIN — common column once

EmpID	Name	DeptID	DeptName
E1	Amit	D1	CSE
E2	Priya	D2	AIDS
E3	Rahul	D1	CSE

EQUI JOIN — join column twice

EmpID	Name	E.DeptID	D.DeptID	DeptName
E1	Amit	D1	D1	CSE
E2	Priya	D2	D2	AIDS

THETA JOIN — any comparison, here on a salary band

Name	Salary	Grade
Amit	32000	G1
Priya	48000	G2

A natural join is an equi join on all same-named attributes, with the duplicate column removed.

PYQ R25 Q4(a), R24 Q4(a), B26 Q1(g)

The idea of a join

A join combines tuples from two relations that satisfy a matching condition. It is the operation that reconstructs information which normalization has distributed across several tables, and it is the most important operator for exams because almost every query uses one.

Natural join

R ⋈ S joins on all attributes that have the same name and removes the duplicate column. It is used when the foreign key and the referenced primary key share a name.

EMPLOYEE (EmpID, EmpName, DeptID)	DEPARTMENT (DeptID, DeptName)
E101, Amit, D1	D1, CSE

EMPLOYEE (EmpID, EmpName, DeptID)	DEPARTMENT (DeptID, DeptName)
E102, Priya, D2	D2, AIDS
E103, Rahul, D1	D3, ECE
E104, Neha, D3	

$\pi_{\text{EmpName, DeptName}}$(EMPLOYEE ⋈ DEPARTMENT) gives: Amit–CSE, Priya–AIDS, Rahul–CSE, Neha–ECE. The common attribute DeptID appears once.

Theta join

A theta join uses any comparison condition, not only equality. It is needed when the match is a range rather than an exact value.

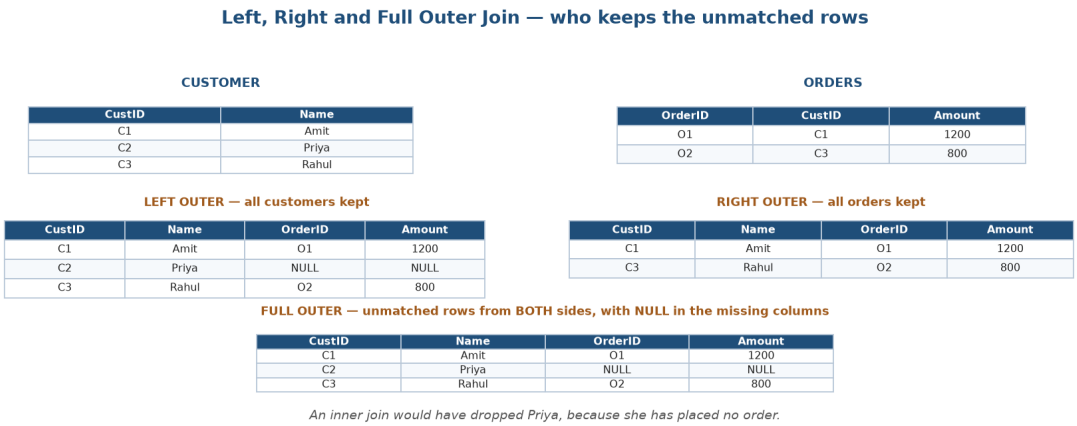
EMPLOYEE (EmpID, Name, Salary)	SALARY_GRADE (Grade, MinSalary, MaxSalary)
E1, Amit, 32000	G1, 30000, 39999
E2, Priya, 48000	G2, 40000, 59999
E3, Rahul, 62000	G3, 60000, 79999
E4, Neha, 76000	

$\pi_{\text{Name, Salary, Grade}}$(EMPLOYEE $\bowtie_{\text{Salary} \geq \text{MinSalary} \wedge \text{Salary} \leq \text{MaxSalary}}$ SALARY_GRADE) gives Amit–G1, Priya–G2, Rahul–G3, Neha–G3.

Equi join

An equi join is a theta join whose only condition is equality, written explicitly: EMPLOYEE $\bowtie_{\text{EMPLOYEE.DeptID} = \text{DEPARTMENT.DeptID}}$ DEPARTMENT. The join column appears twice in the result unless a projection removes it. A natural join is therefore an equi join with the duplicate column dropped.

Left, right and full outer join



CUSTOMER (CustomerID, CustomerName)	ORDERS (OrderID, CustomerID, Amount)
C1, Amit	O1, C1, 1200
C2, Priya	O2, C1, 800
C3, Rahul	O3, C3, 1500
C4, Sneha	

CUSTOMER LEFT OUTER JOIN ORDERS keeps C2 and C4 with NULL order details. ORDER_ITEM RIGHT OUTER JOIN PRODUCT keeps products that were never ordered. STAFF FULL OUTER JOIN

DEPARTMENT keeps both unmatched staff and unmatched departments. The rule to remember is that the side whose tuples must survive goes on the preserved side, and unmatched attributes become NULL.

Self join

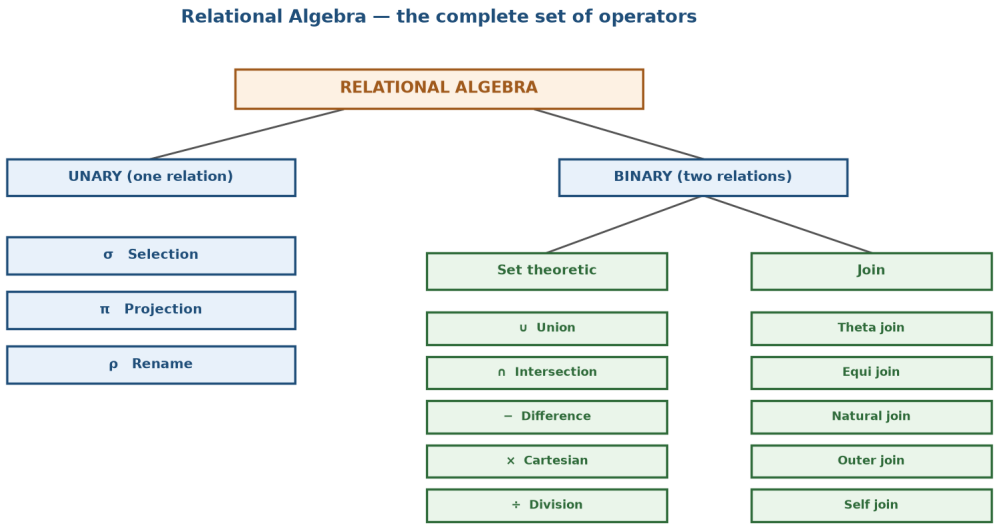
EMPLOYEE (EmpID, EmpName, ManagerID)
E1, Amit, NULL
E2, Priya, E1
E3, Rahul, E1
E4, Sneha, E2

$\pi_{E.EmpName \rightarrow Employee,\ M.EmpName \rightarrow Manager}(\rho_E(EMPLOYEE) \bowtie_{E.ManagerID\ =\ M.EmpID} \rho_M(EMPLOYEE))$ gives Priya–Amit, Rahul–Amit, Sneha–Priya. Amit is absent because the join drops the NULL, so a left outer self join is needed to keep the top-level manager.

Comparison

Join	Condition	Duplicate columns	Unmatched tuples
Theta	Any comparison	Kept	Dropped
Equi	Equality only	Kept	Dropped
Natural	Equality on 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, relation renamed	Kept	Depends on join type

Q2. Explain all the relational algebra operations with examples.



Syllabus Unit III — set theoretic and relational operators

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 R and S	Students in both the cricket and chess teams
Difference	R − S	Tuples in R but not in S	Students who registered but did not appear

Operator	Symbol	Meaning	Example
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 non-S attributes that are associated with every value of S. It is the only operator that expresses "for all" directly. If R(A, B) and S(B), then R ÷ S is the set of A values whose matching B-set contains all of S's B values. Example: students who have taken every course in a required list.

Relational (special) operators

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

Worked example combining operators

To find the names of CSE students enrolled in DBMS:

$\pi_{Name}(\sigma_{Dept='CSE' \wedge Title='DBMS'}((STUDENT \bowtie ENROLLMENT) \bowtie COURSE))$ The joins reconstruct the relationship, the selection filters both conditions, and the projection keeps the required column. The optimiser may push $\sigma_{Dept='CSE'}$ down to STUDENT and $\sigma_{Title='DBMS'}$ down to COURSE before joining, which is cheaper because it shrinks the inputs.

Why the operators matter

Relational algebra is the internal language of the query optimiser. Every SQL query is translated into an algebra tree, and the optimiser rewrites that tree using equivalences such as "selection distributes over join" or "join is commutative" to produce a cheaper plan with the same result.

Q3. Solve relational algebra queries on the product, client and salesman schema.

PYQ R25 Q4(c)

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

$\pi_{salesman_name,\ client_name}(\sigma_{salesman.city\ =\ client.city}(salesman \times client))$ The Cartesian product pairs every salesman with every client, the selection keeps only the same-city pairs, and the projection keeps the two names. A theta join notation is equivalent and clearer: $salesman \bowtie_{salesman.city\ =\ client.city} client$.

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

$\pi_{salesman_name,\ address}(\sigma_{sell_price\ >\ 10000}(salesman \bowtie product))$ The natural join on product_id links each salesman to the product he sells; the selection applies the price condition; the projection returns the required columns. Note that phone is not present in the given schema, so it cannot be projected — in an exam, state this and either omit it or assume a contact column.

(iii) Names of salesmen who sell products manufactured by Samsung

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

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

$\sigma_{state = 'Odisha' \wedge bal_due > 20000}(client)$ No join is needed because all the required attributes are in one relation.

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

$\pi_{client_id,\ client_name}(\sigma_{product_name\ =\ 'Washing\ Machine'}(client \bowtie product))$ The natural join on client_id connects each client to the products he purchased; the selection keeps the washing-machine rows.

Q4. Solve relational algebra queries on the passenger, agency, flight and booking schema.

PYQ R24 Q4(c)

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

$\pi_{pname}(passenger \bowtie \sigma_{fid\ =\ 'UA2321'}(booking))$ The selection is applied to booking first, which is cheaper than joining the whole booking relation.

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

$\pi_{pname,\ pcity}(passenger \bowtie booking \bowtie \sigma_{aname\ =\ 'JET\ Agency'}(agency))$ Two natural joins are needed because booking is the associative relation connecting passenger and agency.

Note on the pattern

Every one of these questions follows the same three-step method: find the relations that hold the required attributes, join them on their common keys, then apply σ for conditions and π for the output columns.

Q5. Solve the P, Q, R natural join and outer join problem.

PYQ B26 Q4(a) — with the actual tuples from the paper

Relations

P: X	P: Y	P: Z
X1	Y1	Z1
X1	Y1	Z2
X2	Y2	Z2
X2	Y4	Z4

Q: X	Q: Y	Q: T
X2	Y1	2
X1	Y2	5
X1	Y1	6
X3	Y1	1

R: Y	R: 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 P ⋈ Q matches on the common attributes X and Y, so only rows where both X and Y agree survive: (X1, Y1) appears in both P and Q. The additional selection repeats the same condition, so it removes nothing further.

Result: 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 having the same Y. Then the selection repeats the join condition, so again nothing extra is removed.

Result: 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

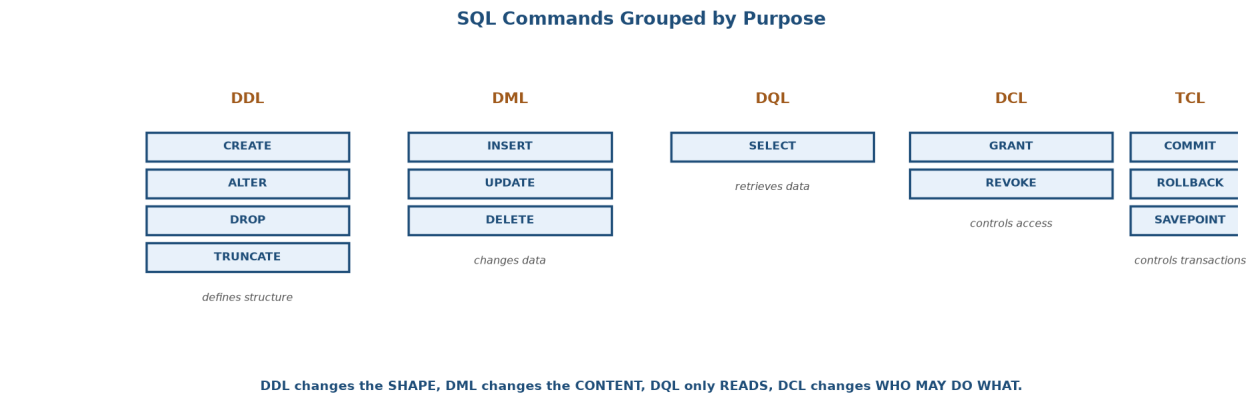
Q LEFT OUTER JOIN R preserves every Q tuple. Q's Y values are Y1, Y2 and Y1, all of which exist in R, so no Q row is lost and no NULL is introduced; the unmatched R rows (Y3, V2) are dropped because they are on the non-preserved side.

Result: 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 the (Y3, V2) row with NULL for X and T, which is exactly the situation an outer join exists to handle.

Q6. Explain the DDL, DML and DQL commands of SQL with syntax and examples.



PYQ B26 Q4(b)

DDL — Data Definition Language

DDL defines and changes the structure of the database objects.

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, modifies or drops a column
DROP TABLE	DROP TABLE Emp;	Removes the table and its data
TRUNCATE	TRUNCATE TABLE Emp;	Removes all rows, keeps the structure
CREATE INDEX	CREATE INDEX idx_name ON Emp(Name);	Builds an index for faster search
CREATE VIEW	CREATE VIEW CSE_Emp AS SELECT * FROM Emp WHERE DeptID = 10;	Defines a virtual table

DML — Data Manipulation Language

DML changes the data inside the tables.

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
UPDATE	UPDATE Emp SET Salary = Salary * 1.10;	Modifies existing rows
DELETE	DELETE FROM Emp WHERE Salary < 50000;	Removes selected rows

DQL — Data Query Language

DQL retrieves data and has the general form:

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;

DCL and TCL

GRANT SELECT ON Emp TO user1; REVOKE SELECT ON Emp FROM user1; and COMMIT, ROLLBACK, SAVEPOINT for transaction control.

Why the categories matter

DDL statements change the schema and update the data dictionary; DML statements change the instance and can be rolled back; DQL only reads. Confusing DROP with DELETE — one removes the structure, the other removes rows — is a common exam error.

Q7. Write the SQL queries on the Department and Employee tables.

PYQ R24 Q4(b)

Tables given in the paper

Department: DeptID	Department: DeptName
10	HR
20	IT
30	Finance

Employee: 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 both tables with constraints 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, whose salary is 45,000.

(v) Display the average employee salary

SELECT AVG(Salary) AS Average_Salary FROM Employee;

Result: 56,000 before the update, 61,600 after the 10 percent increase.

(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.

Q8. Write the SQL queries on the University_Faculty_Details table.

PYQ R25 Q4(b)

The 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
35456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

(1) Name, department and salary for 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., Brandt 92000 is excluded because it exceeds 90,000, Gold 87000 Physics, Singh 80000 Finance.

(2) Name, department and salary of faculty 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;

This refers to a different table that is not part of the given schema; state the assumption and write the statement for 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. Explain relational calculus with TRC and DRC, giving solved queries.

Syllabus Unit III

The idea

Relational calculus is declarative. A query is written as a set of tuples or values satisfying a logical predicate, and the DBMS chooses the evaluation strategy. It is equivalent in power to relational algebra for safe expressions.

Sample database

STUDENT: StudentID	Name	Dept	CGPA
S1	Asha	CSE	8.7
S2	Bikash	AIDS	8.1
S3	Chitra	CSE	7.5
S4	Deepak	ECE	9.0

ENROLLMENT: StudentID	CourseID	Grade
S1	C1	A
S1	C2	B
S2	C3	A
S3	C1	B
S4	C2	A

TRC — tuple variables

Form: { t such that P(t) }. Attributes are accessed as t.Name.

- All CSE students: { s | s ∈ STUDENT ∧ s.Dept = 'CSE' }
- Names with CGPA above 8.5: { t | ∃s (s ∈ STUDENT ∧ s.CGPA > 8.5 ∧ t.Name = s.Name) }
- Students enrolled in at least one course: { s | s ∈ STUDENT ∧ ∃e (e ∈ ENROLLMENT ∧ e.StudentID = s.StudentID) }
- Students not enrolled in any course: { s | s ∈ STUDENT ∧ ¬∃e (e ∈ ENROLLMENT ∧ e.StudentID = s.StudentID) }
- Students enrolled in every required course: { s | s ∈ STUDENT ∧ ∀r (r ∈ REQUIRED ⇒ ∃e (e ∈ ENROLLMENT ∧ e.StudentID = s.StudentID ∧ e.CourseID = r.CourseID)) }

DRC — domain variables

Form: { <x1, x2, ...> such that P(...) }. Attributes are positional.

- Names of CSE students: { <name> | ∃sid ∃cgpa (STUDENT(sid, name, 'CSE', cgpa)) }
- Names and course titles: { <name, title> | ∃sid ∃dept ∃cgpa ∃cid ∃grade ∃cdept (STUDENT(sid,name,dept,cgpa) ∧ ENROLLMENT(sid,cid,grade) ∧ COURSE(cid,title,cdept)) }

The quantifier patterns to memorise

English phrase	Calculus form
at least one / any	∃
every / all	∀, usually with ⇒
none / not enrolled	¬∃
not all	¬∀

Safety

Every variable must be range-restricted by a relation predicate, so the result comes only from the active domain. Without this restriction an expression can describe an infinite set and cannot be evaluated.

Comparison of the three languages

Aspect	Relational Algebra	Relational Calculus	SQL
Style	Procedural	Declarative	Declarative
Basis	Operators	Predicate logic	Both, with extensions
Variables	None	Tuple or domain	None
"Every" queries	Division	∀	NOT EXISTS or double NOT EXISTS
Used by	Optimiser, execution	Formal specification	End users and programs

Q10. Solve the relational algebra and SQL queries on the Suppliers, Parts and Supply schema.

Model question — Unit 3

Schema

Suppliers(Sno, Sname, City); Parts(Pno, Pname, Pcolour); Supply(Sno, Pno, Quantity)

(i) Names of suppliers who supply a red part

$\pi_{Sname}(\sigma_{Pcolour = 'Red'}(Suppliers \bowtie Supply \bowtie Parts))$

```
SELECT DISTINCT S.Sname FROM Suppliers S JOIN Supply SP ON S.Sno = SP.Sno JOIN Parts P ON SP.Pno = P.Pno WHERE P.Pcolour = 'Red';
```

(ii) Names of suppliers who supply some red or green part

$\pi_{Sname}(\sigma_{Pcolour = 'Red' \vee Pcolour = 'Green'}(Suppliers \bowtie Supply \bowtie Parts))$

```
SELECT DISTINCT S.Sname FROM Suppliers S JOIN Supply SP ON S.Sno = SP.Sno JOIN Parts P ON SP.Pno = P.Pno WHERE P.Pcolour IN ('Red','Green');
```

(iii) Names of suppliers who supply every part

$\pi_{Sname}(Suppliers \bowtie (\pi_{Sno,\ Pno}(Supply) \div \pi_{Pno}(Parts)))$ Division is required because the question says "every" — an ordinary join would return suppliers who supply at least one part.

(iv) Suppliers who supply no part

$\pi_{Sname}(Suppliers) - \pi_{Sname}(Suppliers \bowtie Supply)$

The pattern to remember

Join to reconstruct the relationship, select for the condition, project for the output. Any question containing "every" or "all" becomes a division, and any question containing "no" or "not" becomes a set difference.

Q11. Solve the relational algebra queries on the Employee, Assigned_to and Project schema.

Model question — Unit 3

Schema

Employee(eid, ename); Assigned_to(pnum, enum); Project(pnum, pname, supervisor)

(i) Details of employees who work on the Database project

$\pi_{eid,\ ename}(\sigma_{pname = 'Database'}(Employee \bowtie Assigned_to \bowtie Project))$

(ii) Details of employees who work on project number C353

$\pi_{eid,\ ename}(\sigma_{pnum\ =\ 'C353'}(\text{Employee} \bowtie \text{Assigned_to}))$ The selection is applied to Assigned_to before the join, which is cheaper than joining the whole relation first.

(iii) Employee numbers of employees who work on all projects

$\pi_{enum}(\text{Assigned_to}) \div \pi_{pnum}(\text{Project})$

(iv) Employee numbers of employees who do not work on project number C453

$\pi_{eid}(\text{Employee}) - \pi_{enum}(\sigma_{pnum\ =\ 'C453'}(\text{Assigned_to}))$ The difference removes those who do work on it. Both sides must be union compatible, so each projects a single employee-number column.

Q12. Explain the SELECT statement with all its clauses.

Model question — Unit 3

The general form

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

Clause	Purpose	Example
FROM	Names the relations to be read	FROM Employee E
WHERE	Filters rows before grouping	WHERE Salary > 40000
GROUP BY	Groups rows so aggregates apply per group	GROUP BY DeptID
HAVING	Filters groups after grouping	HAVING AVG(Salary) > 50000
SELECT	Chooses output columns and computes expressions	SELECT Name, Salary * 1.1
DISTINCT	Removes duplicate result rows	SELECT DISTINCT DeptID
ORDER BY	Sorts the final result	ORDER BY Salary DESC

Worked example

Department-wise average salary of employees earning above 40,000, showing only departments whose average exceeds 50,000:

SELECT DeptID, AVG(Salary) FROM Employee WHERE Salary > 40000 GROUP BY DeptID HAVING AVG(Salary) > 50000 ORDER BY DeptID;

Why the order matters

The evaluation order is FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY. That explains why an aggregate cannot appear in WHERE, and why a SELECT alias can appear in ORDER BY but not in WHERE.

Q13. Explain SQL joins, subqueries and views with examples.

Model question — Unit 3

Joins

Join	SQL form	Keeps
INNER JOIN	SELECT * FROM E JOIN D ON E.DeptID = D.DeptID;	Only matched rows
NATURAL JOIN	SELECT * FROM E NATURAL JOIN D;	Matched rows, common column once
LEFT JOIN	SELECT * FROM E LEFT JOIN D ON …;	All rows of E
RIGHT JOIN	SELECT * FROM E RIGHT JOIN D ON …;	All rows of D
FULL OUTER JOIN	SELECT * FROM E FULL OUTER JOIN D ON …;	All rows of both sides
SELF JOIN	SELECT a.Name, b.Name FROM Emp a JOIN Emp b ON a.ManagerID = b.EmpID;	The table paired with itself

Subqueries

A scalar subquery returns one value: WHERE Salary > (SELECT AVG(Salary) FROM Emp). A multi-row subquery is used with IN or ANY: WHERE DeptID IN (SELECT DeptID FROM Dept WHERE Location = 'Delhi'). A correlated subquery references the outer row and is the natural way to write EXISTS and NOT EXISTS.

Views

CREATE VIEW CSE_Emp AS SELECT EmpID, Name FROM Emp WHERE DeptID = 10;

A view is a stored query, not stored data. It gives security, because a user can be restricted to the columns and rows the view exposes; simplicity, because a complex join appears as one table; and independence, because the base tables can change without affecting the users of the view.

Q14. Explain QBE with an example.

Model question — Unit 3

The idea

QBE (Query By Example) is a visual, form-based query language. The user is shown a skeleton table for each relation and fills it in, so the query is written by example rather than by syntax.

Notation	Meaning
Constant	Match this exact value
Example variable (_x)	Use the same variable in two tables to join them
P.	Print this column in the output
_x under a value	Not equal to that value
Condition box	Express ranges and compound conditions

Example

To find the names of suppliers who supply part P1, the Suppliers skeleton leaves Sno blank and puts P._n under Sname, while the Supply skeleton puts _s under Sno and 'P1' under Pno. The shared variable _s joins the two skeletons, and P. marks the column to print.

Its place in the syllabus

QBE was designed for non-programmers and survives as a graphical front end in some desktop database products. A short answer needs the idea, the P. notation, and one example of a join written with a shared variable.