Date

a) **Relational Algebra** → It is a pure query language

→ It is procedural in nature, that's why we have to specify the procedure also for performing any given operation.

Following are some of the basic Relational Algebra operator :-

Selection (σ)
Projection (π)
Set operations
↳ Union operation ($\cup$)
↳ Intersection operation ($\cap$)
↳ Set difference operation ($-$)
Cartesian Product ($\times$)
Rename operation (ρ)
Natural Join or Inner Join ($\bowtie$)
Outer Join
↳ Left outer Join ($⟕$)
↳ Right outer Join ($⟖$)
↳ Full outer Join ($⟗$)
Division operation ($\div$)

Selection (σ) → It is use to select the tuples from a given relation satisfying a given condition.

→ In other words, we can say that it is use to restrict or filter the tuples in a relation.

→ It is unary in nature (It takes only one relation as I/p).

→ It is commutative in nature.

Date

Representation of selection operation →

$$\boxed{\sigma_p(r) = \{t \mid t \in r \ \& \ P(t)\}}$$

Example → Consider a relation R(A, B, C) & r(R) be a relation instance on R.

r(R)

A	B	C
a_1	b_1	10
a_2	b_2	15
a_3	b_3	20
a_4	b_4	25

$\sigma_{C>15}(r)$ ⟹

A	B	C
a_3	b_3	20
a_4	b_4	25

(No. of Col. i.e Degree remains Same)

→ ★ The degree or arity of output relation will be same as that of the original relation.

→ It is also known as Horizontal Selection because it is restricting the tuples in the O/P.

Projection (π) → It is use to select specific attributes of a relation.

→ It is unary in nature i.e it takes only one relation as I/P.

→ It is not commutative in nature.

→ It is also known as Vertical Selection.

→ ★ The Degree or arity of the resultant relation will be equal to the no. of attributes in the projection list.

→ ★ But the Cardinality of the resultant relation will be same as that of the original relation

Date

Representation of projection operation.

$$\pi_{A_1, A_2, A_3 \ldots A_K}(r)$$

No. of Rows remains same
No. of col varies

where,
r = relation Instance.
$A_1, A_2 \ldots A_K$ = Projection list.

Exen example :-

A	B	C
a_1	b_1	10
a_2	b_2	20
a_3	b_3	30

$\xrightarrow{\pi_{A,B}(r)}$

A	B
a_1	b_1
a_2	b_2
a_3	b_3

Set operation →

→ The operations union, intersection & set difference are known as Set operation.

→ For performing set operations ~~of~~ on the given relation they must satisfy ~~fio~~ 2 conditions also known as Union compatibility Conditions :-

1. → The degree or arity of participating relation must be same.
2. → The Domains of the corresponding attributes must also be same.

Union → $r \cup s = \{t \mid t \in r \text{ or } t \in s\}$

Intersection → $r \cap s = \{t \mid t \in r \,\&\, t \in s\}$

Set difference → $r - s = \{t \mid t \in r \,\&\, t \notin s\}$

Date

Example →

let $r(R)$ & $s(S)$ be two relation instance.

$r(R)$

A	B	C
a_1	b_1	10
a_2	b_2	20
a_3	b_3	30
a_4	b_4	40

$s(S)$

A	B	C
a_1	b_1	10
a_3	b_3	30
a_5	b_5	50
a_6	b_6	60

→

$r \cup s$

A	B	C
a_1	b_1	10
a_2	b_2	20
a_3	b_3	30
a_4	b_4	40
a_5	b_5	50
a_6	b_6	60

$r \cap s$

A	B	C
a_1	b_1	10
a_3	b_3	30

$r - s$

A	B	C
a_2	b_2	20
a_4	b_4	40

→ If (r) contains (m) tuples & (s) contains (n) tuples, then

		Max tuples	Min tuples
★ $r \cup s$	→	$(m+n)$	$\max(m, n)$
$r \cap s$	→	$\min(m, n)$	0
$r - s$	→	m	0

Cartesian Product →

let $r(R)$ & $s(S)$ be two relation instance on schema R & S respectively, then $r \times s$ be a relation on schema $R \cup S$ given by,

$$r \times s = \{ tu \mid t \in r \ \& \ u \in s \}$$

Example →

r

A	B
a_1	b_1
a_2	b_2

s

C
10
20

$r \times s$

A	B	C
a_1	b_1	10
a_1	b_1	20
a_2	b_2	10
a_2	b_2	20

Date

Example →

depositor

Cname	ac.no.
RAM	a-100
SHYAM	a-200
MOHAN	a-300

accounts

ac.no	bal	branch
a-100	1000	Moran
a-200	2000	Gwl
a-300	3000	Lashkar

depositor x account

Cname	ac.no.	ac.no.	bal	branch
RAM	a-100	a-100	1000	Moran
RAM	a-100	a-200	2000	Gwl
RAM	a-100	a-300	3000	Lashkr.
SHYAM	a-200	a-100	1000	Moran
SHYAM	a-200	a-200	2000	Gwl
SHYAM	a-200	a-300	3000	Lashkr.
MOHAN	a-300	a-100	1000	Moran
MOHAN	a-300	a-200	2000	Gwl
MOHAN	a-300	a-300	3000	Lashkr.

→ Cartesian Product is a very time consuming operation because a lot of tuples are generated.

→ No usefull information can be obtained from cartesian product.

→ If r contains m tuples and s contains n tuples then r x s contain (m x n) tuples.

→ Cartesian Product is also known as Cross Product or Cross Join.

Fetch Data from (Matching) Multiple Tables → Use Joins → (Condition) must have attribute in common

Date

Natural Join or Inner Join →

→ If we want to fetch data from multiple tables then natural Join is the most useful operation.

→ For performing natural join between two or more tables, the tables must have atleast one attribute in common i.e, the tables must be related with the help of foreign key.

→ Natural join is basically a combination of three operations :-

1. → Cartesian product (X)
2. → Selection of tuples based on equality conditions on common attributes.
3. → Projection of distinct attributes of the participating relations.

Example :-

For the same cartesian product example

1. After cartesian product, select useful tuples having same attribute.

2.

cname	depositor ac.no.	accounts acc.no	Bal	Branch
RAM	a-100	a-100	1000	Morar
SHYAM	a-200	a-200	2000	Gwl
MOHAM	a-300	a-300	3000	Lashkar

(depositor X accounts)

$\sigma_{\text{depositor.acc.no} = \text{accounts.acc.no}}$

3. $\pi_{\text{cname, depositor.acc.no, bal, branch}} \left(\sigma_{\text{d.acc} = \text{a.acc.no}} (\text{depositor} \times \text{acc.}) \right)$

Date

Cname	d.acc.no	bal	branch
RAM	a-100	1000	Moran
SHYAM	a-200	2000	Guwl
MOHAN	a-300	3000	Lahuk

depositor ⋈ accounts.

→ For performing natural join between two tables atleast one join condition is require. Therefore, for performing natural join between (n) tables atleast (n-1) join conditions are required.

→ If the tables aren't related with the help of foreign key that means they aren't having any attributes in common then natural join will be equivalent to cartesian product.

→ Natural Join is also known as Inner Join because it retrieves only the (Matching tuples) on the basis of common attribute from the given tables.

Outer Join → Fetch Mult. data (Matching / Non-Matching) → Use outer Join

Example →

depositor

Cname	ac.no
RAM	a-100
SHYAM	a-200
MOHAN	a-300
SURESH	a-400

accounts

acc.no	bal	Branch
a-200	2000	Moran
a-400	4000	Lahuk.
a-500	5000	Hazira
a-600	6000	Thatipur

Date

Left outer Join (⟕)

depositor ⟕ accounts

Cname	d.ac no	a.accno.	Bal	Branch
RAM	a-100	NULL	NULL	NULL
SHYAM	a-200	a-200	2000	MORAR
MOHAN	a-300	NULL	NULL	NULL
SURESH	a-400	a-400	4000	Lashkr.

Right outer Join (⟖)

depositor ⟖ accounts

Cname	d.accno.	a.accno	Bal	Branch
Shyam	a-200	a-200	2000	Morar
Suresh	a-400	a-400	4000	Lashkar
NULL	NULL	a-500	5000	Hazira
NULL	NULL	a-600	6000	Thatipur

Full outer join or Symmetric outer join (⟗)

depositor ⟗ accounts

C.name	d.acc.	a.acc.no	bal	Branch
RAM	a-100	NULL	NULL	NULL
SHYAM	a-200	a-200	2000	Morar
MOHAN	a-300	NULL	NULL	NULL
SURESH	a-400	a-400	4000	Lashkar
NULL	NULL	a-500	5000	Hazira
NULL	NULL	a-600	6000	Thatipur

→ If we want to fetch both matching as well as non-matching tuples from the given relations, then outer joins are used.

→ Outer Joins are basically usefull for policy Making

Date

→ For performing outer Joins also, tables must be related with the help of foreign key.

→ Outer Joins are of three types:-

1. Left outer Join
2. Right outer Join
3. Full or Symmetric Outer Join.

1. **Left outer Join** → If we want to fetch all the tuples of left hand side table but only matching tuples from right hand side table then left outer Join is performed.

- The missing information is represented by NULL

2. **Right outer Join** → If we want to fetch all the tuples from right hand side table but only matching tuples from left hand side table then right outer Join is performed.

3. **Full outer Join** → If we want to fetch all the tuples matching as well as non-matching from both the tables then full outer join is performed.

Q: Consider the fno schema and answer the fno queries.

branch (branchname, branchcity, assets)
customer (customername, customercity, customerstreet)
account (accno., branchname, balance)
loan (loanno., branchname, amount)
depositor (customername, acc.no)
Borrower (customername, loan no.)

Date

(1) Find the information of all loans over rs. 10,000

$\Rightarrow \sigma_{amt > 10,000}(loan)$

(2) Find the loan no. for each loan with an amount greater than 10,000.

$\Rightarrow \pi_{loan\,no.}(\sigma_{amt > 10,000}(loan))$

(3) Find the names of all customers who have a loan ~~and~~ or account or both from the bank.

$\Rightarrow \pi_{cname}(depositor) \cup \pi_{cname}(borrower)$

(4) Find the names of customers who have a loan as well as account at the bank.

common tuples $\Rightarrow \pi_{cname}(depositor) \cap \pi_{cname}(borrower)$

(5) Find the names of all customers who have taken loan but not deposited anything in the bank.

$\Rightarrow \pi_{cname}(borrower) - \pi_{cname}(depositor)$

(6) Find the information of all customers who live in Gwalior

$\Rightarrow \sigma_{customercity = "Gwalior"}(customer)$

(7) Find the customer name of those customers who live in Gwalior

$\Rightarrow \pi_{cname}(\sigma_{customercity = 'Gwl'}(customer))$

Date

(8) Find loan no. and Branch name of all the loans who have an amount > 10,000.

$$\Rightarrow \pi_{\text{loan no.}, \text{Branch name}} \left(\sigma_{\text{amt} > 10,000} (\text{loan}) \right)$$

(9) Find the account no. of those accounts which have amt > 10,000 and are opened in Morar Branch

$$\Rightarrow \pi_{\text{ac.no.}} \left(\sigma_{\text{balance} > 10,000 \,\wedge\, \text{branchname} = \text{Morar}} (\text{account}) \right)$$

(10) Find customer name, city, acc.no. for all the customers.

$$\Rightarrow \pi_{\text{customer.customer name}, \text{customer city}, \text{accno.}} \left(\sigma_{\text{customer.custname} = \text{depositor.custome name}} (\text{customer} \times \text{depositor}) \right)$$

or

$$\pi_{\text{customer.customernane}, \text{city}, \text{acc.no}} (\text{customer} \bowtie \text{depositor})$$

(11) Find customer name, loan no, amount

$$\Rightarrow \pi_{\text{customer name}, \text{loan.loanno}, \text{amt}} \left(\sigma_{\text{loan.loanno.} = \text{borrower loan}} (\text{loan} \times \text{borrower}) \right)$$

or

$$\pi_{\text{customernane}, \text{loan.loan no.}, \text{amt}} (\text{loan} \bowtie \text{borrower})$$

Date

(12) Find customer name who have deposited more than 1 lakh.

$$\pi_{c.name}\left(\sigma_{account.acc\,no = depositor.ac\,no. \wedge balance > 1,00,000}(account \times depositor)\right)$$

or

$$\pi_{customer}\left(\sigma_{balance > 100000}(account \bowtie depositor)\right).$$

(13) Find the customer name & city of those customers, who have deposited more than 1 lakh.

$$\left(\sigma_{balance > 100000}(account \bowtie depositor)\right)$$

$$\pi_{customer.customer name, customer city}\left(\sigma_{account.ac\,no = depositor.acc\,no. \wedge depositor.customer = custom.customer name \wedge balance > 100000}(account \times depositor \times customer)\right)$$

or.

$$\pi_{customer.cust name, customer city}\left(\sigma_{balance > 100000}(account \bowtie depositor \bowtie customer)\right)$$

(14) Find the name of all customers who have a loan at Morar Branch.

$$\pi_{customer name}\left(\sigma_{loan.loan\,no. = borrower.loan\,no \wedge branch name = "Morar"}(loan \times Borrower)\right)$$

or

$$\pi_{customer name}\left(\sigma_{Branch name = "Morar"}(loan \bowtie Borrower)\right)$$

Date

Q:- Consider the fro schema and answer the fro queries:-

employee (ename, street, city)
works (ename, companyname, salary)
manager (ename, managername)
company (companyname, city)

① Find the name of all employees who works for xyz company.

$$\pi_{ename}\left(\sigma_{companyname = xyz}(works)\right)$$

1 → works

② Find the names & cities of residence of all employees who works for xyz company.

$$\pi_{employee.ename,\ city}\left(\sigma_{employeename = works.ename \wedge companyname = "XYZ"}(employee \times works)\right)$$

2 → employee works.

③ Find the names of all employees who works for company located in Gwalior.

$$\pi_{ename}\left(\sigma_{works.companyname = company.companyname \wedge city = "Gwalior"}(works \times company)\right)$$

works company.

Date

(4) Find the names and cities of all employees who works for company located in Gwl.

$$\pi_{\text{employee.ename, employee.city}}\left(\sigma_{\substack{\text{works.companyname} = \text{company.companyname} \\ \wedge\ \text{works.ename} = \text{employee.ename} \\ \wedge\ \text{company.city} = \text{"Gwl"}}}(\text{works} \times \text{company} \times \text{employee})\right) \quad \{\text{employee, works, company}\}$$

(5) Find the names, street, cities of residence of all the employees who works for XYZ company and earn more than 1 Lakh per month.

$$\pi_{\text{employee.ename, street, city}}\left(\sigma_{\substack{\text{employee.ename} = \text{works.ename} \\ \wedge\ \text{companyname} = \text{XYZ} \\ \wedge\ \text{salary} > 1{,}00{,}000}}(\text{employee} \times \text{works})\right) \quad \{\text{employee, works}\}$$

(6) Find the name of all employees who live in the same city as that of the company they works.

$$\pi_{\text{employee.ename}}\left(\sigma_{\substack{\text{employee.ename} = \text{works.ename} \\ \wedge\ \text{works.companyname} = \text{company.companyname} \\ \wedge\ \text{employee.city} = \text{company.city}}}(\text{employee} \times \text{company} \times \text{works})\right) \quad \{\text{employee, company, works}\}$$

Date

Q. Find the names and cities of residence of all the Managers.

$$\pi_{employee.ename,\ city}\left(\sigma_{employee.ename = manage.managername}(employee \times manage)\right)$$

{employee, manage}

(8) Find the names of all managers who works for XYZ company.

$$\pi_{works.ename}\left(\sigma_{works.ename = manager.managername \wedge companyname = XYZ}(works \times manage)\right)$$

{works, manage}

(9) Find the names of all employees who aren't manager.

$$\pi_{ename}(manage) - \pi_{managername}(manage)$$

Q:- Consider the f/w schema and answer the f/w queries:-

S(S#, sname, status, city)
P(P#, pname, color)
J(J#, Jname, city)
SPJ(S#, P#, J#, qty)

(1) Get supply no. for suppliers who supply for project J1.

$$\pi_{S\#}\left(\sigma_{J\# = 'J1'}(SPJ)\right)$$

Date

(2) Get supplier name & city of supplier who supply for project J1.

{SPJ, S}

$$\pi_{sname, city}\left(\sigma_{\substack{S.S\# = SPJ.S\# \\ \wedge\, J\# = 'J1'}}(S \times SPJ)\right)$$

(3) Get part no. & part name for parts supplied by suppliers in Gnel.

S, P, SPJ

$$\pi_{pname, P\#}\left(\sigma_{\substack{S.S\# = SPJ.S\# \\ \wedge\, P.P\# = SPJ.P\# \\ \wedge\, city = \text{"Gnel"}}}(S \times P \times SPJ)\right)$$

(4) Get part no. & part name for projects supplied by suppliers in gwalior who have project in Gnel.

$$\pi_{pname, P\#}\left(\sigma_{\substack{S.S\# = SPJ.S\# \\ \wedge\, P.P\# = SPJ.P\# \\ \wedge\, J.J\# = SPJ.J\# \\ \wedge\, S.city = \text{"Gnel"} \\ \wedge\, J.city = \text{"Gnel"}}}(S \times P \times J \times SPJ)\right)$$

(5) Get part no., part name & part color for projects supplied by a supplier in the same city as that of the project.

$$\pi_{P\#, pname, pcolor}\left(\sigma_{\substack{S.S\# = SPJ.S\# \\ \wedge\, P.P\# = SPJ.P\# \\ \wedge\, J.J\# = SPJ.J\# \\ \wedge\, S.city = J.city}}(S \times P \times J \times SPJ)\right)$$

Date

(6) Get project names for projects supplied by supplier no. S1.

$$\pi_{Jname}\left(\sigma_{\substack{S.S\#=SPJ.S\# \\ J.J\#=SPJ.J\# \\ \wedge S.S\#='1'}}(S \times J \times SPJ)\right)$$

(7) Get colors of parts supplied by supplier no. S1.

$$\pi_{P.color}\left(\sigma_{\substack{P.P\#=SPJ.P\# \\ \wedge S.S\#='1'}}(P \times SPJ)\right)$$

Division Operation →

Let r and s be two relation instances on schema $R(A_1, A_2, A_3 \ldots, B_1, B_2, B_3 \ldots)$ & $S(B_1, B_2, B_3 \ldots)$ such that $R \subseteq S$

then $r \div s$ will be a relation on schema $R-S$ given by

$$r \div s = \{t \mid t \in \pi_{R-S}(r) \;\&\; \forall\, u \in S\, (tu \in r)\}$$

Example →

Let r & s be two relation instances on schema $R(A,B)$ & $S(B)$ respectively.

$r \rightarrow$

A	B
a_1	b_1
a_2	b_2
a_3	b_3
a_4	b_4
a_3	b_4
a_2	b_4

$s \rightarrow$

B
b_2
b_4

$r \div s = \frac{AB}{B}$

A
a_2

Date

r

A	B	C
a_1	b_1	c_1
a_2	b_2	c_1
a_3	b_3	c_1
a_1	b_1	c_2
a_3	b_3	c_2
a_4	b_4	c_2

S
C
c_1
c_2

$r \div s$

A	B
a_1	b_1
a_3	b_3

→ The division operation is use to answer the queries that includes the phrase "for all"

Rename Operation →

Rename operation is use to rename the result of a relational algebra expression or an existing relation.

Syntax →

$$\rho_{r(A_1, A_2 \ldots A_n)}(E)$$

where,

E is any relational Algebra expression

r is new relation name.

$A_1, A_2 \ldots A_n$ is the name of attributes of new relation.

Generalised Projection →

Generalised Projection is a extension of projection operation which allows to perform basic arithmetic operations in the projection list.

Syntax →

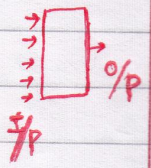
$$\Pi_{F_1(A_1), F_2(A_2) \ldots F_n(A_n)}(E)$$

Date

where, E is any relation algebra expression
$F_1, F_2 \ldots F_n$ are arithmetic functions involving attributes & constants.
$A_1, A_2 \ldots A_n$ are attributes.

Aggregate Functions or Group Functions →

→ Group functions are the functions which takes a collection of values as input and returns a single value as output.



→ Following are some of the popular group functions generally used :-

Min → It finds the minimum value among the set of values provided as input.

Max → It finds the maximum value among the set of values provided as input.

Sum → It finds the total of values provided as i/p.

Average → It find the average of i/p values.

Count → It finds the no. of values in the group.

→ All the aggregate Functions ignore Null Values.

Syntax :- $\underset{G_1, G_2 \ldots G_n}{g} \; \underset{F_1(A_1), F_2(A_2) \ldots F_n(A_n)}{(E)}$

where,
g is the group operator
E is any relational Algebra expression

Date

$F_1, F_2 \ldots F_n$ are group fn.
$A_1, A_2 \ldots A_n$ are the attributes.
$G_1, G_2 \ldots G_m$ are the attributes used to form groups, (This list may be empty).

Example →

account (acc.no., Branchname, bal)

Ac.no	Branchname	Bal
a-100	Morar	10,000
a-200	Lashkar	20,000
a-300	Gwl	30,000
a-400	Gwl	35,000
a-500	Lashkar	25,000
a-600	Morar	15,000
a-700	Morar	40,000
a-800	Lashkar	50,000
a-900	Gwl	35,000

(1) Find the average Balance.

$g_{Avg(bal)}$ (account)

Avg (bal)
28,888

(2) Find Branchwise avg. balance.

$g_{branchname, Avg(bal)}$ (account)

Branch name	Avg
Morar	21666
Gwl	33333
Lashkar	31666

(3) Find the count of those accounts that have ~~avera~~ balance greater than 35,000.

Date

$$g_{count(ac.no.)}\left(\sigma_{bal>35000}(account)\right)$$

(4) Find the account no, , Balance of those accounts that have balance greater than avg Balance.

$$g_{Avg(bal)}(account) \quad As \quad Averagebalance(avgbal)$$

$$\pi_{ac.no,bal}\left(\sigma_{bal>avgbal}(account \times Averagebalance)\right)$$

(5) Find account no. with maximum Balance.

$$g_{max(bal)}(account) \quad As \quad MaxBalance(maxbal)$$

$$\pi_{accno.}\left(\sigma_{bal=Maxbalance}(account \times MaxBalance)\right)$$

(6) Find the count of all accounts.

$$g_{count(acc.no.)}(account).$$

(7) Find the branchwise count of accounts.

$$_{branchname}g^{branchname}_{count(accno.)}(account)$$

(8) Find min, max, total & avg balance.

$$g_{min(bal),\ max(bal),\ sum(bal),\ Avg(bal)}(account)$$

Date

(9) Find Branchwise min, max, total, avg.

$$_{branchname}\mathcal{G}_{branchname\ min(bal),\ max(bal),\ sum(bal),\ avg(bal)}(account)$$

Modification in the database using Relational Algebra →

We can modify the content of database using relational algebra queries that means we can perform insertion, deletion as well as modification using queries of relational Algebra.

All these operations in relational algebra are performed using assignment operator.

Deletion operation → A delete query in relational algebra is expressed as below;

$$\boxed{r \leftarrow r - E}$$

where r is the relation
E is any relations algebra expression.

ex:- Delete a tuple having acc.no a-100 from account from previous example.

$$account \leftarrow account - \left(\sigma_{acno = a-100} (account) \right)$$

Insertion operation → In relational algebra an insertion operation is expressed as below;

$$\boxed{r \leftarrow r \cup E}$$

Date

example :- insert a tuple (a-900, Lashkar, 1000) in account.

account ← account ∪ {a-900, Lashkar, 1000}.

Modification Operation → Modification in relational algebra is performed using generalised projection.

$$\boxed{\Pi_{F_1, F_2, \ldots F_n}(E)}$$

example →

Increase the Balance of all accounts by 10%.

account ← $\Pi_{ac.no,\ branchname,\ bal * 1.10}$ (account).

SQL (Structure query language) →

SQL is an abbreviation for structure query language. It is non-procedural or declarative in nature, i.e, we have to specify only what is to be done, but not how it is to be done.

SQL contains a pre-defined set of commands also known as SQL statements to perform various operations.

All the SQL statements can be broadly classify into two four categories

SQL statements

- DDL statement
- DML statement
- DCL statement
- TCL statement

Date

DDL (Data definition language) →

→ These statements are use to specify the schema of the database. i.e, to create new database objects ~~like~~, modify existing database objects, delete an existing database object or rename an existing database object.

Following are some of the popular DDL statements

Create → This statement is use to create a new database object like table, view, user, synonym, index, trigger, cursor etc.

Drop → This statement is use to delete an existing database object.

Alter → This statement is use to modify an existing database object.

Rename → This statement is use to change the name of an existing database object.

Truncate → This statement is use to delete the data of an existing table, but not structure.

→ DDL statements are auto-commit therefore we can't undo the changes performed by any DDL statement.

Date

DML (Data Manipulation Language) →

These statements are use to perform manipulation operations on the data stored in the database.

Following are some of the popular DML statement

1. Select → It is use to retrieve the required data from the database.

Insert → This is use to insert new data in the database.

Delete → This is use to delete an existing data from the database.

Update → This is use to modify the existing data in the database.

DML statements aren't auto-commit i.e, The changes made by DML statements in the database can be rolled back before commit operation.

DCL (Data Control Language) →

These statements are use to give the permissions or to get the existing permission back from the user.

In other words, we can say that DCL statement are use to control the access to database objects.

ACID → Atomicity, consistency, Isolation, Durability

Date

Following are two popularly use DCL Statem :-

Grant → This statement is use to give the permissions to a user for performing operations on existing database objects.

Revoke → This statement is use to get the existing permission back from a user

Like DDL statements, DCL statements are also auto-commit.

PCL (Transaction control language) →

→ These statements are use to provide the support of transaction processing in the Database.

→ A transaction is nothing but a process with ACID properties.

Following are some of the TCL statements which are commonly used :-

Commit → This statement is use to make the changes performed by DML statement permanent in the database

Roll back → This statement is use to undo the changes perform by DML statements in the database.

Savepoint → This statement is use to mark the synchronisation point to be used by

Date

Commit and roll back statements.

Select statement →

It is use to retrieve the data from the database as per the given requirement.

```
SELECT    attribute(s) /*
FROM      table(s)
WHERE     condition(s)
GROUP BY  attribute(s)
HAVING    condition(s)
ORDER BY  attribute(s) [ASC/DESC]
```

A select query basically consists of two six clauses

Select clause → It is a compulsory clause.
It is equivalent to projection operation of relational algebra.
It is use to project the required attribute of a given relation.

From clause → It is also a compulsory clause
It is use to specify the tables where the required data will be found.

Where clause → It is an optional clause in the select query.
It is equivalent to selection operation of relational Algebra. It is use to restrict or filter the tuples in the o/p.

Queries

Date

The condition of where clause is checked on all the tuples individually. and only those tuples will be displayed in the o/p that satisfies the condition of where clause.

Group by clause → It is also an optional clause in the select query.

It is equivalent to group operator (g) of relational algebra.

It is use to create the groups on the basis of attributes specified in the Groupby clause.

If there is no groupby clause then the whole table will be treated as a single group.

Having clause → It is also an optional clause in the select query. It is use to restrict or filter the groups in the o/p.

The condition of having clause is applied on groups created by groupby clause. and only those groups will be available in the o/p that satisfies the condition of having clause.

For write having clause first we have to write groupby

There is no significance of using having clause without Groupby clause but no syntax error will be generated.

Orderby clause → Optional clause.

It is use to arrange the data to be displayed in a specific order according to the attributes specified in Orderby clause.

Date

The default ordering is ascending & descending order can be specified with the help of DESC.

Q:- Consider the fllw schema and answer the fllw query :-

Student (Roll.no, Name, Branch, dob, city, marks, phone no.)

~~Studen~~ Write the query to display data of All the students

TYPE-1 SIMPLE QUERIES

```
Select *
FROM Student;
```

OR

```
Select Roll.no, Name, Branch, city, marks, phone no.
FROM Student;
```

Write a query to display roll.no, name & Branch of all the students.

```
Select Roll.no, name, Branch
from student;
```

Write a query to display roll.no, name, branch, city of all the students. Give the columns suitable heading.

```
Select RollNO as "Student ID", Name as "Stu Name",
Branch as "Course", city as "Address".
from student;
```

TYPE 2 Queries Based on Relational Operator.

like (<, <=, >, >=, =, <> .

Date

Write a query to display data of students of CS Branch.

```
Select *
from Student
where branch='CS';
```

Write a query to display roll.no, name, branch and city of all the students of CS or IT Branch.

```
Select roll.no, name, branch, city
from Student
where branch='CS' OR branch='IT';
```

Write a query to display roll.no, name, branch and city of all the students of CS or IT branch who belong to Gwalior or Bhopal.

```
Select roll.no, name, branch, city
from student
where (branch='CS' OR branch='IT') AND
(city='Gwalior' OR City='Bhopal').
```

Write a query to display roll.no, name, Branch and marks of all the students who have secured more than 70 marks.

```
Select Roll.no, name, Branch, marks
from student
where marks > 70
```

Write a query to display roll.no, name, Branch, marks and city of all the students of

DD-MON-RR ← Oracle Default format for DOB

Date

CS or IT Branch who have secure more than 70 marks

```
select roll.no, name, Branch, marks, city
    from Student
    where (Branch = 'CS' or Branch = 'IT') AND
        (MARKS > 70).
```

Type-3 Queries based on Range Operator

BETWEEN AND → It is known as Range operator. It is use to specify a range of values. The limit values are also including in the range.

Write a query to display Roll Number, name, Branch, marks of all the students of CS or IT Branch who have secured marks from 75 to 90.

```
select Roll.No, name, Branch, marks
    from Student
    where (branch = 'CS' or branch = 'IT') AND
        Marks BETWEEN 75 AND 90;
```

OR

```
AND (marks >= 75 AND MARKS <= 90);
```

Write a query to display Roll, name, DOB & City of all the students who are born in the year 1997.

```
Select Roll.no, Name, DOB, City
    from Student
    where dob BETWEEN '01-JANUARY-1997'
        AND '31-DEC-1997';
```

Date

Type - 4 Queries based on Pattern Matching

For Pattern Matching like and not like are used.

'_' matches to any single character
'%' matches to any sequence of zero or more characters.

Write a query to display roll.no, name, branch, & city of all the students whose name starts with 'M'.

```
select roll.no., name, Branch, city
    from Student
    where Name like 'M%';
```

Write a query to display rollno., name, Branch & city of all the students whose names ends with 'A'.

```
select roll.no, name, Branch, city
    from student
    where Name like '%A' OR
        Name like '%a';
```

Write a query to display roll.no, name, branch & city of all the students whose name starts with M and ends with A.

```
select roll.no, name, branch, city
    from student.
    where Name like 'M%A';
```

Date

Write a query to display roll.no, name, branch & city of all the students whose names are 3 letters long.

```
select roll.no, name, branch, city
from student
where name like '___';
```

Write a query to display roll.no, name, branch & city of all the students having third letter 'N' in their name.

```
select roll.no, name, branch, city
from student
where name like '__N%';
```

Write a query to display all data of all the students who are born in 1997.

```
Select *
from student
where dob like '%1997';
```

Write a query to display all data of all stu. whose name contains '_'.

```
Select *
from student
where name like '%\_%';
```

Write a query to display all data of all stud. whose name contain %

```
Select *
from student
where name like '%\%%';
```

ESCAPE

Date

By default escape character is '\' but if we want to use different escape character then it can be use with escape keyword.

TYPE - 5 Queries based on Membership operator

IN / NOT IN are known as Membership operator because they are used for comparing a set of values.

Write a query to display roll.no, name, branch and city of all those students whose branch is 'CS' or 'IT' & who don't belong to 'Gwal' or 'Bhopal'.

```
select roll.no, name, branch, city
from student
where branch IN ('CS', 'IT')
AND CITY NOT IN ('Gwalior', 'Bhopal');
```

TYPE - 6 Queries based on Null Values

Any comparison involving NULL values is always false therefore for comparing NULL values two are 2 special operators are used
'IS NULL'
'IS NOT NULL'

Write a query to display roll. no, name, city & phone no. of all the students whose city is known but phone no is not known.

```
select roll. no, name, city, phone no.
from Student
where city IS NOT NULL
AND phone no. IS NULL;
```

Date

Fetching Data from Multiple Tables →

For fetching data from multiple Tables two two operations are used :-

1) Join operation
2) Sub query

Types of Join

Cross Join → It is equivalent to cartesian product operation of relational Algebra.

depositor

Cname	ac.no.
RAM	a-100
SHYAM	a-200
MOHAN	a-400

accounts

ac.no	bal	branchname
a-100	10000	Moran
a-200	20000	Lashkar
a-300	30000	Gol

Syntax →

```
Select *
from depositor CROSS JOIN Account
```

OR

```
select *
from depositor, accounts
```

Cname	dep acc.no	accounts acc.no.	Bal	Branchname
Ram	a-100	a-100	10,000	M
Ram	a-100	a-200	20,000	L
Ram	a-100	a-300	30,000	G
shyam	a-200	a-100	10,000	M
shyam	a-200	a-200	20,000	L
shyam	a-200	a-300	30,000	G
Mohan	a-400	a-100	10,000	M
Mohan	a-400	a-200	20,000	L
Mohan	a-400	a-300	30,000	G

Date

→ Cartesian Product is not usefull operation because it generates so many invalid tuples and also it is very time consuming.

→ If one table contain M tupples and other table contain N tupples then after cartesian product of both the tables, the no. of tuples generated will be (M*N).

Natural Join or Inner Join → If we want to fetch data from multiple tables then natural join is a most usefull operation.

→ For performing natural join between 2 tables or relations, the tables must be related with the help of foreign key i.e, they must have atleast one attribute in common.

→ Natural Join fetches only the matching tupples between 2 tables based on the equality condition on all the common attributes that's why it is also known as Inner Join.

→ To perform natural join with 2 tables atleast one join condition is required.

→ To perform natural join between n tables atleast (n-1) join conditions are required.

→ If the tables aren't related with the help of foreign key then natural join will be equivalent to cross join.

Date

Example →

depositor

Cname	acc.no
RAM	a-100
SHYAM	a-200
MOHAN	a-300

accounts

acc.no	branch	bal
a-100	Moran	10K
a-200	Lashkar	20K
a-500	Gwal	30K

depositor × accounts

Cname	d acc.no	a acc.no	branch	bal
RAM	a-100	a-100	Moran	10K
RAM	a-100	a-200	Lashk.	20K
RAM	a-100	a-500	Gwal	30K
SHYAM	a-200	a-100	Moran	10K
SHYAM	a-200	a-200	Lashk	20K
SHYAM	a-200	a-500	Gwal	30K
MOHAN	a-300	a-100	Moran	10K
MOHAN	a-300	a-200	Lashk	20K
MOHAN	a-300	a-500	Gwal	30K

depositor ⋈ accounts

Cname	d-acc.no	a-acc.no	branch	bal
RAM	a-100	a-100	Moran	10K
SHYAM	a-200	a-200	Lashkr	20K

```
Select *
from depositor d, accounts a
where d.accno = a.accno
```

OR

```
Select *
from depositor d NATURAL JOIN
accounts a
```

OR

```
select*
from depositor d JOIN accounts
a
ON d.accno = a.acc.no
```

OR

```
select *
from depositor d JOIN accounts
a
using(accno)
```

Date

Outer Join → It is an extension of join operation which avoids loss of information contain.

Outer Join fetches both matching as well as non-Matching tuples.

It is generally used by higher level managment for policy making.

For performing outer joins between 2 tables, the tables must be related with the help of foreign key.

Based on the data fetched, outer Join is of three types:-

- Left outer Join
- Right Outer Join
- Full or Symmetric Outer Join.

Left outer Join → It fetches all the tuples from left handside table but only matching tuples from right handside tables. The missing information is represented with the help of null values.

Syntax →

```
select *
from depositor d LEFT OUTER JOIN accounts a
ON d.accno = a.accno
```

OR

```
select *
from depositor d, accounts a
where d.accno = a.accno (+).
```

Date

depositor ⋈ accounts

cname	d.accno	a.accno	branch	bal
RAM	a-100	a-100	Moran	10K
SHYAM	a-200	a-200	Lashkar	20K
MOHAN	a-300	NULL	NULL	NULL

Right outer Join → It fetches all the data from right handside tables but only matching tuples from ~~right~~ left handside tables.
The missing info is represented with the help of null values.

Syntax →

```
Select *
from depositor d RIGHT OUTER JOIN accounts a
ON d.accno = a.accno
```

OR

```
Select *
from depositor d, accounts a
where d.accno (+) = a.accno
```

Match side containing (+)

depositor ⋈ accounts

c.name	d.accno	a.accno	branch	Bal
RAM	a-100	a-100	Moran	10K
SHYAM	a-200	a-200	Lashk.	20K
NULL	NULL	a500	Gwl	30K

Full outer Join →
It fetches all the tuples from both the table.
The missing info. is represented with the help of null values :-

Date

Syntax →

```
Select *
from depositer d FULL OUTER JOIN accounts a
ON d.accno = a.accno
```

OR

```
Select *
from depositor d , accounts a
where d.accno(+) = a.accno(+).
```

depositor ⋈ accounts

Cname
RAM
SHYAM
NULL
MOHAN

Q: Consider a fro schema

Customer (cname, cstreet, ccity)
branch (bname, bcity, assets)
loan (loanno., bname, amt)
account (accno, bname, bal)
depositer (accno, cname)
borroner (loan no., cname)

① Find the info. of all loans over rs. 12K.

```
select *
from loan
where amt > 12k
```

② Find loan no. for each loan within amt greater than 12 lakhs.

Date

select loan no.
from loan
where amt > 12,00,000.

(3) Find the info. of all customers who live in Gwl.
select *
from customer
where ccity='Gwl'

(4) Find the name of the customers who live in Gwl
select cname
from customer
where c-city = 'Gwl'.

(5) Find acc.no, branchname of all the accounts having bal > 10,00,000.
select accno, bname
from account
where bal > 10,00,000.

(6) Find customer name, city, accno. for all the customers.
select c.cname, ccity, accno
from customer c, depositor d
where c.cname = d.cname

or

select c.cname, ccity, accno
from customer c NATORAL JOIN depositor d

(7) Find customer name, loan no. and account of all the customers.
borrower
select c.cname, l.loanno, amt

Date

from loan l, borrower b
where l.loanno. = b.loanno.

OR

select l.loanno., cname, amt
from loan l NATURAL JOIN borrower b

(8) Find customer name who have deposited more than 10,00,000.

select cname
from account a, depositor d
where a.accno = d.accno.
AND ~~bal~~ bal > 10,00,000

(9) Find customer name & city of customers who have deposited more than 10,00,000.

depo.
custome.

select c.cname, ccity
from customer c, depositor d, account a
where a.accno = d.accno
AND c.cname = d.cname
AND bal > 1000000

OR

Select c.cname, ccity
from account a NATURAL JOIN depositor d
NATURAL JOIN customer c
where bal > 1000000

(10) Find the name of all the customers who have taken loan from Morar branch of an amt greater than 10 lakkhan and are living in Gwal,

Date

Loan
borrower
Customer

```
select C.Cnance
from loan l , borrower b, customer c
    where l.loanno = b.loanno.
        AND c.cname = b.cname
            AND amt > 1000000
            AND ccity = 'Gwl'.
            AND bname = 'Morar'
            OR
```

```
Select C.cname
from loan l NATURAL JOIN borrower b
    NATURAL JOIN customer c
    where amt > 1000000
        AND city = 'Gwl'
        AND bname = 'Morar'.
```

Q:- Consider the fno schema & answer the fno ques:-

employee (ename, street, city)
works (ename, companyname, salary)
company (companyname, city)
Manager (ename, manager name)

(1) Find the name of all employees who works for XYZ comp.

```
select ename from works
    where companyname = XYZ
```

(2) Find the names & cities of residence of all the employees who works for XYZ company.

```
Select e.ename, city
    from works w, employee e
    where w.ename = e.ename
        AND companyname = 'XYZ'
```

(3) Find the names & cities of residence of all the employees who works for companies located in GWL.

Date

```
select e.ename, e.city
from employee e, works w, company c
where e.ename = w.ename
AND w.companyname = c.companyname
AND c.city = "Gwl".
```

(4) Find the name, city, street of residence of all the employees who works for XYZ company and earn more than 1,Lakh per month.

```
select e.ename, city, street
from employee e, works w
where e.ename = w.ename
AND companyname = 'XYZ'
AND salary > 1 Lakh
```

(5) Find the names of the employees who live in the same city as the company for which they work.

```
select e.ename
from employee e, works w, Company c
where e.ename = w.ename
AND e.city = c.city
AND e.companyname = c.companyname
```

(6) Find the names and cities of residence of all the Managers

```
select e.ename, city
from employee e, Manager m
where e.ename = m.managername
```

Date

(7) Find the names & cities of residence of all the managers who works for XYZ company

```
select e.ename, city
from ~~select~~ employee, works w, manager m
where e.ename = m.managername
  AND companyname = 'XYZ'.
  AND w.ename = m.managername
```

(8) Find the name of the employees who aren't managers

```
select e.ename
from employee e, manager m
where e.ename <> m.managername.
```

(9) Find the name of the employees who work for managers living in Goa.

```
select m.ename
from employee e, manager m
where e.ename = m.managername
  AND e.city = 'Goa'
```

(10) Find the name of the employees who lives in the same city and street as do their manager.

```
select w.ename
from employee e, manager m, employee w
where e.ename = m.managername
  AND m.ename = w.ename
  AND e.city = w.city
  AND e.street = w.street.
```

Date

Q: Consider the fno schema and answer the flw queries

supplier → S(S#, sname, city, status)

part → P(P#, pname, color)

J(J#, Jname, Jcity)

SPJ(S#, P#, J#, qty)

① Get supplier number for supplies who supply for project J_1.

```
Select S#
from SPJ
where J# 'J1'.
```

② Find supplier name & city of supplier who supply for project J_1.

```
Select sname, city
from SPJ S1, S S2
where S1.S# = S2.S#
AND J# = 'J1'.
```

③ Get partno & partname for parts supplied by suppliers in Gwl.

```
select P.P#, pname
from S S1, P S2, SPJ S3
where S3.S# = S1.S#
AND S3.P# = S2.P#
AND S1.city = 'Gwl'
```

④ Get part no, & part name for projects supplied by suppliers in Gwl who have project in Gwl.

Date

select p.p#, pname
from s, p, J, spJ
where S.s# = spJ.s#
AND p.p# = spJ p#
AND J.J# = spJ J#
AND s.city = 'lwl'
AND J.jcity = 'lwl'

(5) Get part no, part name for parts supplied to a project by a supplier in the same city as that of the project.

select p.p#, pname
from s, p, J, spJ
where s.s# = spJ.s#
AND p.p# = spJ.p#
AND J.J# = spJ.J#
AND S.city = J.jcity

(6) Get project names for which supplier is supplying

select Jname
from J, spJ
where spJ.J# = J.J#
AND S# = S1.

(7) Get colors of parts supplied by supplier 1.

select color
from p, spJ
where spJ.P# = P.P#
AND S# = S1.

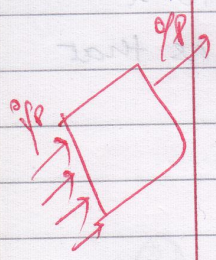
Date

Aggragate functions, Group by clause & Having clause →

→ Aggragate functions or group functions are the functions which operate on group of values.

→ They always take multiple values as input as produce single value as output.

→ Following are some of the popular group functions :-



Min → It returns minimum value from a group of values.

Max → It returns maximum value from a group of values.

Sum → It returns total of all the values in the group.

Average → It finds the average of group of values.

Count → It returns count of values in the group.

(Min-Max → Numeric/Non-Numeric) (Sum/Avg → Numeric)

Characteristics of Aggragate func^n →

→ Min and Max works with numeric as well as non-numeric data whereas Sum and average work with only numeric data.

→ Aggragate functions work on a single column and produces single column as o/p. They never take multiple independent columns.

→ Aggragate functions work with simple mathematical

Date

functions but not with complex mathematical functions.

→ There is no affect of distinct Keyword in min-max functions.

→ If select list contains both aggragate & non-aggragate columns, then groupby clause must be use and all non-aggragate column must appear in groupby clause, otherwise syntax error will be generated.

→ Aggragate functions cann't be used in where clause, groupby & orderby clause.

→ All the aggragate functions ignore null values.

Groupby Clause → It is use to group the tupples of a table on the basis of specific attributes.

→ Groupby clause must not contain aggragate functions.

Having Clause → It is use to restrict the groups in the output i.e, to apply condition on group.

→ There is no significance of using having clause without Groupby clause. Although no syntax error will be generated.

Aggregate func → ✗ Groupby, orderby, where
✓ Having

Date

Difference between →

	Having clause (O/P)	where clause (I/P)
→	It is use to restrict the groups in the O/P.	It is use to restrict the tuples in the O/P.
→	The condition of Having is applied on groups.	The condition of where is applied on individual tuple.
→	Aggragate functions are allowed in having	Aggragate functions aren't allowed in where.
→	There is no significance of using Having with-out groupby clause although no syntax error will be generated.	There is no relation between groupby & where clause.

employee (eno, ename, salary, dept)

eno.	ename	salary	dept.
101	Ram	10,000	10
102	Shyam	15,000	20
103	Mohan	2,0000	30
104	Ganesh	25,000	30
105	Suresh	30,000	20
106	Mahesh	35,000	10
107	Sita	40,000	10
108	Radha	15,000	20
109	Gita	10,000	30
110	Kavita	8,000	30
111	Rekha	18,000	20
112	Monika	20,000	10