SQL

Structured Query Language :

- Data Definition Lang. (DDL) — modification allowed at schema (definition) level
  - CREATE TABLE
  - DROP TABLE
  - ALTER TABLE
- Data Manipulation Lang. (DML) — mod. at data level
  - INSERT
  - UPDATE
  - DELETE
- Data Control Lang. (DCL) — control transactional operations
  - Commit
  - ABORT/ROLLBACK
  - TCC (Transaction Check Consistency Point)
- Data Query Lang. (DQL)
  - used to retrieve data from DB
    - SELECT
    - FROM
    - WHERE etc....

SELECT DISTINCT A1, A2, ... AN
FROM : R1, R2, R3, ... Rm
WHERE P

} SQL

SELECT → $\Pi_{A1,A2,\ldots An}$ FROM → $\sigma_{P}(R_1 \times R_2 \times R_3 \times \ldots \times R_m)$ WHERE

} R.A

Q Supplier (Sid, Sname, Rating)
Catalog (sid, Pid, cost)
Parts (Pid, Pname, Color)

Q Write a relational algebra query to retrieve Sname of Suppliers who supplied RED colored parts)

—Soln

$$\Pi_{Sname}\left(\sigma_{\substack{S.sid=C.sid \wedge \\ C.pid=P.pid \wedge \\ Color=RED}}(Supplier \times Catalog \times Parts)\right)$$

R.A

SQL — Supplier S or Supplier As S — Renaming Operator [Space, As]

```
SELECT Sname
FROM Supplier S, Catalog c, Parts P
WHERE s.sid = c.sid and c.pid = p.pid
and color = RED ;
```

} SQL

Difference b/w SELECT (SQL) and $\pi_A(R.A)$

SELECT A : not going to eliminate duplicate values

π_A : ——— result distinct rows always.

Example

R

A	B	C
1	2	3
1	2	4
2	3	5

$\Pi_{AB}(R) \Rightarrow$

A	B
1	2
2	3

SELECT A,B FROM R $\Rightarrow$

A	B
1	2
1	2
2	3

SELECT DISTINCT A,B FROM R

DISTINCT SELECT $\equiv \pi_A$

: Retrieve sid of supplies, who supplied some red part or green part.

$$\pi_{sid}\left(\sigma_{c.pid=p.pid \wedge (color=RED \vee color=GREEN)}(Catalog \times Parts)\right)$$

(join two table)

Relational Algebra

SQL

```
SELECT  sid
FROM    Catalog c, Parts p
WHERE   c.pid = p.pid AND
        (col = RED OR col = GREEN)
```

Sid of Supplier who supplied some RED and some green part

$$\pi_{sid}\left(\sigma_{col=RED}(Catalog \bowtie Parts)\right) \cap \pi_{sid}\left(\sigma_{col=green}(Catalog \bowtie parts)\right)$$

```
SELECT sid
FROM Catalog C1, Parts P1
WHERE C1.pid = P1.pid AND COL = RED
INTERSECT
SELECT sid
From Catalog C2, Parts P2
WHERE C2.pid = P2.pid AND
      COL = GREEN
```

BASIC SQL CLAUSES

(optional clauses)

(1) SELECT [DISTINCT] A1, A2, A3, ... AN
FROM R1, R2, R3, ... RM
[WHERE P]
[GROUP BY Attribute [HAVING Condition]]
[ORDER BY Attribute [DESC]]

Execution of query

(1) FROM clause — Query execution start from FROM clause, it is the cross product simply.

(2) WHERE clause — second executable clause, select tuples based on specified condition (predicate condition P).

(iii) **GROUP BY** : third executable clause, if used in query.

It groups the table based on the specified attribute.

Sid	Branch	marks
S1	CS	90
S2	IT	70
S3	CS	70
S4	EC	45
S5	CS	NULL

GROUP BY → BRANCH

Sid	Branch	marks
S1	CS	90
S3	CS	70
S5	CS	NULL
S2	IT	70
S4	EC	45

Aggregation Operations

operators

1. COUNT ([DISTINCT] Attribute)
2. SUM ([DISTINCT] Attribute)
3. AVG ([DISTINCT] Attribute)
4. MIN (Attribute)
5. MAX (Attribute)

Always DISCARDS NULL Values

⇒ Count (Marks) = 4 —— ∵ one NULL entry exist there.

⇒ Count (*) = 5 — No. of rows in relation

⇒ Count ([DISTINCT] marks) = 3

⇒ Sum (marks) = 275

⇒ Sum ([DISTINCT] marks) = 205

⇒ $AVG(marks) = \frac{Sum(marks)}{Count(marks)} = \frac{275}{4}$

⇒ $AVG([DISTINCT]\ marks) = \frac{Sum([DISTINCT]\ marks)}{Count([DISTINCT]\ marks)} = \frac{205}{3}$

Arithmetic Operation with NULL result is NULL

NULL + 5 = NULL

Gate

R

Sid	marks
S1	5
S2	10
S3	10
S4	NULL

Operations performed are

(1) Update R, set marks = marks + 5

(2) SELECT AVG of marks from R.

(1) Update R set marks = mark + 5

S1	10
S2	15
S3	15
S4	NULL

(2) AVG = $\frac{\text{Sum(marks)}}{\text{Count(marks)}} = \frac{40}{3} = 13.33$

11/11

SQL:-

SELECT [DISTINCT] A1, A2, ... AN

1) FROM R1, R2, ... RN ——— ≡ Cros Product
2) [WHERE P] ——— ≡ Selection of tuples
3) [GROUP BY Attribute ——— ≡ Grouping of data
4) [HAVING Condition]]

[ORDER BY Attribute [DESC]]

Sid	Branch	mark
S1	CS	66
S2	IT	70
S3	CS	90
S4	IT	60
S5	EC	50
S6	EC	NULL

Group BY (Branch)

S1	CS
S3	CS
S2	IT
S4	IT
S5	EC
S6	EC

1) HAVING Clause :- fourth executable clause. (If used) always precedeed by GROUP BY

- used to select the groups those which satisfy condition.
- Condition is for each group (checked in each group)

⇒ Student whose (Branch) group Avg marks greater than 60

⇒

```
Select *
FROM Student
GROUP BY (Branch)
    HAVING AVG(marks) > 60
```

→ Condition applied for every group in the table

S1	CS	60
S2	CS	70
S3	IT	60
	IT	70

— Group1
— Group2

(A) #② Select min(marks) ⟹ 50
① From student

(B) #③ Select min (marks) ⟹ 60
① From student
② WHERE Branch='CS'

(C) #③ Select min(marks) ⟹ Whenever group BY exist, then aggregation [min(marks)] done for every group.
① From student
② GROUP BY (Branch)

60	CS
60	IT
50	EC

(D) SELECT AVG(min(marks)) ⟹
① From student
② GROUP BY (Branch)

min(marks): 60, 60, 50 ⟹ Avg(min(marks)): 60, 60, 50 #

In SQL, multiple aggregation ≡ innermost aggregation

AVG(min(marks)) ≡ min(marks)

min(AVG(marks)) ≡ AVG(marks)

(E) Select branch, min(marks) ⟹ min(marks): 50 | Branch: CS, IT, CS, IT, EC, EC ✗
From student

Such syntax not allowed in SQL Language.

↳ Note:- If aggregation operator and other attribute used in SELECT clause is allowed only if other attribute should be in GROUP BY clause.

Select branch, min(marks) From student ✓ allowed

— SELECT Branch, min(marks) ③
① FROM student
② GROUP BY (Branch)

CS	60
IT	60
EC	50

2) SELECT min(A), B
FROM R
GROUP BY (C)
} X not allowed syntax error

SELECT min(A), C
or
GROUP BY (B) then allowed

3) Retrieve Sid of students who secured highest marks
& marks

① SELECT Sid, max(marks)
FROM student
} X

② SELECT Sid, max(marks)
From student
GROUP BY (sid)
} X

③ Nested Queries

Select max(marks) FROM student
90 ⇒ o/p

SELECT Sid, Marks
FROM STUDENT
WHERE marks = (Select max(marks) from student)

» inner query — do aggregation
» outer query — select desired tuples corresponding to aggregation.

» ORDER BY Clause: order the o/p based on specified attributes.

SQL SET OPERATORS

UNION / UNION ALL

INTERSECT / INTERSECT ALL

MINUS / MINUS ALL

⇑ followed by Relational Algebra

⇑ not supported by Relational Algebra

{ R∪S, R∩S, R−S } } R and S both should be union/intersect/minus Compatible

» Consider two relations

R

A
1
1
1
2
2
4
6
6

S

B
1
1
2
4
5
5

① RUS (R UNION S)

└ result distinct tuples

1
2
4
6
5

② R UNION ALL S

└ result all values

1
1
1
2
2
4
6
6
1
2
4
5
5

③ R ∩ S
↑ distinct common tuples from bon R&S

1
2
4

④ R INTERSECT ALL S
↑ How many max. no of times common in both R&S

1
1
2
4

⑤ R − S (R SET DIFFERENCE S)
↑ distinct tuples from R those are not there in S

6

⑥ R MINUS ALL S
↑ # duplicates in R − # duplicates in S

1
2
6
6

OTHER SET OPERATIONS

1. IN / NOT IN
2. ϕ ANY
3. ϕ ALL
4. EXISTS / NOT EXISTS

ϕ : comparison operator: <, >, <=, >=, =, <> (<> : not equal)

↳ Mostly used in Nested queries.

Nested Queries

- **Independent Nested Query**
  - inner query independent of outer query
  - IN
  - ANY
  - mostly
- **Co-related Nested Query**
  - ↳ Inner query uses attribute defined in outer query.

IN NOT IN

Supplier (Sid, Sname, rating)
Parts (Pid, Pname, Color)
catalog (sid, pid, Cost)

(1) Retrieve sid of the supplier who supplied some RED color part.

(a) SELECT Sid
FROM Catalog c, Parts p
WHERE p.pid = c.pid AND
Color = RED

OR

(b) SELECT Sid
(iii) FROM CATALOG
WHERE pid = (Select Pid
(ii) P1 ... From Parts
P1 P3 where color = 'Red')
(i)

=, <, >, >=, <=, .. etc
Compare $C_1 = C_2$
one constant with another but here
P1 .... P1 P3
one value compared to set of values P1 P3
NOT Allowed

Sid	Pid	Cost
S1	P1	
S1	P2	
S2	P3	
S4	P4	

Pid	Color
P1	R
P2	G
P3	R
P4	G

SO

```
SELECT Sid
FROM Catalog
WHERE pid IN (Select Pid From Parts
              WHERE Color = 'Red')
```

⇒

S1
S2

↳ If NOT IN used then o/p is

S1
S4

NOT IN — Compliment of IN

(c) Co-related nested query:

Inner query using attribute defined in outer query.

```
SELECT c.sid
FROM catalog C
where EXISTS ( Select * from Parts P where p.pid=c.pid
                                           Color='Red' )
```

↳ execute similar to nested for loop.

```
for (i=1 to n)
  {
  for (j=1 to m)
    {
    }
```

i=1 → j = 1 2 ... m
i=2 → j = 1 2 ... m
i=n → j = 1 2 3 ... m

NORMAL NESTED QUERY

Bottom → Top

CORELATED NESTED QUERY

Top → Bottom → Top

EXISTS :- (checks)
↳ returns True if inner query results non-empty.

NOT EXISTS :- returns True if inner query results empty.

Query Execution

Catalog

Sid	Pid
S1	P1
S1	P2
S2	P3
S4	P4

Parts

Pid	Color
P1	R
P2	G
P3	R
P4	Y

* — (I) Iteration

(S1) (P1) — Catalog

Compare this Pid with — Parts: Pid — one tuple selected.

(P1) ::: P1 ::: Red

P2

P3

P4

* — (II) Iteration — Zero tuple selected

(S2) (P2) — P.pid = c.pid AND Color = 'Red'

P2 = P2 . P2 color ≠ Red

* — (III) Iteration — one tuple selected

(S2) (P3) — P.pid = c.pid. Color = Red

P3 = P3 P3 color = Red

* — (IV) Iteration — Zero tuple selected.

(S4) (P4) — P4 = P4 P4.color ≠ Red

Final output

sid
S1
S2

⇒ No attribute before EXIST required

WHERE Eid EXISTS (select * from Parts where P.pid = c.pid) } ✗

⇒ Before IN attribute required

ANY

Emp

	Eid	dno	Salary
✓	E1	5	40
X	E2	3	10
X	E3	5	20
✓	E4	3	30
✓	E5	4	50

> 20,40

⇒ Retrieve Eid's who gets more salary than any employees of dept 5.

i.e Eids of Emp whose salary > 20,40

20 or 40

⇒ Inner: SELECT Sal FROM Emp WHERE deptt no=5 ⇒ 20, 40

⇒ How check every employee's salary one by one with inner query o/p.

```
SELECT Eid
From Emp
where Sal > (Select sal FROM Emp
             where deptt no=5)
```

X ∵ Inner query returns more than one value. So one to many comparision is not possible

```
SELECT Eid
FROM Emp
WHERE Sal > ANY (Select Sal FROM Emp
                 where deptt no=5)
```

✓

(R)

```
SELECT Eid FROM Emp
WHERE Sal > (Select min(sal) FROM Emp
             where dept =5).
```

✓

AR ∵ inner query returns single value [SA]

ϕ ALL

⇒ Retrieve sid from Emp whose salary greater than all employees of deptt 5.

⇒
```
SELECT Eid
FROM Emp
WHERE Sal >ALL (Select sal FROM Emp
                WHERE dno=5)
```

```
SELECT Eid
From Emp
WHERE Sal > (select max(sal)
             FROM Emp
             WHERE dno=5)
```

Question:

Given Emp (Eid, Ename, sal)

Dependent(Eid , dname , dage)

" Retrieve Employees who have no dependent"

[Q1]
```
SELECT E.Eid
FROM Emp E, Dependent D
where E.Eid <> D.Empid
```

[Q2]
```
SELECT Eid
FROM Emp E
where NOT EXISTS (Select *
                  From dependent D
                  WHERE D.Empid= E.Eid)
```

Which is TRUE ?

a) Q1 correct but not Q2

b) Q2 " " " Q1

c) Q1 and Q2 correct

d) Q1 and Q2 incorrect.

Soln Take some sample data.

Emp

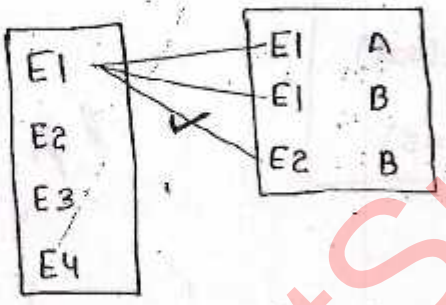
Eid
E1
E2
E3 ✓
E4 ✓

Dependat

Empid	dnam
E1	A
E1	B
E2	B

E3 and E4 should be the output.

[Q1] FROM ——— CROSS PRODUCT



E1 <> E1 — F
E1 <> E1 — F
E1 <> E2 — True

E2 <> E1 — T
E2 <> E1 — T
E2 <> E2 — F

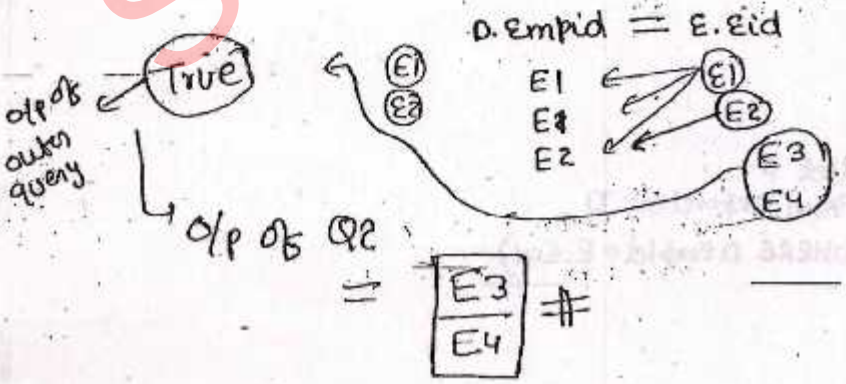
E3 <> E1 — T
E3 <> E1 — T
E3 <> E2 — T

E4 <> E1 — T
E4 <> E1 — T
E4 <> E2 — T

So o/p of [Q1]

E1
E2
E2
E3
E3
E3
E4
E4
E4

[Q2]



NULL

- non-zero value
- no two null values are equal
- (unknown / Unexisted)

Employee

Eid	Ename	PPno
E1	A	10
E2	B	NULL
E3	B	5

⇒ Retrieve Eid's who doesn't have any passport details.

```
Select Eid
FROM Emp
WHERE PPno = NULL
```

✗ ∵ No two null values equal

For Comparison with NULL, SQL supports

IS / IS NOT clause

```
Select Eid
FROM Emp
Where PPno IS NULL
```

✓ no Passportno

```
where ISNOT NULL
```

✗ have some Passport no.

Regular Expressions

'%' ⇒ Zero or more characters

'_' ⇒ Exactly any one character

⇒ Retrieve students whose Name starts with D and ends with A and atleast 5 character

```
D_ _ _ % A
```

» Starting with D ⟹ 'D%'

» Name containing underscore['_'] ⟹ ⟶ For this '_' should not be used as reg. expression symbol.

» '%_%' ⟹ atleast one character

» Name Containing '_'

= '%_%'

⇓

For this we use ESCAPE character ('\') (Backslash)

Then

```
» Select *
  From student
  where sname = 'D___%A'
```

o/p = 'D___%A' — such pattern is the o/p

↳ will not return names starting with D & ending with A, & containing atleast 5 character.

```
Select *
From student
where sname LIKE 'D___%A'
```

```
Select *
From student
where sname NOT LIKE 'A%'
```

All names not starting with A.

Pg27

(2) π_{name}(employee) — π_{name}(employee ⋈ works_on)

(All Emp)
0 or more projects

Emp who works for any project
1 or more project

Employees with no project

(3) All addresses of emp. working in project P.name = "database"

⇒ All three tables required.

```
Select Addresses
From   Emp E, Proj P, Workon W
WHERE  E.no = W.no AND
       P.Pno = W.Pno AND
       Pname = 'DB'
```

Q:-

Prim Key → P#
P1
P2
P3

Update P3 to P4 ✓ — still unique

Update P3 to P2 ✗ — duplicate

SQL

R X S ⇒ Cross Join
R ∪ S ⇒ Union Join
R ⋈ S ⇒ Inner Join
R ⟕ S ⇒ Outer Join

(b) 1. Cross join equal to an inner join without join condition

Inner Join = Natural Join.

$R \bowtie S \Longrightarrow R(AB) \quad S(CD)$

$$R \bowtie S = \Pi_{ABCD}(\sigma_{*}(R \times S)) \# \equiv R \times S$$

R

A	B
1	2
2	1

S

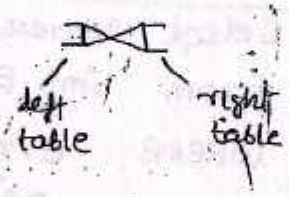
A	B
3	2
1	4

(Inner Join) $R \bowtie S$ = Empty

$R \bowtie S$ = Natural Join + attributes not selected or failed join condition

A	B
1	2
2	1
3	2
1	4

$\equiv$ RUS



⟹ If attributes are different then it is not allowed to perform union. (d)

(1) Relation can contain "NULL"

All comparisons with NULL treated as False

NULL = 5 — False

NULL ≠ 5 — False

NULL = NULL — False

⟹

X
NULL
5
4

x = NULL — False (not selected)

x = 5 — 5 = 5 ✓ Selected

x = 4 — 4 = 5 ✗ — Fail

(5) only one value selected

not (not (x=5))

↓

not (not (null=5))

not (not (False))

not (True) = False

NULL ✗

not (not (5=5))

not (not (True))

not (False)

True

⇓

5

Selected

not (not (4=5)

not (not (False))

not (True)

False

4 ✗ not selected.

$x = 5$	not (not (x=5))	Equivalent
⇓	⇓	
only 5 selected	only 5 selected	

Option c

LHS

x = NULL

NULL ≠ 5

⇓

False

x=5

5≠5

⇓

False

x=4

4≠5

⇓

True

4 selected

x: 4

RHS

not (x=5)

not (null=5)

not (False)

True

Null selected

not (5=5)

not (True)

False

not (4=5)

not (False)

True

4 also selected

x: null, 4

NOT EQUIVALENT

4/11/21

(16) FROM means cross product

(Student ⋈ Course ⋈ Grade)

S.sno = G.sno AND
C.cno = G.C.no AND
↑ for this join

Grade = A and
Instructor = "Korth"

$$\Pi_{name}\left(\sigma_{\substack{\text{Grade A} \wedge \\ \text{Instructor}=\text{"K"}}}(\text{Student} \bowtie \text{Course} \bowtie \text{Grade})\right)$$

OR

All Grade A

R(A,B) / S(B)

$$= \Pi_A(R) - \Pi_A\left(\Pi_A(R) \times S - R\right)$$

All Red Part

$$\Pi_{sid,pid}(\text{catalog}) \,/\, \Pi_{pid}\left(\sigma_{Col=Red}(\text{Parts})\right)$$

Some Red Parts

$$\Pi_{sid}\left(\sigma_{Col=Red.}(\text{Catalog} \bowtie \text{Parts})\right)$$

(19) Select Title from book as B
WHERE
(Select Count *
FROM Book as T
where T.price > B.price) < 5

Book

(B)

Title	Price
C	5
C++	10
Java	20
SQL	40

(T) Another Instance

Title	Price
C	5
C+	10
Java	20
SQL	40

T.price > B.price

40
20
10 (5)

(3)

Title	Cost
A	10
B	20
C	30
D	40
E	5
F	50
G	60

Top → down → top

A 10 B.price = 10 T.price = 10

T.price > B.price

10 — X
✓ 20
✓ 30
✓ 40
5 — X
✓ 50
✓ 60

5 values

(5) < 5 = False

WHERE FALSE → Tuple ~~select~~ selected not in answer

B 20 B.price = 20

T.price = 20

10 — X
20 — X
30 ✓
40 ✓
5 — X
50 ✓
60 ✓

(4) tuple

(4 < 5) (B selected)

C | 30 B.price = 30 T.price = 30

(3 < 5)

where TRUE

↳ C selected

10 X
20 X
30 X
40 ✓
5 X (3)
50 ✓
60 ✓

D | 40 B.price = 40 T.price = 40

2 < 5

D selected

10 X
20 X
30 X
40 X
5 X
50 ✓
60 ✓

E | 5 B. price = 5. T. price = 5

E not selected.

10 X
20 X
30 X
40 X
50 X
60 X
5 X

F | 60 B. price = 60 T. price 60

T.price < B.price

60 < 60

10 ✓
20 ✓
30 ✓
40 ✓
5 ✓
50 ✓
60

T. price > B. price ≤5 (5th most exp book)

T. price < B. price >5 (6th cheapest book)

T. price = B. price =5 (6th most exp. book)

(20)

Cid	Bal
A	40
B	40
C	30
D	30
E	30
F	20

⇒

	Rank
A	1
B	1 — 2 skipped
C	3
D	3 — 4 and 5 skipped
E	3
F	6

[Q1] : Select A. Customer, count (B.customer)
FROM account A, account B
WHERE A.balance <= B. Balance
GROUP BY A.customer

[Q2] : Select A.customer, 1+ Count (B.customer)
FROM account A, account B
WHERE A.balance < B.balance
GROUP BY A. Customer

⇒ To solve such questions, take some sample data and thoroughly execute the query.

Account A

Cust	Bal
P	40
Q	40
R	30
S	20

Acc.B

Cus	Bal
P	40
Q	40
R	30
S	20

⇒ FROM clause means Cross-PRODUCT (AXB)

(AXB)

Aid	Bal	Cid	Bal
P	40	P	40
P	40	Q	40
Q	40	P	40
Q	40	Q	40
R	30	P	40
R	30	Q	40
R	30	R	30
S	20	P	40
S	20	Q	40
S	20	R	30

o/p [Q1]

P	2
Q	2
R	3
S	1

If no two customer have same balance then [Q1] is correct.