

Relational Algebra

In Relational Model $\rightarrow$ database is collection of table

Query $\rightarrow$ Data retrieval

Given by Codd $\rightarrow$ Relation Algebra
Relation Calculus } theoretical languages

Query $\rightarrow$ Data retrieval

given by Codd $\rightarrow$ $\left. \begin{array}{l} \text{Relation Algebra} \\ \text{Relation Calculus} \end{array} \right\} \text{ theoretical languages}$

Very Important in Gabe

→ What is the difference btw algorithm vs function?

- ↳ Function → Input output relation
- ↳ Algorithm → Step by step recipe

} Analogy

① What you want

② How to get it

→ Relational algebra

- Machine point of view
 ↳ This language is better.

① What you want

- Relational calculus
- SQL

- User's point of view $\rightarrow$ This is easy

P1

Query processors
& Optimisers
QPO

Relational algebra
equivalent query

Optimise

Equivalent but
efficient Rel. algebra
query. Q_2

→ Procedural
Manner

- An algebraic query language for querying relational data based on “operators”
- Not used in commercial DBMSs
 - use “real” language SQL
- But SQL basically implements (extended) RA
- SQL query gets translated into RA for optimizations
- RA has well-defined meaning – builds the foundation of database queries in SQL

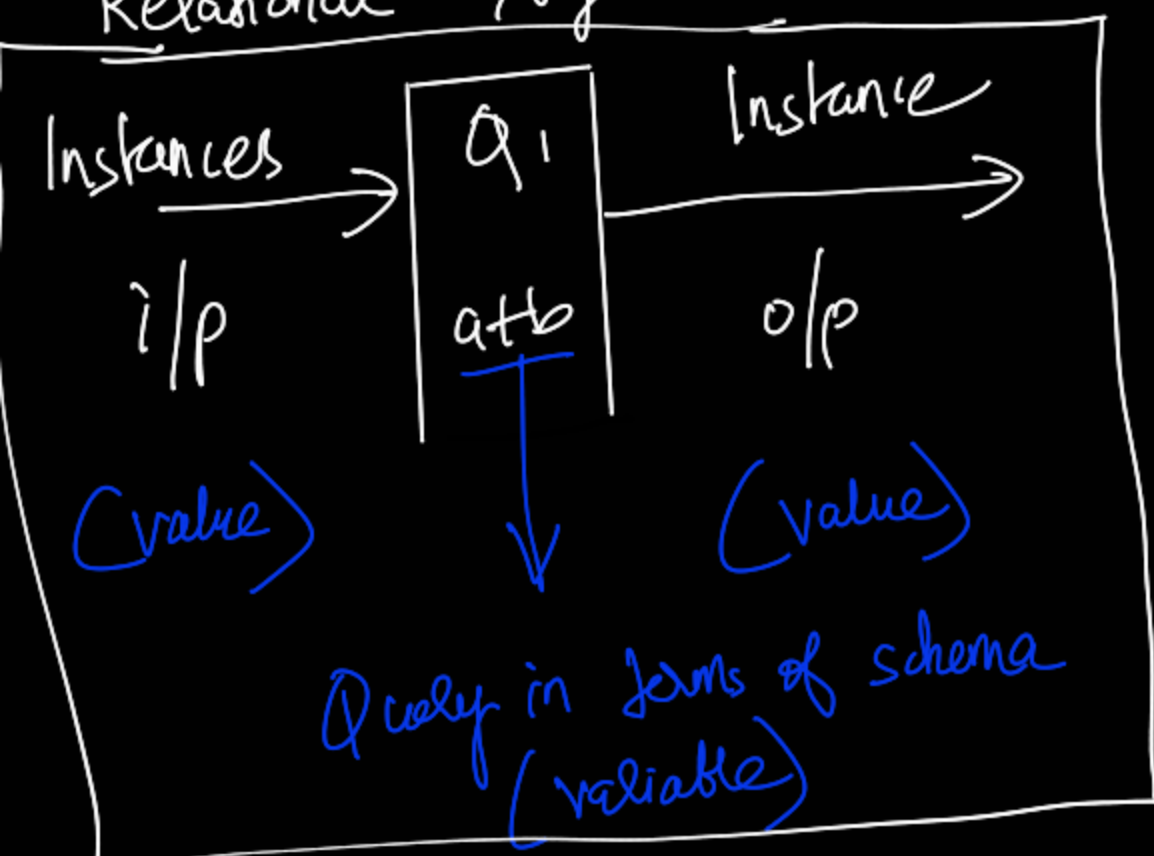
Relational Algebra

- Values — Values are instances
- Operators — Basic
Derived

Variable → we can write but cannot evaluate
for evaluate, we need instance (value)

Algebra	Relational Algebra
variable	Relation (Schema)
value	Instance
operator	operator

Relational Algebra



The inputs and outputs of a query are relations

A query is evaluated using instances of each input relation and it produces an instance of the output relation.

Queries often involve the computation of intermediate results, which are themselves relation instances

eg: - $R(a, b, c)$

⇒ R.1 → a refers attribute a by position

Operators

Unary operators → Take one give one
Binary operators → Take two give two

Procedural language	+919496862434
Queries in relational algebra are applied to relation instances, result of a query is again a relation instance	
Six basic operators in relational algebra:	
select	σ selects a subset of tuples from reln
project	π deletes unwanted columns from reln
Cartesian Product	$\times$ allows to combine two relations
Set-difference	$-$ tuples in reln. 1, but not in reln. 2
Union	$\cup$ tuples in reln 1 plus tuples in reln 2
Rename	ρ renames attribute(s) and relation
The operators take one or two relations as input and give a	

→ In Retrieval Queries the original database & original relations do not change for processing the query
→ Only temporary instances of relations may be created & they are removed, once the query is complete.

1) Selection operators

Symbol $\sigma_P(R)$

→ horizontal slice operator
→ Commutative
ie

$$\sigma_{P_1}(\sigma_{P_2}(R)) =$$

$$\sigma_{P_2}(\sigma_{P_1}(R)) =$$

$$\sigma_{P_1, P_2}(R)$$

P → conditions that evaluate the tuple
They act individually & independently on each tuple
Follows propositional logic

of the form

a) attribute <operator> attribute	$\wedge$ — and $\vee$ — or
b) attribute <operator> value	

R → relation on which to apply selection

Schema, degree → doesn't change
cardinality → May reduce

$$|\sigma_C(R)| \leq R$$

Selection example.

Query: list students with GPA > 3

Student		
Name	GPA	Country
Bob	3	Canada
John	3	Britain
Tom	3.5	Canada
Maria	4	Mexico



S		
Name	GPA	Country
Tom	3.5	Canada
Maria	4	Mexico

Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

select tuples

$\sigma_{A=B \wedge D > 5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Select Operation

- Notation: $\sigma_p(r)$
- p is called the **selection predicate**
- Defined as:

$$\sigma_p(r) = \{t \mid t \in r \text{ and } p(t)\}$$

Where p is a formula in propositional calculus consisting of **terms** connected by $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)

Each **term** is one of:

<attribute> op <attribute> or <constant>

where op is one of: $=, \neq, >, \geq, <, \leq$

- Example of selection:

propositional operators.



$\wedge$ — and
 $\vee$ — or
 $\neg$ — not

eg.

More on selection

User (uid, name, age, pop)
Member (uid, gid)

- Selection condition can include any column of R , constants, comparison ($=, \leq$, etc.) and Boolean connectives ($\wedge$: and, $\vee$: or, $\neg$: not)
- Example: users with popularity at least 0.9 and age under 10 or above 12

Ans

$$\sigma_{(pop \geq 0.9) \wedge ((age < 10) \vee (age > 12))}$$

eg

A Question:

Find the rows with maximum marks.

Student

sid	marks
a	2
b	3
c	4

$$\sigma_{\text{marks} \geq \# \text{marks}(\text{student})}$$

- Selection apply's individually & independently on any tuple
- So max operator cannot be represented by only selection
- While processing 1 tuple info on other tuple is not available.

Projection Operator

$\Pi_{A_1, A_2}(R) \rightarrow$ filters attributes / project attributes

$\rightarrow$ order of projection in query is the order of attribute in the query result

$\rightarrow$ Vertical slice operators

$\rightarrow \text{degree } \Pi(R) < \text{degree}(R)$

$\rightarrow$ schema changes

$\rightarrow$ Duplicates merged - If we want cardinality unchanged, then projected attributes must be a SK. Then $|\Pi_p(R)| = |R| \iff p \text{ is SK}$

eg

Example: Projection

Relation Sells:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

Prices := $\Pi_{\text{beer, price}}(\text{Sells})$:

beer	price
Bud	2.50
Miller	2.75
Miller	3.00

→ merged in projection

To make sure that Cardinality of Projected relation is necessarily same the input relation;

what can you say about the Attribute List of projection operation? → They should be SK.

Q : Let $R(a,b,c)$ be a RDBMS relation having the following set of functional dependencies(FD set) :

FD Set $F = \{a \rightarrow b, bc \rightarrow a\}$

We apply some queries on a relational instance r of R . Which of the following queries are guaranteed to return the same number of tuples as in relation instance r ?

- ① $\Pi_a(R)$ ② $\Pi_b(R)$ ③ $\Pi_c(R)$
 ✓ ④ $\Pi_{ac}(R)$ ⑤ $\Pi_{ab}(R)$ ⑥ $\Pi_{abc}(R)$

If the attribute list includes only nonkey attributes of R , duplicate tuples are likely to occur. The PROJECT operation removes any duplicate tuples, so the result of the PROJECT operation is a set of distinct tuples, and hence a valid relation. This is known as **duplicate elimination**. For example, consider the

The number of tuples in a relation resulting from a PROJECT operation is always less than or equal to the number of tuples in R . If the projection list is a superkey of R —that is, it includes some key of R —the resulting relation has the same number of tuples as R . Moreover,

Properties of Selection & Projection

① $\sigma_{p_1}(\sigma_{p_2}(R)) = \sigma_{p_2}(\sigma_{p_1}(R)) = \sigma_{p_1, p_2}(R)$ } → Selection is commutative, projection is not

② $\Pi_{A_1}(\Pi_{A_2}(R)) \neq \Pi_{A_2}(\Pi_{A_1}(R))$

③ $\Pi_Y(\Pi_Z(R)) = \Pi_Y(R)$ → if $Y \subseteq Z$

④ $\Pi_A(\sigma_P(R)) = \sigma_P(\Pi_A(R))$ → if P is a condition that only uses attribute set A .

⑤ $\sigma_P(\Pi_A(R)) \xrightarrow{\text{Transform}} \Pi_A(\sigma_P(R))$ → Since we know P acts on set of attributes A .

Selection before projection is efficient than projection before selection

transformation between selection and projection;

1. Cascading of selection: $\sigma_{C_1 \wedge C_2}(R) \equiv \sigma_{C_1}(\sigma_{C_2}(R))$
2. Commutativity of selection: $\sigma_{C_1}(\sigma_{C_2}(R)) \equiv \sigma_{C_2}(\sigma_{C_1}(R))$
3. Cascading of projection: $\pi_{\text{Atts}}(R) = \pi_{\text{Atts}}(\pi_{\text{Atts}'}(R))$ if $\text{Atts} \subseteq \text{Atts}'$
4. Commutativity of projection: $\pi_{\text{Atts}}(\sigma_C(R)) = \sigma_C(\pi_{\text{Atts}}(R))$ if Atts includes all attributes used in C .

3.13.3 Relational Calculus: GATE CSE 1998 | Question: 2.19 up

https://gateoverflow.in/1082

Which of the following query transformations (i.e., replacing the l.h.s. expression by the r.h.s expression) is incorrect? R_1 and R_2 are relations, C_1 and C_2 are selection conditions and A_1 and A_2 are attributes of R_1 .

A. $\sigma_{C_1}(\sigma_{C_2}(R_1)) \rightarrow \sigma_{C_2}(\sigma_{C_1}(R_1))$ ✓B. $\sigma_{C_1}(\pi_{A_1}(R_1)) \rightarrow \pi_{A_1}(\sigma_{C_1}(R_1))$ ✓C. $\sigma_{C_1}(R_1 \cup R_2) \rightarrow \sigma_{C_1}(R_1) \cup \sigma_{C_1}(R_2)$ ✓D. $\pi_{A_1}(\sigma_{C_1}(R_1)) \rightarrow \sigma_{C_1}(\pi_{A_1}(R_1))$ ✗

option D

Set Operations:

→ Since relation is a set of tuples - Some set operations are defined in Relation

Union (U)

R U S

→ final result will have schema of R

→ S U R → final schema is S.
→ Union is still commutative since in data retrieval (Query) we are only concerned about result tuple (data) not result schema.

→ Common tuple are not repeated.

$$|R \cup S| = |R| + |S| - |R \cap S|$$

Set difference (→) Basic

$$R - S = R \cap S^c$$

$$\rightarrow S - R \neq R - S$$

Intersection (∩)

R ∩ S

$$= R - (R - S) \therefore \text{Derived}$$

→ Relations must be compatible schema

OR Union compatible schema.

R is union compatible with S if

① #Attributes(R) = #Attributes of S
↳ degree/arity same

② Domain of ith attribute should be same for R & S for all i

Consequences →

→ Names can be different

Difference example.

Query: Who is registered in the Database course but not in the Algorithms?

RegisteredFor	
Name	Topic
Bob	Algorithms
John	Algorithms
Tom	Algorithms
Bob	Python
Tom	Python
Bob	Databases
John	Databases
Maria	Databases
John	GUI

First do some selections:

$$\pi_{\text{Name}}(\sigma_{\text{Topic}=\text{"Databases"}}(R)) - \pi_{\text{Name}}(\sigma_{\text{Topic}=\text{"Algorithms"}}(R))$$

Intersection example.

Query: Which courses are taught at both Universities?

Alright University		EvenBetter University	
Course	Topic	Course	Topic
	Algorithms		Algorithms
	Python		Java
	Databases		Databases
	GUI		Networks
			Human-Computer Interaction

R ∩ S

Cartesian product (X)

→ every tuple of R combine with every tuple of S

→ Need not be union compatible

→ if two tables have same attribute

$$\text{eg:- } R(a,b) \quad S(c,e,d)$$

$$R \times S = RS(a,b,c,e,d)$$

$$\text{eg:- } R(ab) \quad S(c,b,d)$$

$$R \times S \neq RS(abcb,d)$$

Convention 1
Convention 2
refer by position

→ We can't put same for different attributes in a relation.

Example: $R3 := R1 \times R2$

R1(A, B)	R2(B, C)	R3(A, R1.B, R2.B, C)
1 2	5 6	1 2 5 6
1 2	7 8	1 2 7 8
1 2	9 10	1 2 9 10
3 4	5 6	3 4 5 6
3 4	7 8	3 4 7 8
3 4	9 10	3 4 9 10

Properties of $R \times S$

- $|R \times S| = |R| \cdot |S|$
- $Arity(R \times S) = Arity(R) + Arity(S)$
- $R \times S$ is commutative
 - schema is different
 - Data is same.

Assignment operation

Temp1 $\leftarrow \sigma_P(R)$
 Temp2 $\leftarrow \pi_A(Temp1)$
 Assignment operators

Rename operation

→ Another copy is wanted

$\rho_S(R)$

→ Rename R as S
 → New copy will be created
 → Existing relation R is left unchanged.

eg:- $\rho_{S(a)}(R)$ → change relation name to S &
 change attr. name to (1,2)

$\rho_{(1,2)}(R)$ → Do not change relation name
 → change attr. name to (1,2)

$\rho_{S(a \rightarrow 1)}(R)$ → change relation name to S
 → change attribute a to 1

$\rho_{S(1,2)}(R)$

eg. $\rho_\phi(R-S) \rightarrow R-S$
 equivalent

T (uid1, uid2)	Find all true friends in twitter dataset
A → B	
B → A	
B → C	
A → C	
C → B	

A & B C & B

Idea

$\rho_{M(sid1, sid2)}(T)$

$F = T \times M$

$\pi_{uid1, uid2} \left(\sigma_{(uid1=sid2) \wedge (sid1=uid2)}(F) \right)$

How to decide, how many times to combine — like in the above example

→ Generally, if we need to select tuples based on their relation | criteria involving two pairs of tuples, then, we need to take the cross product of a relation with itself.

→ So count how many rows (tuples) we need to combine (access) to select (criteria) the required amount.

Twitter database $T(\text{uid1 follows uid2})$

$T(\text{uid1, uid2})$	
A	B
B	A
A	C
B	C
C	B

- a) People who A follows $\rightarrow \pi_{\text{uid2}}(\sigma_{\text{uid1}=A}(T))$
- b) People with min 2 followers $\rightarrow \pi_{\text{uid2}}(\sigma_{\text{uid2}=\text{sid2}}(T \times \rho_{\text{sid1, sid2}}(T)))$
- c) People with min 3 followers $\rightarrow \pi_{\text{uid2}}(\sigma_{\text{uid2}=\text{sid2}}(T \times \rho_{\text{sid1, sid2}}(T) \times \rho_{\text{fid1, fid2}}(T)))$
- d) People with at least one follower $\rightarrow \pi_{\text{uid2}}(T)$
- e) People with exactly 1 follower $\rightarrow \text{minTwo} \leftarrow \pi_{\text{uid2}}(\sigma_{\text{uid2}=\text{sid2}}(T \times \rho_{\text{sid1, sid2}}(T)))$
 $\text{atleastOne} \leftarrow \pi_{\text{uid2}}(T)$
 $\text{exactlyOne} \leftarrow \text{atleastOne} - \text{minTwo}$
- f) Pairs of people who follow each other $\rightarrow \pi_{\text{uid1, uid2}}(\sigma_{\text{uid1}=\text{sid2}}(T \times \rho_{\text{sid1, sid2}}(T)))$

→ Why projection after selection of cross products is very common.
 → But pairing every row with every row doesn't give useful information.
 → So usually cross product is followed by some selection & projection conditions.
 → Similarly, other derived operators are introduced, when some complex query that can be optimised in the algorithm is used frequently.
 → Remember, how we write the query may not be exactly same as the algorithm implementation. Often optimisation is used.

Basic operators	Derived operators
σ	$\bowtie$ — Inner join
π	$\div$ — Division
$\cup$	$\cap$ — Intersection
$-$	$\bowtie$ — outer join } not part of Basic relational algebra.
$\times$	
ρ	

Join

→ Conditional join (Theta Join) $\rightarrow \left. \begin{matrix} R \bowtie_C S \\ R \bowtie_{\theta} S \end{matrix} \right\} \begin{matrix} \text{aka } \theta \text{ join} \\ \text{notation equivalent} \end{matrix}$

$R \bowtie_C S = \sigma_C(R \times S)$, where C is the condition

★ Dangling tuple is the set of tuples in R or S that is not present in $R \bowtie_C S$ — Does not pair with any row

Suppose relation R(A,B) has the following tuples:

A	B
2	a
4	t
2	g
4	c
9	t

→ Dangling Tuple

and relation S(B,C,D) has the following tuples:

B	C	D
c	5	6
a	7	8
t	8	3

Compute the theta-join of R and S with the condition $R.B = S.B$ AND $R.A < S.C$. Which of the following tuples is in the result? Assume each tuple has schema (A, R.B, S.B, C, D).

(9, t, t, 8, 3)

(2, g, t, 8, 3)

(4, t, t, 8, 3) ✓

A	R.B	S.B	C	D
2	a	a	7	8
4	t	t	8	3
4	c	c	5	6

$\rightarrow R \bowtie_{\theta} S$

Let us consider the following two relations R(A, B, C) and S(B, C, D).

A	B	C
4	1	1
6	3	8
8	6	3
8	6	8
7	1	5

B	C	D
1	5	6
3	7	8
5	8	9

Compute the following relational algebra: $\sigma_{(R.B = S.B) \wedge (A < D)}(R \times S)$. Which of the following tuples in the result? Assume each tuple has schema (A, R.B, R.C, S.B, S.C, D).

☐ a. (8, 6, 3, 5, 8, 9) ✗

☐ b. (4, 1, 1, 5, 8, 9) ✗

☐ c. (6, 3, 8, 3, 7, 8) ✗

☐ d. (7, 1, 5, 1, 5, 6) ✗

Equi-Join

- subset of condⁿ join
- Condition should only contain equality conditions
- All equi-joins are conditional joins

$\rightarrow$ At least two columns will be identical ☆

eg:-

A	B	C
4	5	6
7	8	9

C	D
6	8
10	12

$r \bowtie_{C=SC} (\rho_{S(SC,D)}(s))$

A	B	C	SC	D
4	5	6	6	8

Q

Consider three tables T1, T2 and T3 below. Show the results of the following operations:
(Remember, every Relational Algebra expression results into a table. For each question, you need to write down the column (attribute) names on the top, similar to tables below)

T1 (P, R, C)	T2 (A, B, C)	T3 (A, B, C)
10 a 5	10 b 6	10 b 6
15 b 8	25 c 3	30 c 3
25 a 6	10 b 5	25 c 6
15 b 5	25 c 6	
10 b 6		

f) $T1 \bowtie_{T1.C = T3.C} T3$ [equi-join is done on the specified attributes]
(i.e., get all combination of tuples that have equal values on join condition attributes, T1.C and T3.C; attribute list is the same as the Cartesian Product)

$T1 \bowtie_{T1.C = T3.C} T3 \rightarrow$

P	R	T1.C	A	B	T3.C
25	a	6	10	b	6
25	a	6	25	c	6
10	b	6	10	b	6
10	b	6	25	c	6

identical column

g) $T1 \bowtie_{T1.P = T3.A} T3 \rightarrow$

P	R	T1.C	A	B	T3.C
10	a	5	10	b	6
10	b	6	10	b	6
25	a	6	25	c	6

identical

i) $|T2 \cup T3| = 5$

ii) $|T2 \cap T3| = 2$

iii) $|T2 - T3| = 2$

iv) $|T3 - T2| = 1$

Natural Join

notation

$R \bowtie S$

eg: $R(abc)$
 $S(aba)$

then $R \bowtie S \rightarrow$ Natural join

} $R \bowtie S$

Special case: - If there is no common attribute then Natural join is cross product

$R \bowtie S = \Pi_{R \cup S}(R \times S)$
 $\theta = R \bowtie S \Rightarrow \times$ product

(a) Relation R

B	C	D
2	5	6
4	7	8
9	10	11

(b) Relation S

A	R.B	S.B	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

$A \mid B$
1 2
3 4

$R \bowtie S =$

A	B	C	D
1	2	5	6
3	4	7	8

(a) Relation U

A	B	C
1	2	3
6	7	8
9	7	8

(b) Relation V

B	C	D
2	3	4
2	3	5
7	8	10

(c) Result $U \bowtie V$

A	B	C	D
1	2	3	4
1	2	3	5
6	7	8	10
9	7	8	10

Figure 2.16: Natural join of

Division Operation

Conventional algebra analogy :-

$a/b = c$ $a, b, c \rightarrow$ Integers : $c \rightarrow$ quotient
This implies c is the largest integer such that

$$cb \leq a$$

$$\Rightarrow a/b = c \mid a, b, c \in I \Rightarrow bc \leq a \mid b(c+1) > a$$

Similarly for Relations :-

R, S, Q are relations then $\frac{R}{S} = Q$, where Q is the largest possible relation such that
 $\text{tuples}(Q \times S) = \text{tuples}(S \times Q) \subseteq \text{tuples}(R)$
 division only defined is schema $(S) \subset \text{schema}(R)$ proper subset
 Cross product in commutative in form of data

Another perspective :-

$$\frac{R(A, B)}{S(A)} = Q(B) \text{ where } A, B \text{ are set of attributes, then } \frac{R}{S} = Q \text{ implies}$$

Q has the largest set of tuples of attributes B such that every tuple in Q pairs with every tuple in S and all of the combination is present in $R(A, B)$

Consequence: - tuples in $Q(B)$ is subset of tuples in $\Pi_B(R)$
 $\Rightarrow$ While find Q , try only values/tuples in $\Pi_B(R)$

$$\frac{R \times S}{S} = R \quad \star$$

$$\frac{R}{S} = Q \Rightarrow Q \times S \subseteq R$$

eg.

find $\frac{r_1}{r_2} = q?$

a	b
1	1
1	4
2	1
2	2
2	3
2	4
3	1
3	3
3	4

(a) r_1 (dividend)

b
1
3

(b) r_2 (divisor)

$$\frac{r_1}{r_2} = q = \begin{array}{c|c} a & \\ \hline 3 & \\ \hline 2 & \end{array}$$

Figure 4.14 Examples Illustrating Division

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

pno
p2

A/B1

pno
p2
p4

B2

pno
p2
p4

A/B2

pno
p1
p2
p4

B3

pno
p1
p2
p4

A/B3

$$\frac{A}{B_1} = \begin{array}{c|c} sno & \\ \hline s1 & \\ \hline s2 & \\ \hline s3 & \\ \hline s4 & \end{array}$$

$$\frac{A}{B_2} = \begin{array}{c|c} sno & \\ \hline s1 & \\ \hline s4 & \end{array}$$

$$\frac{A}{B_3} = \begin{array}{c|c} sno & \\ \hline s1 & \end{array}$$

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

R

A
a1
a2
a3

S

B
b1
b4

T

$$R/S = q_1 = \begin{array}{c|c} B & \\ \hline b1 & \\ \hline b4 & \end{array}$$

$$R/T = q_2 = \begin{array}{c|c} A & \\ \hline a1 & \\ \hline a2 & \\ \hline a3 & \end{array}$$

Consider a database that has the relation schema CR(StudentName, CourseName). An instance of the schema CR is as given below.

StudentName	CourseName
SA	CA
SA	CB
SA	CC
SB	CB
SB	CC
SC	CA
SC	CB
SC	CC
SD	CA
SD	CB
SD	CC
SE	CD
SE	CA
SE	CB
SF	CA
SF	CB
SF	CC

The following query is made on the database.

- $T_1 \leftarrow \pi_{CourseName} (\sigma_{StudentName=SA} (CR))$
- $T_2 \leftarrow CR \div T_1$

The number of rows in T_2 is 4.

$$T_1 \rightarrow \begin{array}{c} \text{CourseName} \\ CA \\ CB \\ CC \end{array}$$

$$T_2 \rightarrow \begin{array}{c} \text{StudentName} \\ SA \\ SC \\ SD \\ SF \end{array} \quad \left. \begin{array}{c} \\ \\ \\ \end{array} \right\} 4 \text{ rows}$$

Intuitive Understanding of Division & What is the result.

$R(kid, game)$ $S(game)$

$\frac{R}{S} \rightarrow$ All the kids that play all the games in S.

→ Division operation as a derived operation $R(kid, game) \ S(game) \Rightarrow \frac{R}{S}$

→ Find all kids $\rightarrow \pi_{kid}(R)$

→ Remove kids that don't play some game in $S \rightarrow [\pi_{kid}(R) \times S] - R \rightarrow$ What remains is kid game pairs that are not in R

All possible pairs of kids & games
- for all kids & games in R

→ kid-game pairs that is not in R .

→ kids that play all game are removed

$$[\pi_{kid}(R) \times S] - R = T$$

let $(k_i, g_i) \in T$

⇒ $(k_i, g_i) \notin R \Rightarrow k_i \text{ doesn't play } g_i$

$(k_i, g_i) \in m \Rightarrow g_i \in S$

⇒ kids that don't play some game in S

$$\Rightarrow \frac{R}{S} = \pi_{kid}(R) - \pi_{kid}([\pi_{kid}(R) \times S] - R)$$

→ All who play all games in S

→ Healy's Implementation

$$\frac{R(A,B)}{S(B)} = \frac{\pi_A(R)}{\text{All A's}} - \frac{\pi_A([\pi_A(R) \times S] - R)}{\text{A's that do not pair with some value in S}}$$

→ Another Implementation

Q: $R(Kid, Game) ; S(Game)$

For Every Game g_i , we find the set of kids K_i who play that game g_i ...

$$K_{g_i} = \pi_{kid}(\sigma_{game=g_i}(R))$$

What can we say about Intersection of All K_i ?

$$\bigcap_{g_i} K_{g_i} = \frac{R}{S} = \bigcap_{\forall g_i \in S} \pi_{kid}(\sigma_{game=g_i}(R))$$

eg:-

$R(A,B)$	$S(B)$
1 a_1	a_1
1 a_2	a_2
1 a_3	
2 a_1	
2 a_2	
3 a_1	
4 a_2	

$$K_{a_1} = \{1, 2, 3\}$$

$$K_{a_2} = \{1, 2, 4\}$$

$$K_{a_1} \cap K_{a_2} = \{1, 2\}$$

$$\frac{R}{S} = \{1, 2\}$$

Maier's Implementation

$$\bigcap_{\forall b_i \in B} \pi_A(\sigma_{B=b_i}(R)) = \frac{R(A,B)}{S(B)}$$

Q: R(Kid, Game); S(Game)
 Assume both R, S are Non-Empty,
 $|R| = n$; $|S| = m$
 What is minimum & maximum
 number of tuples in R/S?

Maximum $\rightarrow \frac{n}{m} \rightarrow$ largest integer
 $\rightarrow$ In case when $\left(\frac{R}{S}\right) \times S = R$
 Minimum $\rightarrow 0 \rightarrow$ No one plays all games

What happens when $|S| = 0$? In the above question

$$\frac{R(A,B)}{S(B)} = \pi_A(R) - \frac{\text{from Hady's Implementation}}{\pi_A(R) - \pi_A(\underbrace{(\pi_A(R) \times S)}_{\text{empty}}) - R}$$

$\pi_A(R) \rightarrow$ minimum = 1
 maximum = n
 $\downarrow$
 # unique A rows in R

if $|R| = 0$ then $\frac{R}{S} = 0$ from Hady's Implementation

Outer Join $\rightarrow$ By default is nothing mentioned $\rightarrow$ Outer join calculated on Natural join $\rightarrow$ Full outer join
 $\rightarrow$ If condition is mentioned $\rightarrow$ Outer join on θ join

R/S $R \bowtie S \rightarrow$ Some dangling tuples in R, S may be there in Natural join
 Outer-join $\rightarrow$ Includes dangling tuples of R or S are included.
 $\rightarrow$ Uses null values.

$R \bowtie S \rightarrow$ left outerjoin
 $R \bowtie S \rightarrow$ right outerjoin
 $R \bowtie S \rightarrow$ full outerjoin

Alternative symbol

eg.

name	phone
A	B
C	D

R =

name	email
A	F
G	H

S =

If we do a left outer join on R and S, and we match on the first column, the result is:

name	phone	email
A	B	F
C	D	-

R/S
 Natural

name	phone	email
A	B	F

R/S
 left outer

name	phone	email
A	B	F
C	D	NULL

R/S
 right outer

name	phone	email
A	B	F
G	NULL	H

R/S
 full outer

name	phone	email
A	B	F
C	D	NULL
G	NULL	H

Cross Join = Cross product

(4 points) Given R1 and R2 compute the resulting relation for the relational algebra expression:

R1

ID	A	B
0	A0	B0
1	A1	B1
2	A2	B3

R2

ID	B	D	E
0	B0	D0	E0
2	B99	D2	E2
3	B3	D3	E3

a. $R1 \cap R2$ (intersection):

b. $R1 \times R2$ (Cartesian product):

c. $R1 \bowtie R2$ (natural join):

d. $R1 \bowtie_{R1.ID = R2.ID} R2$ (left outer join):

$R1 \cap R2 \rightarrow \text{invalid}$

$R1 \times R2 \rightarrow \text{all combi}$

$R1 \bowtie R2 \rightarrow$

ID	A	B	D	E
0	A0	B0	D0	E0

$R1 \bowtie R2$
on $R1.ID = R2.ID$

Since condition is given this is outer join on conditional join

$R1 \bowtie R2 \mid R1.ID = R2.ID$

$R1.ID$	A	$R1.B$	$R2.ID$	$R2.B$	D	E
0	A0	B0	0	B0	D0	E0
2	A2	B3	2	B99	D2	E2
1	A1	B1	⊥	⊥	⊥	⊥

GO Class

(a) Relation U

A	B	C
1	2	3
4	5	6
7	8	9

Dangling Tuples of U

(c) Result $U \bowtie V$

A	B	C	D
1	2	3	10
1	2	3	11
4	5	6	⊥
7	8	9	⊥
⊥	6	7	12

Figure 5.6: Outerjoin of relations

⊥ → Null value

(a) Relation U

A	B	C
1	2	3
4	5	6
7	8	9

(b) Relation V

B	C	D
2	3	10
2	3	11
6	7	12

Figure 5.7: Result of a theta-outerjoin

A	U.B	U.C	V.B	V.C	D
4	5	6	2	3	10
4	5	6	2	3	11
7	8	9	2	3	10
7	8	9	2	3	11
1	2	3	⊥	⊥	⊥
⊥	⊥	⊥	6	7	12

Left Dangling
Right Dangling

$U \bowtie_{U.A > V.C} V$

Exercise 5.2.1: Here are two relations:

$R(A,B): \{(0,1), (2,3), (0,1), (2,4), (3,4)\}$

$S(B,C): \{(0,1), (2,4), (2,5), (3,4), (0,2), (3,4)\}$

k) $R \bowtie_L S$; l) $R \bowtie_R S$; m) $R \bowtie S$; n) $R \bowtie_{R.B < S.B} S$.

$R \bowtie S \rightarrow$

A	B	C
2	3	4
0	1	⊥
2	4	⊥
3	4	⊥
⊥	0	1
⊥	2	4
⊥	2	5
⊥	0	2

Left Dangling
Right Dangling

$A \bowtie_{R.B < S.B} S$

A	R.B	S.B	C
0	1	2	4
0	1	2	5
0	1	3	4
2	3	⊥	⊥
2	4	⊥	⊥
3	4	⊥	⊥
⊥	⊥	0	1
⊥	⊥	0	2

Left Dangling

All rows in A and matching rows in B, where there is no match in B null values will appear

$\rightarrow A \overset{?}{\bowtie} B$

All rows in B and matching rows in A, where there is no match in A, null values will appear

$\rightarrow A \overset{?}{\bowtie} B$

All rows in both tables, where there is no match in a relation null values will appear

None of these

Only matching rows in A and B

B. Consider the following two relations:

Relation r

P B R

10 a 5

15 b 8

25 a 6

Relation s

B A C

b 10 6

c 25 3

b 10 5

show the results of the following relational algebraic operations. Give $\emptyset$ as your answer if you think the operation makes no sense. However, if you believe an operation makes sense but no tuples should be returned, you should still draw the schema of the resulting table.

j. $\Pi_{P,B}(\sigma_{R < 6}(r))$

$\frac{P \mid B}{10 \mid a}$

k. $r \times s$

l. $r \bowtie s$

m. $r \bowtie_{r.R < s.C} s$

n. $r \bowtie_{r.P = s.A \text{ AND } s.C \leq r.R} s$

o. $\Pi_{B,R}(r) \div (\Pi_R \sigma_{R < 6}(r))$

p. $r \div s$

q. $r - s$

r. $\Pi_{P,R}(r) \cap \Pi_{A,C}(s)$

$T \rightarrow \Pi_R(\sigma_{R < 6}(s)) \rightarrow \frac{R}{5}$

$\hookrightarrow \Pi_{B,R}(s) \div T$

$\frac{B \mid R}{a \mid 5}$

$\frac{B \mid R}{10 \mid 5}$

Which of the following is **true** regarding joins?

A. A *theta join* allows the use of various comparators, such as =, >, and $\geq$ in the predicate

B. An *equijoin* only allows the use of the comparator = in the predicate

C. A *natural join* is an equijoin where one occurrence of each common attribute is eliminated in the resulting relation

D. The result of a *left outer join* may have cells with a NULL value

E. All of the above

1. (30 points) [Relational algebra] Consider the relations shown below.

R(ABC)

S(DE)

T(FGH)

a 1 p

b 3

b 3 q

c 2 q

a 1

a 2 r

b 3 r

a 3

a 1 p

Evaluate the relational algebra expressions listed below and show the resulting schema (i.e., attributes) and tuples.

Example of result:

AB

a 1

c 2

b 3

The relational algebra operators used here are Π for project, σ for select, $\times$ for Cartesian product, $\bowtie$ for join, $-$ for set difference, $\cup$ for set union, and $\cap$ for set intersection. For the exercise d, assume R(ABC) and T(FGH) are union-compatible and AB and DE are union-compatible, and use the attributes of the left-hand-side relation as the attributes of the resulting relation (e.g., Result(ABC) = R(ABC) - T(FGH)). Show intermediate steps as well as the final answer. If two expressions are equivalent, it suffices to evaluate the first equivalent expression and for the second (equivalent) expression only state the reason for the equivalence and omit the evaluation.

a) $\Pi_{CH}((R \bowtie_{A=D} S) \bowtie_{E > G} T)$

b) $\Pi_{CH}((S \bowtie_{G < E} T) \bowtie_{D=A} R)$

c) $\Pi_{CH}(\sigma_{A=D \text{ and } E > G}(R \times S \times T))$

$\sigma_{A < E} T \rightarrow$

$T \rightarrow$

$\frac{D \mid E \mid F \mid G \mid H}{b \mid 3 \mid a \mid 2 \mid p}$

$\frac{D \mid E \mid F \mid G \mid H \mid A \mid B \mid C}{b \mid 3 \mid a \mid 2 \mid p \mid b \mid 3 \mid p}$

$T \bowtie_{D=A} R$

$\Pi_{CH}(T \bowtie_{D=A} R)$

$\frac{C \mid H}{p \mid p}$

For problems 1-6, give a table representing the result of the given relational algebra expression on the tables below. You should still draw the table even if it is empty. If a relational algebra expression doesn't make sense, though, write "doesn't make sense" as your answer.

R:			S:		T:		
a	b	c	a	d	e	f	g
U101	Daniel	Day-Lewis	U101	2.36	0.00	3.00	No
U102	Denzel	Washington	U103	3.52	3.01	3.09	Maybe
U103	Margot	Robbie	U104	3.09	3.10	4.00	Yes
U104	Saoirse	Ronan					

- $\pi_a(\sigma_{b \geq c}(R))$ → doesn't make sense
- $S \times T$ → $|S| \times |T| = 9$
- $R \bowtie S$ → $a \ b \ c \ d$
- $R \bowtie S$ → 3 tuples → no dangling T in S
- $\pi_{b,d,g}(R \bowtie S \bowtie_{T.e \leq S.d \leq T.f} T)$
- $\rho_{f \rightarrow d}(T) \div \pi_d(S \bowtie_{\sigma_{c=Ronan}(R)})$

d
3.09

e	g
3.01	Maybe

b	d	g
Daniel	2.36	No
Margot	3.52	Yes
Saoirse	3.09	Maybe

★ Natural join is associative

$$R \bowtie S \bowtie T = (R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

Consider the following relations and query.

$r(A, B, C)$
 $s(A, B, C)$
 $\sigma_{A > C}((r \cup s) - (r - s) - (s - r))$

Which of the following is equivalent to the above query?

- $\sigma_{A > C}(r \bowtie s)$
- $\sigma_{A > C}(r \cup s)$
- $\sigma(r \cap s)$
- $\sigma_{A > C}(r \cap s)$

$\sigma_{A > C}(R \cap S)$

$R \cap S$ → Reminds intersection as a derived operator

$$R \cap S = [R \cup S - (R - S)] - (S - R)$$

$\underbrace{\text{in } R \cap S} = \underbrace{\text{in } R} - \underbrace{\text{only in } R} - \underbrace{\text{only in } S}$

Consider the relations r,s and query:

$t1 \leftarrow \sigma_{\theta}((r \times s) \div r)$
 $t2 \leftarrow \sigma_{\theta}(s)$
 $t3 \leftarrow \sigma_{\theta}(r)$

Which of the following is correct?

- $t1 = t3$
- $t2 = t3$
- $t1 = t2$ ✓
- None of the above

Expressing Outer Join in terms of Basic Operators

$$R \bowtie S \rightarrow R \bowtie S + \text{Dangling Tuples of } R$$

$$\rightarrow R \bowtie S \cup \left[R - \pi_R \left(\pi_{R \text{ Attributes}}(R \bowtie S) \right) \right] \times \{ \text{Null, Null, Null} \}$$

$\xrightarrow{\# \text{Null}}$
 $= \text{Arity}(S) - \# \text{Common attribute}$

$$R \bowtie S \rightarrow \pi_{R, S} \left[\sigma_{\substack{R: \text{Common A} = \\ S: \text{Common A}}} (R \times S) \right]$$

$\downarrow$
 Without Repeating Common attribute

Understanding Query

We present a number of sample queries using the following schema:

Sailors(sid: integer, sname: string, rating: integer, age: real)
 Boats(bid: integer, bname: string, color: string)
 Reserves(sid: integer, bid: integer, day: date)

The key fields are underlined, and the domain of each field is listed after the field name. Thus *sid* is the key for Sailors, *bid* is the key for Boats, and all three fields together form the key for Reserves. Fields in an instance of one of these relations will be referred to by name, or positionally, using the order in which they are listed above.

In the rest of this chapter (and in Chapter 5), we illustrate queries using the instances *S3* of Sailors, *R2* of Reserves, and *B1* of Boats, shown in Figures 4.15, 4.16, and 4.17, respectively.

sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance *S3* of Sailors

sid	bid	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance *R2* of Reserves

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green

4) Find the Name of Sailors who have reserved boat 103

$$\rightarrow \pi_{\text{sname}}(S \bowtie_{\theta} R) \quad \theta \Rightarrow R.\text{bid} = 103$$

$$\rightarrow \pi_{\text{sname}}(\sigma_{R.\text{bid}=103}(S \bowtie R))$$

$$\rightarrow \pi_{\text{sname}}(S \bowtie (\sigma_{R.\text{bid}=103}(R))) \rightarrow \text{Optimised query}$$

☆ if selection acts only on one relation, then, it can be taken inside cross product or join

$$\text{ie. } \sigma_P(S \times P) = \sigma_P(S) \times P$$

if P only acts on S.

Note:-

→ Apply selection as early as possible if possible, it saves computation, memory, comparison

Q) Find the name of sailors who have reserved a red boat

→ 3 tables to be used

→ Must end with $\pi_{sname}(\dots)$

→ get boat id ie $\pi_{bid}(\sigma_{color=red}(Boats))$

→ $T \leftarrow \pi_{bid}(\sigma_{color=red}(Boats))$

→ List of people red boat $RB \leftarrow \pi_{sid}(T \bowtie Reserves)$

→ Names = $\pi_{sname}(Sailors \bowtie (Reserves \bowtie (\sigma_{color=red}(Boats))))$

More efficient

Names = $\pi_{sname}(Sailors \bowtie (\pi_{sid}(Reserves \bowtie (\sigma_{color=red}(Boats))))$

↓
Remove duplicates of sid



→ if the desired attributes can repeat in a query, then projection before further operation, can reduce the #tuples

Q) Find the colors of boats reserved by Lubber.

$\pi_{color}[Boats \bowtie \pi_{bid}(Reserves \bowtie \sigma_{name="Lubber"}(Sailors))]$

↓
→ only care about bid in this

→ projection removes duplicates of bid

→ since bid is not CK in Reserve so may repeat

Q) Find sid of sailors who have reserved at least one boat.

$\pi_{sid}(Reserves)$

Q) Names of sailor who have reserved at least one boat

$\pi_{name}(Sailors \bowtie \pi_{sid}(Reserves))$

↓
Remove repetition of sid

Find the names of sailors who have reserved red or green boat

$$\pi_{name} \left(\text{Sailors} \bowtie \pi_{sid} \left(\text{Reserves} \bowtie \sigma_{\text{color}=\text{red} \vee \text{color}=\text{green}}(\text{Boats}) \right) \right)$$

Find name of sailors who have reserved a red & green boat

$$\pi_{name} \left(\text{Sailors} \bowtie \left[\pi_{sid} \left(\text{Reserves} \bowtie \sigma_{\text{color}=\text{red}}(\text{Boats}) \right) \cap \pi_{sid} \left(\text{Reserves} \bowtie \sigma_{\text{color}=\text{green}}(\text{Boats}) \right) \right] \right)$$

$$\pi_{name} \left(\text{Sailors} \bowtie \left(\pi_{sid, color}(\text{Reserves} \bowtie \text{Boats}) \div \left(\frac{\text{Temp-color}}{\text{green}} \right) \right) \right)$$

$$\rightarrow \pi_{name} \left(\text{Sailors} \bowtie \pi_{sid} \left(\text{Reserves} \bowtie \sigma_{\text{color}=\text{red}}(\text{Boats}) \right) \cap \pi_{name} \left(\text{Sailors} \bowtie \pi_{sid} \left(\text{Reserves} \bowtie \sigma_{\text{color}=\text{red}}(\text{Boats}) \right) \right) \right)$$

This doesn't work $\rightarrow$ Since Sname need not be unique in Sailors
Sname is not a key $\rightarrow$ Two different people with different sid can have same name

★ $\rightarrow$ Set difference, intersection, union, joins $\rightarrow$ if applied to non key attribute may lead to wrong answer

Find Name of sailors $\rightarrow$ exactly one boat

$$\pi_{sid}(\text{Reserves}) - \pi_{sid}(\text{R} \bowtie_{\neq} \text{R}) \quad \rightarrow \quad \begin{matrix} R1.sid = R2.sid \\ R1.bid \neq R2.bid \end{matrix}$$

Atleast one boat Atleast 2 boats

Find Name of sailors $\rightarrow$ All boats

$$\pi_{name} \left(\text{Sailors} \bowtie \pi_{sid} \left(\pi_{sid, bid}(\text{R}) \div \pi_{bid}(\text{Boats}) \right) \right)$$

Find Name of sailors $\rightarrow$ All boats called 'Interlake'

$$\pi_{name} \left(\text{Sailors} \bowtie \left[\pi_{sid, bid}(\text{R}) \div \pi_{bid} \left(\sigma_{\text{name}=\text{"Interlake"}}(\text{Boats}) \right) \right] \right)$$

Tuple Relational Calculus

→ Declarative Query Language
→ Non procedural Query Language

4.3.1 Tuple Relational Calculus

A **tuple variable** is a variable that takes on tuples of a particular relation schema as values. That is, every value assigned to a given tuple variable has the same number and type of fields. A tuple relational calculus query has the form $\{ T \mid p(T) \}$, where T is a tuple variable and $p(T)$ denotes a *formula* that describes T ; we will shortly define formulas and queries rigorously. The result of this query is the set of all tuples t for which the formula $p(T)$ evaluates to **true** with $T = t$. The language for writing formulas $p(T)$ is thus at the heart of TRC and is essentially a simple subset of first-order logic. As a simple example, consider the following query.

- Every variable is tuple variable
- When evaluated, these variables take tuples as values
- $R(t)$, where t is a variable → denotes t is a tuple variable that belongs to relation R
 $t \in R$
- $t.A$ or $t[A]$ → denotes in tuple referred by t the value corresponding to attribute A

2. A query in the tuple relational calculus is expressed as

$$\{ t \mid P(t) \}$$

i.e. the set of tuples t for which predicate P is true.

3. We also use the notation

- $t[a]$ to indicate the value of tuple t on attribute a .
- $t \in r$ to show that tuple t is in relation r .

TRC:

$\{ t \mid P(t) \}$
↓
tuples t which satisfy $P(t)$
↓
predicate or condition on t
Uses First order logic

Two ways to write Queries in TRC
→ Both are equivalent, Any Query can be written in both ways.

→ Way 1

Logic → Result

eg. Relational algebra Query $\sigma_{B=17}(R)$

$$\text{TRC: } \{ t \mid t \in R \wedge t[B] = 17 \}$$

$$\{ t \mid R(t) \wedge t[B] = 17 \}$$

eg. Sailors with rating above 7

$$\text{TRC } \{ t \mid t \in \text{Sailors} \wedge t.\text{rating} > 7 \}$$

eg. $\sigma_{A=B \wedge D > 5}(R)$ → Selection in TRC

$$\text{TRC: } \{ t \mid t \in R \wedge t.A = t.B \wedge t.D > 5 \}$$

eg. $\pi_A(R)$ → Projection in TRC

$$\text{TRC: } \{ t.A \mid t \in R \}$$

$$\text{TRC } \{ t[A] \mid R(t) \}$$

order of information is not specified by TRC
TRC $\{ t.A, t.C \mid R(t) \}$ is equivalent to $\{ t.C, t.A \mid R(t) \}$

→ Way 2

Result → Logic

$$\text{eg. } \pi_A(R)$$

$$= \begin{array}{|c|} \hline A \\ \hline \alpha \\ \hline \beta \\ \hline \end{array}$$

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

$$\text{TRC: } \{ t \mid \exists x \in R (x.A = t.A) \}$$

→ Tuple variable is for result
→ In which relation t belongs to is not explicitly mentioned.
→ schema of t is also not explicitly mentioned

But we are only using $t.A \Rightarrow$ schema of t will be A .

eg. $\pi_{A,C}(R) \rightarrow$ Relational Algebra Query

$$TRC_{way1} \rightarrow \{t[A], t[C] \mid t \in R\}$$

$$TRC_{way2} \rightarrow \{t \mid \exists x \in R (x.A = t.A \wedge x.C = t.C)\}$$

→ schema of result is not explicitly mentioned
→ But schema referred by t can be inferred from the query

eg. $\sigma_{B=17} \rightarrow$ Relational Algebra Query

$$TRC_1 \rightarrow \{t \mid R(t) \wedge t.B = 17\}$$

$$TRC_2 \rightarrow \{t \mid \exists x \in R (t.B = x.B \wedge x.B = 17 \wedge t.A = x.A)\}$$

→ all conditions inside $\exists$ should use x

→ we should not write $t.B$ here although intuitive

→ Needed
→ otherwise it becomes $\pi_B(\sigma_{B=17}(R))$

eg → Combining projection and selection ☆

$$\text{Relation Algebra Query} = \pi_A(\sigma_{B=10}(R))$$

$$TRC_{way1} \rightarrow \{t.A \mid t \in R \wedge t.B = 10\}$$

$$TRC_{way2} \rightarrow \{t \mid \exists x \in R (x.B = 10 \wedge x.A = t.A)\}$$

→ Do not mention $x.B = t.B$
→ if we do that then B becomes part of result schema
→ Use only $x.A = t.A$ to get A in schema of result. Apply condition on x .

$$\text{eg} \rightarrow \pi_{AB}(\sigma_{C=1}(R))$$

$$TRC_1 \rightarrow \{t.A, t.B \mid t \in R \wedge t.C = 1\}$$

$$TRC_2 \rightarrow \{t \mid \exists x \in R (t.A = x.A \wedge t.B = x.B \wedge x.C = 1)\}$$

→ supply/project to result

→ filter from R

$$\text{eg} \rightarrow \pi_A(\sigma_{B=10, A=B}(R))$$

$$TRC_1 \rightarrow \{t.A \mid R(t) \wedge t.B = 10 \wedge t.A = B\}$$

$$TRC_2 \rightarrow \{t \mid \exists x \in R (\underbrace{x.A = t.A}_{\text{supply}} \wedge \underbrace{x.B = 10 \wedge x.A = B}_{\text{filter}})\}$$

eg. ☆ Union Query of Rel algebra in TRC

→ RVS $R(A,B)$
 $S(A,B)$

→ TRC₁ → $\{t \mid R(t) \vee S(t)\}$
 $\{t \mid t \in R \vee t \in S\}$

→ TRC₂ → $\{t \mid \exists x \in R (x=t) \vee \exists y \in S (y=t)\}$

→ I can also reuse the variable x
 $\{t \mid \exists x \in R (x=t) \vee \exists x \in S (x=t)\}$ → also valid

→ Can rename if confusing

→ Most Questions in gate, they reuse same variable

eg. - Intersection $R(A,B) \cap S(A,B)$ ☆

TRC₁ → $\{t \mid t \in R \wedge t \in S\}$

TRC₂ → $\{t \mid (\exists x \in R (t=x)) \wedge (\exists x \in S (t=x))\}$

eg. Set difference $R(A,B) - S(A,B)$ ☆

TRC₁ → $\{t \mid t \in R \wedge \neg (t \in S)\}$

TRC₂ → $\{t \mid \exists x \in R (t=x) \wedge \neg \exists x \in S (t=x)\}$

TRC₂ → $\{t \mid \exists x \in R (t=x) \wedge \forall x \in S (t \neq x)\}$

eg. Cross product → $R(A,B) \times S(C,D)$

TRC₁ → $\{t, v \mid t \in R, v \in S\}$

TRC₂ → $\{t \mid \exists x \in R (t[A] = x[A]) \wedge \exists x \in S (t[C] = x[C])\}$

TRC₂ → $\{t \mid \exists x \in R \exists y \in S (t[A] = x[A] \wedge t[C] = y[C])\}$

Cartesian/Cross Product

$R = (A, B)$ $S = (C, D)$

Relational Algebra: $r \times s$

Tuple Calculus: $\{t \mid \exists p \in r \exists q \in s (t[A] = p[A] \wedge t[B] = p[B] \wedge t[C] = q[C] \wedge t[D] = q[D])\}$

eg. Natural join → $R(A,B) \bowtie S(A,C)$

TRC₁ → $\{t, v \mid t \in R \wedge v \in S \wedge t[A] = v[A]\}$

TRC₂ → $\{t \mid \exists x \in R (x[A] = t[A] \wedge x[B] = t[B]) \wedge \exists y \in S (t[A] = y[A] \wedge t[C] = y[C])\}$

eg. Division → $\frac{R(A,B)}{S(B)} = Q(A)$

TRC₁ → $\{t[A] \mid t \in R \wedge (\forall x \in S \exists y \in R (x[B] = y[B] \wedge t[A] = y[A]))\}$

TRC₂ → $\{t \mid \forall x \in S \exists y \in R (x[B] = y[B] \wedge t[A] = y[A])\}$

Q) Salary(Employee, salary-amount) Find the maximum salary amount ☆

TRC₁ → $\{ t[\text{salary-amount}] \mid (t \in \text{Salary}) \wedge (\nexists x \in \text{Salary} (x[\text{salary-amount}] \leq t[\text{salary-amount}])) \}$

TRC₂ → $\{ t \mid \forall x \in \text{Salary} (x[\text{salary-amount}] \leq t[\text{salary-amount}]) \}$

TRC₂ → $\{ t \mid \forall x (x \notin \text{Salary} \vee x[\text{salary-amount}] \leq t[\text{salary-amount}]) \}$

side quest

Other way of writing variable belonging to a relation

$$P \rightarrow Q \equiv \bar{P} \vee Q \quad \left. \begin{array}{l} \text{we know from} \\ \text{propositional logic} \end{array} \right\}$$

$$P \rightarrow Q \equiv (P \wedge \bar{Q})'$$

eg. $\emptyset \rightarrow \{ t \mid \forall x \in R (x[A] \leq t[A]) \} \rightarrow \text{To find maximum value of } R(A)$

$$= \{ t \mid \forall x (P \rightarrow Q) \}$$

$$= \{ t \mid \forall x (\bar{P} \vee Q) \}$$

$$= \{ t \mid \forall x ((x \notin R) \vee (x[A] \leq t[A])) \}$$

Q) Find the name & age of Sailors with rating above 7

TRC₁ → $\{ s[\text{name}, \text{age}] \mid s \in \text{Sailors} \wedge s[\text{rating}] > 7 \}$

TRC₂ → $\{ s \mid \exists x \in \text{Sailors} \left(\begin{array}{l} x[\text{name}] = s[\text{name}] \wedge \\ x[\text{age}] = s[\text{age}] \wedge \\ x[\text{rating}] > 7 \end{array} \right) \}$ → Supply → Filter

Q: Find the sailor name, boat id, and reservation date for each reservation.

Sailors(sid: integer, sname: string, rating: integer, age: real)
Boats(bid: integer, bname: string, color: string)
Reserves(sid: integer, bid: integer, day: date)

TRC₁ → $\{ s[\text{name}], R[\text{bid}, \text{date}] \mid s \in \text{Sailors} \wedge R \in \text{Reserve} \wedge s[\text{sid}] = R[\text{sid}] \}$

TRC₂ → $\{ t \mid \exists s \in \text{Sailors} \exists R \in \text{Reserve} \left(\begin{array}{l} \text{Supply: } s[\text{sname}] = t[\text{sname}] \wedge s[\text{sid}] = R[\text{sid}] \\ \text{Selection: } R[\text{bid}] = t[\text{bid}] \wedge R[\text{date}] = t[\text{date}] \end{array} \right) \}$

Q) Find the name of sailor who reserved boat 103.

TRC₁ → $\{ t[\text{sname}] \mid t \in \text{Sailors} \wedge \exists R \in \text{Reserves} \left(\begin{array}{l} t[\text{sid}] = R[\text{sid}] \wedge \\ t[\text{bid}] = 103 \end{array} \right) \}$

TRC₂ → $\{ t \mid \exists R \in \text{Reserves} \exists s \in \text{Sailors} \left(\begin{array}{l} t[\text{sname}] = s[\text{sname}] \wedge \\ s[\text{sid}] = R[\text{sid}] \\ R[\text{bid}] = 103 \end{array} \right) \}$

Find the names of sailors who have reserved at least two boats.

$$TRC_1 \rightarrow \{ t[sname] \mid t \in \text{Sailors} \wedge \exists R_1, R_2 \in \text{Reserves} \left(\begin{array}{l} R_1[sid] = R_2[sid] \wedge \\ R_1[bid] \neq R_2[bid] \wedge \\ t[sid] = R_1[sid] \end{array} \right) \}$$

★ notation for two variable

$$TRC_2 \rightarrow \{ t \mid \exists R_1 \in \text{Reserves} \exists R_2 \in \text{Reserves} \exists s \in \text{Sailors} \left(\begin{array}{l} R_1[sid] = R_2[sid] \wedge \\ R_1[bid] \neq R_2[bid] \wedge \\ s[sid] = R_1[sid] \wedge \\ s[sname] = t[sname] \end{array} \right) \}$$

Instead of writing $\exists R_1 \in \text{Reserve}$
 $\exists R_2 \in \text{Reserve}$

★★★★

Find the name of sailors who has reserved all boats.

$$TRC_1 \rightarrow \{ t[sname] \mid \forall B \in \text{Boats} \exists R \in \text{Reserves} \left(\begin{array}{l} t[sid] = R[sid] \wedge \\ R[bid] = B[bid] \end{array} \right) \}$$

order important

$$TRC_2 \rightarrow \{ t \mid \exists s \in \text{Sailors} \left(s[sname] = t[sname] \wedge \forall B \in \text{Boats} \exists R \in \text{Reserves} \left(\begin{array}{l} s[sid] = R[sid] \wedge \\ R[bid] = B[bid] \end{array} \right) \right) \}$$

★ This ordering is important & cannot be any other way.

$$TRC_2 \rightarrow \{ t \mid \exists s \in \text{Sailors} \forall B \in \text{Boats} \exists R \in \text{Reserve} \left(\begin{array}{l} t[sname] = s[sname] \wedge \\ s[sid] = R[sid] \wedge \\ R[bid] = B[bid] \end{array} \right) \}$$

Find the sailors who have reserved all red boats.

$$TRC_1 \rightarrow \{ t[sname] \mid (t \in \text{Sailors}) \wedge \forall B \in \text{Boats} \left((B[\text{colour}] \neq \text{red}) \vee \exists R \in \text{Reserves} \left(\begin{array}{l} B[bid] = R[bid] \wedge \\ R[sid] = t[sid] \end{array} \right) \right) \}$$

$$TRC_2 \rightarrow \{ t \mid \exists s \in \text{Sailors} \forall B \in \text{Boats} \left((B[\text{colour}] \neq \text{red}) \vee \exists R \in \text{Reserves} \left(\begin{array}{l} B[bid] = R[bid] \wedge \\ R[sid] = s[sid] \wedge \\ s[sname] = t[sname] \end{array} \right) \right) \}$$

Safe Vs Unsafe TRC

- Provides data not in database $\Rightarrow$ Unsafe $\rightarrow$ eg. $\{ t \mid t \neq s \}$
 $\rightarrow$ unsafe
- All data from within database $\Rightarrow$ safe
- If a Query returns infinite #tuples $\Rightarrow$ Unsafe
- Unsafe Query can have finite #tuples also

★ $\rightarrow$ All relational algebra Queries are safe & can be expressed as relational calculus Queries.

Safe TRC $\equiv$ Relational Algebra

PYQs

The relational algebra expression equivalent to the following tuple calculus expression:

$\{t \mid t \in r \wedge (t[A] = 10 \wedge t[B] = 20)\}$ is

- A. $\sigma_{(A=10 \vee B=20)}(r)$ ✗
- B. $\sigma_{(A=10)}(r) \cup \sigma_{(B=20)}(r)$ ✗
- C. $\sigma_{(A=10)}(r) \cap \sigma_{(B=20)}(r)$ ✓
- D. $\sigma_{(A=10)}(r) - \sigma_{(B=20)}(r)$ ✗

$\rightarrow \sigma_{A=10 \wedge B=20}(R)$

3.13.6 Relational Calculus: GATE CSE 2002 | Question: 1.20

With regards to the expressive power of the formal relational query languages, which of the following statements is true?

- A. Relational algebra is more powerful than relational calculus
- B. Relational algebra has the same power as relational calculus
- C. Relational algebra has the same power as safe relational calculus ✓
- D. None of the above

Consider the relation **employee**(name, sex, supervisorName) with *name* as the key, *supervisorName* gives the name of the supervisor of the employee under consideration. What does the following Tuple Relational Calculus query produce?

$\{e.name \mid employee(e) \wedge (\forall x) [\neg employee(x) \vee x.supervisorName \neq e.name \vee x.sex = "male"]\}$

- A. Names of employees with a male supervisor.
- B. Names of employees with no immediate male subordinates.
- C. Names of employees with no immediate female subordinates. ✓
- D. Names of employees with a female supervisor.

Another way to write $\forall x \in employee$

Name of the employee who is not a supervisor for a female employee

$p \Rightarrow q$

$x.sex = female \Rightarrow x.supervisorName \neq e.name$

$\bar{p} \vee q$

$x.sex = male \vee x.supervisorName \neq e.name$

3.13.9 Relational Calculus: GATE CSE 2008 | Question: 15

Which of the following tuple relational calculus expression(s) is/are equivalent to $\forall t \in r (P(t))$?

- I. $\neg \exists t \in r (\neg P(t))$
- II. $\exists t \notin r (P(t))$
- III. $\neg \exists t \in r (\neg P(t))$ ✓
- IV. $\exists t \notin r (\neg P(t))$

- A. I only
- B. II only
- C. III only ✓
- D. III and IV only

III only



$$\neg \exists x P(x) \equiv \forall x \neg P(x)$$

$$\neg \forall x P(x) \equiv \neg \exists x \neg P(x)$$

SQL → Structured Query Language

↳ Declarative, commercial, widely used

- ① DDL → Data Definition Language → create, alter, drop, - Integrity constraints
- ② DML → Data Manipulation Language → Query, insert, delete, modify

Theoretical

Practical

Pure Relational

Real world SQL model

- | | |
|---------------------------------|--|
| → No same rows | → Same rows allowed → gives frequency info |
| → Exactly one PK | → May or maynot have any PK, FK & CK |
| → Atleast one CK & SK | |
| → No Nulls except for outerjoin | → Null values allowed |
| → Tuple is a <u>set</u> | → Tuple is <u>multiset</u> $\{1, 2, 2\} \neq \{1, 2\}$ |

SQL allows duplicate tuples for following reasons:

1. Duplicate elimination is a expensive operation.
2. User may want to see duplicate tuples in the result of query.
3. When an aggregate function is applied to tuples, in most cases user don't want to remove duplicates.

→ SQL → not case-sensitive

- Set
 - No duplicates
 - Relational model and algebra use set semantics
- Bag
 - Duplicates allowed
 - Number of duplicates is significant
 - SQL uses bag semantics by default

Database Management System

Before proceeding, we must point out an important distinction between the practical SQL model and the formal relational model discussed in Chapter 5: SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values. Hence, in general, an SQL table is not a set of tuples, because a set does not allow two identical members; rather, it is a multiset (sometimes called a bag) of tuples. Some SQL relations are constrained to be sets because a key constraint has been declared or because the DISTINCT option has been used with the SELECT statement (described later in this section). We should be aware of this distinction as we discuss the examples.

→ Queries do not change the original data

Basic SQL Query

Convention — keyword capital
— identifier small letter

Mandatory SELECT
FROM
Optional WHERE

$A_1, A_2, A_3, \dots, A_k$ } Attribute selection
 R, S, T } Cross product
Condition C

→ By default duplicate tuples are not removed

SELECT [DISTINCT] select-list

→ Keyword to specify unique tuples in result
→ Removes duplicate

eg: SELECT A₁, D
FROM R, S
WHERE $(A_1 > 10) \wedge (C \leq 5)$

$\equiv \pi_{A_1, D} (\sigma_{(A_1 > 10) \wedge (C \leq 5)} (R \times S))$ → may include duplicate

→ Note: This is just conceptual procedure
→ Actual query will be optimised

NOTE: IN SQL

PK cannot be NULL & SHOULD BE UNIQUE.

- But all tables need not have PK.
→ If it has, then above applies

eg.

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade='B'
```

Enrolled

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

Result)

name	cid
Jones	History105

Conceptually $\rightarrow \Pi_{S.name, E.cid} (\sigma_{E.grade='B' \wedge S.sid=E.sid} (S \times E))$

Actually $\rightarrow \Pi_{S.name, E.cid} (S \bowtie \sigma_{grade='B'} (E))$

* $\rightarrow$ all attributes

eg. $\star$ SELECT $\star$ FROM R } Retrieve R as it is

```
SELECT S.name, S.login
FROM Students S
```

$\star$ This syntax can be used to temporarily create a handle to a table in another name.

eg. SELECT $\star$ FROM Students S WHERE S.age=18

eg. SELECT S.name, E.cid FROM Students S, Enrolled E WHERE S.sid=E.sid AND E.grade='B'

Basic SQL Query

```
SELECT [DISTINCT] target-list
FROM relation-list
WHERE qualification
```

relation-list: A list of relation names

possibly with a **range-variable** after each name

target-list: A list of attributes of tables in **relation-list**

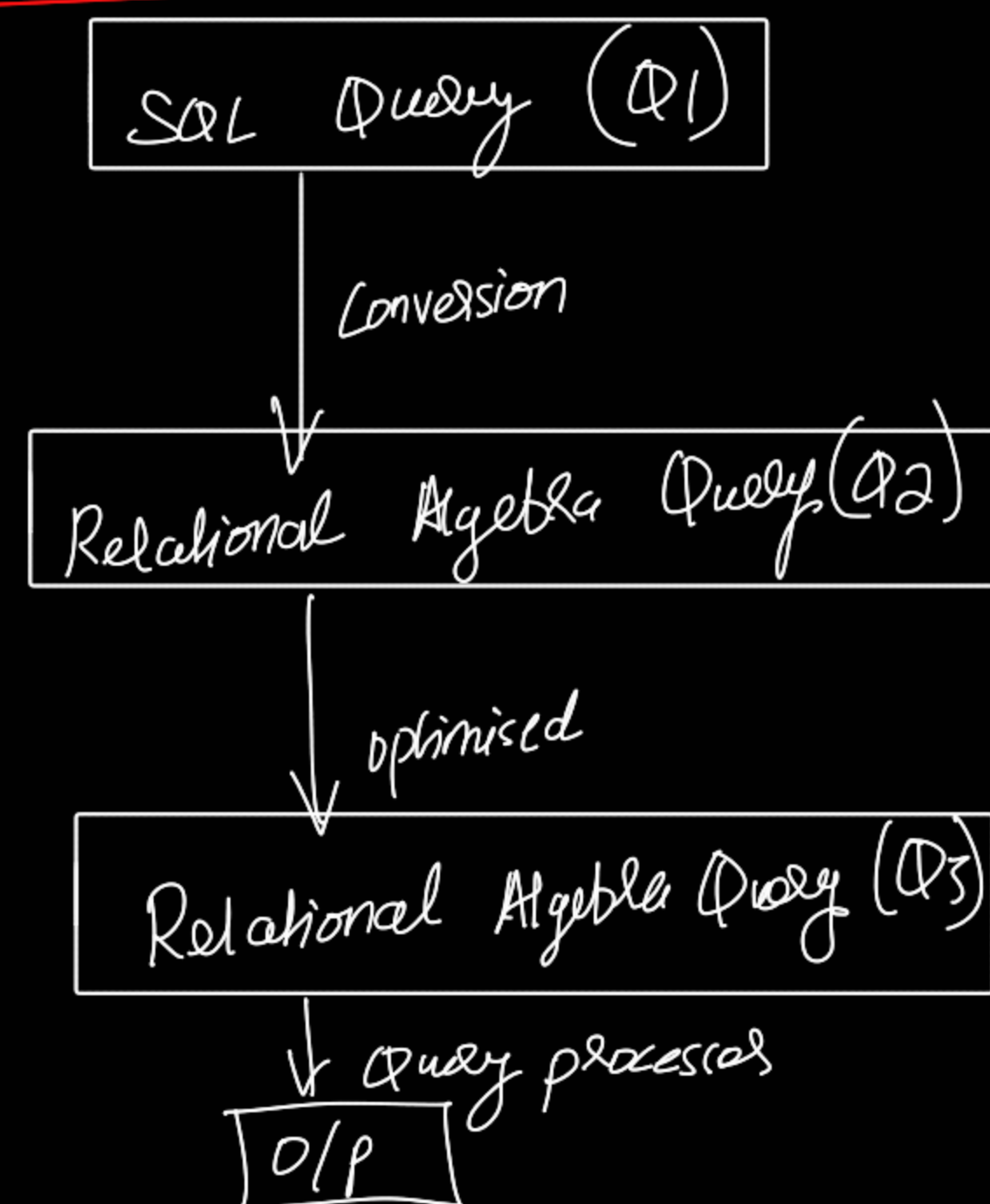
qualification: Comparisons combined using **AND**, **OR** and **NOT**.

Comparisons are Attr op const or Attr1 op Attr2, where op is one of **<**, **>**, **=**, **≤**, **≥**, **≠**, **<>**

DISTINCT: optional keyword indicating that the answer should not contain duplicates.

In SQL SELECT, the default is that duplicates are **not** eliminated! (Result is called a "multiset")

Actual Backend implementation



$\star$ In SQL $\neq$ wrong
 $\lt \gt$ correct

We will present a number of sample queries using the following table definitions:

Sailors(sid: integer, sname: string, rating: integer, age: real)
Boats(bid: integer, bname: string, color: string)
Reserves(sid: integer, bid: integer, day: date)

Q1 → Names of sailors who have reserved boat #103

SQL
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.bid = 103

Query optimisation Side quest

SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.bid = 103

Natural join

Conditional Natural join

$\Pi_{sname} [Sailors \bowtie \tau_{bid=103}(Reserves)]$

eg. Find name & age of all sailors.

SELECT S.sname, S.age
FROM Sailors S
#Tuples in o/p = 10

sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

No duplicates
SELECT DISTINCT S.sname, S.age
FROM Sailors S
#Tuples in o/p = 9

eg. Sailors with rating above 7.

SELECT *
FROM Sailors
WHERE rating > 7

RANGE VARIABLE & AS KEYWORD

(Q11) Find all sailors with a rating above 7.

SELECT S.sid, S.sname, S.rating, S.age
FROM Sailors AS S
WHERE S.rating > 7

optional

This query uses the optional keyword **AS** to introduce a range variable. Incidentally, when we want to retrieve all columns, as in this query, SQL provides a convenient shorthand: We can simply write **SELECT ***. This notation is useful for interactive querying, but it is poor style for queries that are intended to be reused and maintained.

As these two examples illustrate, the **SELECT** clause is actually used to do *projection*, whereas *selections* in the relational algebra sense are expressed using the **WHERE** clause! This mismatch between the naming of the selection and projection operators in relational algebra and the syntax of SQL is an unfortunate historical accident.

Renaming / Alias (AS keyword range variable)
Sailors AS S $\equiv$ Sailors S

Optional → All other cases
Mandatory → When same table taken more than once

eg. Query to find the supervisor of supervisor of 101

SELECT S.sup_id
FROM Emp E, Emp S
WHERE E.sup_id = S.emp_id AND E.emp_id = 101

mandatory alias

Relation
Emp(emp_id, sup_id)

101	102
102	103
103	104
104	101

☆☆☆
The table name cannot be used as an implicit range variable once a range variable is introduced for the relation.

Range variables need to be introduced explicitly only when the FROM clause contains more than one occurrence of a relation.

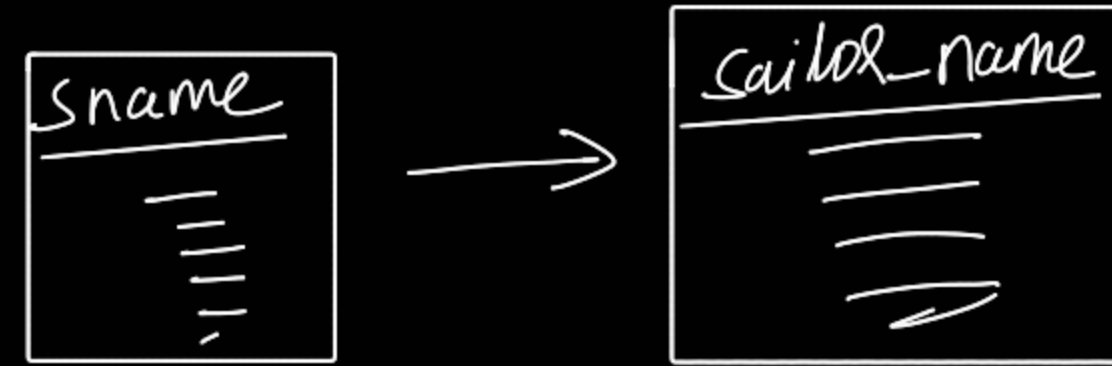
Q) Find sailors names who have reserved atleast one boat
referencing implicitly is not mandatory.

SELECT S.sname
 FROM Sailors S, Reserves R
 WHERE S.sid = R.sid

SELECT sname
 FROM Sailors S, Reserves R
 WHERE S.sid = R.sid

SELECT sname AS sailor_name
 FROM Sailors S, Reserves R
 WHERE S.sid = R.sid

→ o/p column name will change



Renaming column headers

Sometimes, we wish to produce a relation with column headers different from the attributes of the relation mentioned in the FROM clause. We may follow the name of the attribute by the keyword AS and an *alias*, which becomes the header in the result relation. Keyword AS is optional. That is, an alias can immediately follow what it stands for, without any intervening punctuation.

Summary

We now consider the syntax of a basic SQL query in more detail.

- The **from-list** in the FROM clause is a list of table names. A table name can be followed by a **range variable**; a range variable is particularly useful when the same table name appears more than once in the from-list.
- The **select-list** is a list of (expressions involving) column names of tables named in the from-list. Column names can be prefixed by a range variable.
- The **qualification** in the WHERE clause is a boolean combination (i.e., an expression using the logical connectives AND, OR, and NOT) of conditions of the form *expression op expression*, where *op* is one of the comparison operators {<, <=, =, >, >=, >}.² An *expression* is a *column name*, a *constant*, or an (arithmetic or string) expression.
- The DISTINCT keyword is optional. It indicates that the table computed as an answer to this query should not contain *duplicates*, that is, two copies of the same row. The default is that duplicates are not eliminated.

Although the preceding rules describe (informally) the syntax of a basic SQL query, they don't tell us the *meaning* of a query. The answer to a query is itself a relation—which is a *multiset* of rows in SQL!—whose contents can be understood by considering the following conceptual evaluation strategy:

1. Compute the cross-product of the tables in the **from-list**.
2. Delete those rows in the cross-product that fail the **qualification** conditions.
3. Delete all columns that do not appear in the **select-list**.
4. If DISTINCT is specified, eliminate duplicate rows.

This straightforward conceptual evaluation strategy makes explicit the rows that must be present in the answer to the query. However, it is likely to be quite inefficient.

for now, our purpose is simply to explain the meaning of a query.

Practice

Q1) Sailors who have reserved red boat

SELECT sname, S.sid
 FROM Sailors S, Reserves R, Boats B
 WHERE R.bid = B.bid AND color = "red" AND
 R.sid = S.sid

Q2) Names of sailors who reserved red.

SELECT sname
 FROM Sailors S, Reserves R, Boats B
 WHERE R.bid = B.bid AND color = "red" AND
 R.sid = S.sid

Q3) Color of Boats reserved by "Lubber"

SELECT color
 FROM Sailors S, Reserves R, Boats B
 WHERE R.bid = B.bid AND sname = "Lubber" AND
 R.sid = S.sid

Keywords: Union, Intersect, Except (set difference)

→ Exactly same as Relational algebra

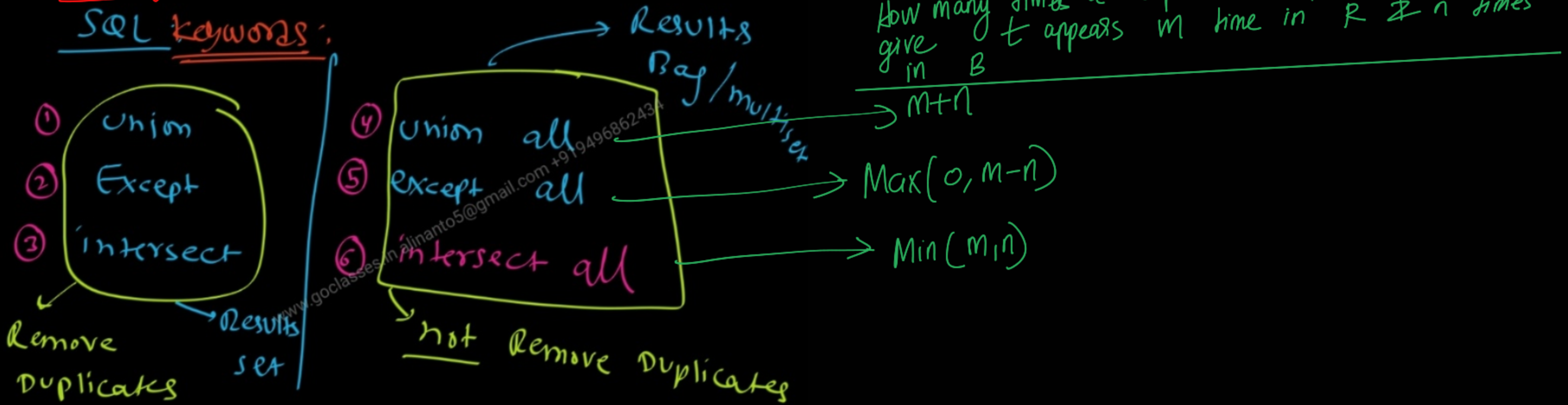
EXCEPT keyword is MINUS in Oracle DBMS

Note that UNION, INTERSECT, and EXCEPT can be used on any two tables that are union-compatible, that is, have the same number of columns and the columns, taken in order, have the same types. For example, we can write the following query:

- Result of these never contain duplicates
- Result is not multiset but set

SQL has directly incorporated some of the set operations from mathematical set theory, which are also part of relational algebra (see Chapter 8). There are set union (UNION), set difference (EXCEPT),¹¹ and set intersection (INTERSECT) operations. The relations resulting from these set operations are sets of tuples; that is, duplicate tuples are eliminated from the result. These set operations apply only to type-compatible relations, so we must make sure that the two relations on which we apply the operation have the same attributes and that the attributes appear in the same order in both relations. The next example illustrates the use of UNION.

ALL keyword → used to retain duplicates



eg. $R(A)$:

A
a1
a2
a2
a3

$S(A)$:

A
a1
a2
a4
a5

$R \text{ union } S \rightarrow \text{invalid SQL} \star$

$\left(\text{SELECT } * \text{ FROM } R \right) \text{ union } \left(\text{SELECT } * \text{ FROM } S \right)$

D/p →

A
a1
a2
a3
a4
a5

removed duplicate 'a2'

$\left(\text{SELECT } * \text{ FROM } R \right) \text{ union all } \left(\text{SELECT } * \text{ FROM } S \right)$ o/p →

A
a1
a1
a2
a2
a2
a3
a4
a5

retains duplicates

$\left(\text{SELECT } * \text{ FROM } R \right) \text{ intersect } \left(\text{SELECT } * \text{ FROM } S \right)$ o/p →

A
a1
a2

$\left(\begin{smallmatrix} \text{SELECT} & * \\ \text{FROM} & R \end{smallmatrix} \right) \text{ intersect all } \left(\begin{smallmatrix} \text{SELECT} & * \\ \text{FROM} & S \end{smallmatrix} \right) \quad \text{o/p} \rightarrow \begin{array}{c} A \\ a_1 \\ a_2 \end{array}$

retains duplicates
 → but a_2 not repeated here
 Since S only has one ' a_2 '
 → if S had 2 ' a_2 ' then
 o/p → a, a_2, a_2

$\left(\begin{smallmatrix} \text{SELECT} & * \\ \text{FROM} & R \end{smallmatrix} \right) \text{ except } \left(\begin{smallmatrix} \text{SELECT} & * \\ \text{FROM} & S \end{smallmatrix} \right) \quad \text{o/p} \rightarrow \begin{array}{c} A \\ a_3 \end{array}$

$\left(\begin{smallmatrix} \text{SELECT} & * \\ \text{FROM} & R \end{smallmatrix} \right) \text{ except all } \left(\begin{smallmatrix} \text{SELECT} & * \\ \text{FROM} & S \end{smallmatrix} \right) \quad \text{o/p} \rightarrow \begin{array}{c} A \\ a_2 \\ a_3 \end{array}$

retains duplicates
 → if S had 2 ' a_2 ' then
 o/p → a_3 only.

to note about UNION, INTERSECT, and EXCEPT follows. In contrast to the default that duplicates are not eliminated unless DISTINCT is specified in the basic query form, the default for UNION queries is that duplicates *are* eliminated! To retain duplicates, UNION ALL must be used; if so, the number of copies of a row in the result is $m + n$, where m and n are the numbers of times that the row appears in the two parts of the union. Similarly, one version of INTERSECT retains duplicates—the number of copies of a row in the result is $\min(m, n)$ —and one version of EXCEPT also retains duplicates—the number of copies of a row in the result is $m - n$, where m corresponds to the first relation.

• UNION ALL, EXCEPT ALL, INTERSECT ALL

- Bag semantics
- Think of each row as having an implicit **count** (the number of times it appears in the table)
- Bag union: **sum** up the counts from two tables
- Bag difference: **proper-subtract** the two counts
- Bag intersection: take the **minimum** of the two counts

TAKE CARE NOTE (Seen in Rel.Alg.):

Are you taking care of Key or Non-key??

Union, Intersect, Except; **When done on Non-key attribute, Might lead to wrong answer... BUT NOT Always.. So again take care!!**

Q) Sids of sailors who reserved a red or a green boat

a)

```
SELECT R.sid
FROM Reserves R, boats B
WHERE R.bid = B.bid AND (B.color = "RED" OR B.color = "GREEN")
```

b)

```
(SELECT R.sid
FROM Reserves R, boats B
WHERE R.bid = B.bid AND B.color = "RED") UNION
(SELECT R.sid
FROM Reserves R, boats B
WHERE R.bid = B.bid AND B.color = "GREEN")
```

Q) Find the name of sailors who reserved both a red and a green boat.

```
SELECT S.sname
FROM Sailors S, Reserves R1, Reserves R2, boats B1, boats B2
WHERE S.sid = R1.sid AND S.sid = R2.sid AND
R1.bid = B1.bid AND R2.bid = B2.bid AND
B1.color = "RED" AND B2.color = "GREEN"
```

Q) Find sid of sailors who have reserved red boats but not green boats.

$Q_{Red} =$

```
SELECT R.sid
FROM Reserves R, Boats B
WHERE R.sid = B.sid AND B.color = "RED"
```

$Q_{Green} =$

```
SELECT R.sid
FROM Reserves R, Boats B
WHERE R.sid = B.sid AND B.color = "GREEN"
```

$Q_{Red} \text{ EXCEPT } Q_{Green}$ ✓

Q) Find sids of sailors who have rating of 10 or have reserved boat 104

```
SELECT S.sid
FROM Sailors S, Reserves R
WHERE (S.sid = R.sid AND R.bid = 104) OR S.rating = 10
```

Q) Find all sailors (sid) who have not reserved a red boat

```
SELECT sid
FROM Reserves R, Boats B
WHERE R.bid = B.bid AND color <> "RED"
```

 } X

```
SELECT S.sid
FROM Sailors S
EXCEPT
SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid = B.bid
AND B.color = 'red'
```

```
(SELECT sid FROM sailors) EXCEPT (SELECT sid FROM RESERVES R, Boats B
WHERE R.bid = B.bid AND
B.color = "RED")
```


Q Find all pairs (distinct) of same colour books $\rightarrow$ different b.sid of same colour.

```
SELECT B1.bid B2.bid
FROM Books B1, Books B2
WHERE B1.bid <> B2.bid AND B1.colour = B2.colour
```

Q Find all pairs of sailors in which older sailor has a lower rating.

```
SELECT S1.sname, S2.sname
FROM Sailors S1, Sailors S2
WHERE S1.sid <> S2.sid AND S1.age > S2.age AND S2.rating < S1.rating
```

Redundant

Side note

SQL allows us to use the keyword **all** to specify explicitly that duplicates are not removed:

```
select all dept_name
from instructor;
```

optional (redundant)

Since duplicate retention is the default, we shall not use **all** in our examples. To ensure the elimination of duplicates in the results of our example queries, we shall use **distinct** whenever it is necessary.

The **select** clause may also contain arithmetic expressions involving the operators $+$, $-$, $*$, and $/$ operating on constants or attributes of tuples. For example, the query:

```
select ID, name, dept_name, salary * 1.1
from instructor;
```

returns a relation that is the same as the *instructor* relation, except that the attribute *salary* is multiplied by 1.1. This shows what would result if we gave a 10% raise to each instructor; note, however, that it does not result in any change to the *instructor* relation.

$\rightarrow$ Arithmetic operations in select clause

```
SELECT title, length*0.016667 AS length, 'hrs.' AS inHours
FROM Movies
WHERE studioName = 'Disney' AND year = 1990;
```

produces tuples such as

title	length	inHours
Pretty Woman	1.98334	hrs.
...	...	...

We have arranged that the third column is called **inHours**, which fits with the column header **length** in the second column. Every tuple in the answer will have the constant **hrs.** in the third column, which gives the illusion of being the units attached to the value in the second column. $\square$

$\rightarrow$ Attaching constant column to o/p

Can use arithmetic expressions in **SELECT** clause (plus other operations we'll discuss later)

Use **AS** to provide column names

```
SELECT S.age, S.age-5 AS age1, 2*S.age AS age2
FROM Sailors S
WHERE S.sname = 'dustin'
```

Can also have expressions in **WHERE** clause:

```
SELECT S1.sname AS name1, S2.sname AS name2
FROM Sailors S1, Sailors S2
WHERE 2*S1.rating = S2.rating - 1
```

$\star$ Expression in where clause

Example

$\rightarrow$ Where clause is like σ in Relational algebra
 $\rightarrow$ It applies on one tuple at a time $\rightarrow$ Independently

5.1.5 Product of Bags

The rule for the Cartesian product of bags is the expected one. Each tuple of one relation is paired with each tuple of the other, regardless of whether it is a duplicate or not. As a result, if a tuple r appears in a relation R m times, and tuple s appears n times in relation S , then in the product $R \times S$, the tuple rs will appear mn times.

Duplicate mn times in cross product

GATE exam will NOT ask you about "Errors in SQL queries", So, DO NOT Worry about these "minor issues" too much.. Focus only on Meaning of the query.

Q) SQL Query to remove duplicate from a input table

```
SELECT DISTINCT *
FROM R
```

SQL Aggregate Operators

→ Assuming there are no NULL values in table for a moment.

5 Built in aggregate functions

Average: **avg**

Minimum: **min**

Maximum: **max**

Total: **sum**

Count: **count**

→ Aggregate functions apply on whole table
→ produce only one value & tuple

example → where A is attribute name

MIN(A)	MIN(DISTINCT)	} No change
MAX(A)	MAX(DISTINCT)	
AVG(A)	AVG(DISTINCT)	} These will be evaluated after removing duplicates
SUM(A)	SUM(DISTINCT)	
COUNT(A)	COUNT(DISTINCT)	

eg.

```
SELECT SUM(A), COUNT(A), MAX(B), MIN(B), AVG(B)
FROM R;
```

R =		A	B
		1	3
		3	4
		3	2
SUM(A) = 7			
COUNT(A) = 3			
MAX(B) = 4			
MIN(B) = 2			
AVG(B) = 3 = 5			

Remark

- We can also use **COUNT(*)** which counts the number of tuples in the relation constructed from the FROM and WHERE clauses of the query.

☆ **COUNT(DISTINCT *)** Invalid

SQL does not allow the use of **distinct** with **count (*)**. It is legal to use **distinct** with **max** and **min**, even though the result does not change. We can use the keyword **all** in place of **distinct** to specify duplicate retention, but, since **all** is the default, there is no need to do so.

Sum, Avg → Numeric domain

Min, Max → Numeric, string domain

count(A) → Any domain

eg) Avg age of all sailors —

```
SELECT AVG(age)
FROM SAILORS
```

eg. Avg age of sailors with a rating of 10 —

```
SELECT AVG(S.age)
FROM SAILORS S
WHERE S.rating = 10
```

eg.

```
SELECT sid, max(age)
FROM SAILORS;
```

 — Invalid ☆ → if we use at least one aggregate function (in select clause) then, No Unaggregated attribute allowed" — Unless there is a group by clause

eg: Find name & age of oldest sailors —

```
SELECT sname, max(age)
FROM SAILOR
```

 } X → Above violation

```
SELECT sname, age
FROM SAILOR
WHERE age = max(age);
```

 } X → where clause can only apply individually & independently on each tuple

Solution
Nested query

```
SELECT sname, age
FROM SAILOR
WHERE AGE = (SELECT MAX(age) FROM SAILOR);
```

 ✓
SQL implicitly converts
1 tuple, 1 column table to
relation

Observe that we have used the result of an aggregate operation in the subquery as an argument to a comparison operation. Strictly speaking, we are comparing an age value with the result of the subquery, which is a relation. However, because of the use of the aggregate operation, the subquery is guaranteed to return a single tuple with a single field, and SQL converts such a relation to a field value for the sake of the comparison.

eg:- Count the number of sailors —

```
SELECT COUNT(*)
FROM SAILORS
```

•

```
SELECT COUNT(DISTINCT uid)
FROM Member;
```


is equivalent to:
•

```
SELECT COUNT(*)
FROM (SELECT DISTINCT uid FROM Member);
```

Null UIDs are counted only once

Distinct vs Null

Yes, the DISTINCT keyword in SQL retains NULL values, but it treats all NULL values as identical duplicates, so only one NULL value will appear in the final result set regardless of how many NULL tuples are present in the original data.

When SELECT DISTINCT is used on a single column:

- If the column contains multiple NULL entries, only a single NULL is returned as part of the unique values list.

Aggregate Operators in case of NULL Values

Assume you have a Company, GO Classes. There are 10 employees, some of them work for free (like my wife); So, Salary column is "Null" for some employees.

→ Aggregate functions ignore the NULL value in the input relation.

What should happen when we find SUM(Salary)?

R(A)	A
5	5
0	0
5	5
4	4
Null	
Null	

$$\begin{aligned} \text{Count}(\ast) &= 6 \\ \text{Count}(A) &= 4 \\ \text{Sum}(A) &= 14 \\ \text{min}(A) &= 0 \\ \text{max}(A) &= 5 \\ \text{Avg}(A) &= \frac{\text{Sum}(A)}{\text{Count}(A)} = \frac{14}{4} = 3.5 \end{aligned}$$

Note: Count(*) always counts #Tuples so it doesn't ignore NULL

First removes NULL & then computes

→ for an empty relation

R(A)	A

$$\begin{aligned} \text{Count}(A) &= 0 \\ \text{Sum}(A) & \\ \text{Avg}(A) & \\ \text{min}(A) & \\ \text{max}(A) & \end{aligned}$$

= NULL → SQL defines

Special case — Relation only has NULL

R(A)	A
NULL	

- ① Count(*) = 1
- ② Count(A) = 0
- ③ MAX(A), MIN(A), AVG(A), SUM(A) } = NULL

eg. Consider the query

SELECT sid

FROM Sailors

WHERE age = max(age)

|||

WHERE age = age

Returns all sides

Returns per tuple max = age

Null values & their effect in SQL

Why SQL allows NULL → ① Value unknown but value belongs
② Value not applicable

Null × anything = NULL

Null + " = "

Null - " = "

Null / " = "

→ Arithmetic operations with null result in NULL

→ Comparison with null is always "unknown"

→ Another truth value

3 truth values in SQL { True, False, Unknown

1. When we operate on a NULL and any value, including another NULL, using an arithmetic operator like × or +, the result is NULL.

2. When we compare a NULL value and any value, including another NULL, using a comparison operator like = or >, the result is UNKNOWN. The value UNKNOWN is another truth-value, like TRUE and FALSE; we shall discuss how to manipulate truth-value UNKNOWN shortly.

Note: - WHERE clause passes a tuple only when "True" & does not pass for "false" or "unknown"

eg:-

R(A,B)	
Null	Null
Null	Null
3	3
2	1
3	Null

SELECT *

FROM R

WHERE A-B=0

o/p

A	B
3	3

Behaviour of Logic Operators with 3 truth values.

F - 0
V - 1/2
T - 1

AND $\equiv$ output minimum(operands)
OR $\equiv$ output maximum(operands)
NOT X $\equiv$ 1-X

NOT T = F	T OR V = T	T AND V = V
NOT F = T	F OR V = V	F AND V = F
NOT V = V	V OR V = V	V AND V = V

eg.

R(A)
2
Null
3
3
Null

SELECT *

FROM R

WHERE (A ≤ 2) OR (A ≥ 2)

o/p =

R(A)
2
3
3

eg:- How to find #null values in a table R(A)

SELECT COUNT(*)

FROM R WHERE A=Null;

X invalid

Null cannot be used in query as an operand.

New Keywords — IS NULL, IS NOT NULL

SELECT COUNT(*)

FROM R

WHERE A IS NULL;

o/p = 2

SELECT COUNT(*)

FROM R

WHERE A IS NOT NULL;

o/p = 3

SQL conditions, as appear in WHERE clauses of select-from-where statements, apply to each tuple in some relation, and for each tuple, one of the three truth values, TRUE, FALSE, or UNKNOWN is produced. However, only the tuples for which the condition has the value TRUE become part of the answer; tuples with either UNKNOWN or FALSE as value are excluded from the answer. That situation leads to another surprising behavior similar to that discussed in the box on "Pitfalls Regarding Nulls," as the next example illustrates.

Movies(title, year, length, genre, studioName, producerC#)

the following query:

SELECT *

FROM Movies

WHERE length ≤ 120 OR length > 120;

Intuitively, we would expect to get a copy of the Movies relation, since each movie has a length that is either 120 or less or that is greater than 120.

However, suppose there are Movies tuples with NULL in the length component. Then both comparisons length ≤ 120 and length > 120 evaluate to UNKNOWN. The OR of two UNKNOWN's is UNKNOWN, by Fig. 6.2. Thus, for any tuple with a NULL in the length component, the WHERE clause evaluates to UNKNOWN. Such a tuple is not returned as part of the answer to the query. As a result, the true meaning of the query is "find all the Movies tuples with non-NULL lengths." □

Peculiar Questions

Given a table R(A,B,C) where C is an integer-type field that permits NULL values. Choose the statement below that best describes the result from the SQL query:

```
SELECT * FROM R WHERE C > 10 OR C <= 10
```

- ☐ All rows from R
- ☐ The empty set, due to conflicting conditions in the WHERE clause
- ☐ All rows from R where column C contains the value NULL
- ☒ All rows from R where the value in column C is non-NULL

} → if C doesn't permit NULL values
 } → if we replace OR with AND

The SQL aggregate function that gives the number of rows containing non-null values for a given column is _____.

Select one:

- ☐ a. SUM
- ☐ b. MIN
- ☐ c. MAX
- ☒ d. COUNT

Which aggregate function returns a count of all non-null values returned by a value expression?

- ☐ COUNT(*) → Counts NULL also
- ☐ MAX(value expression)
- ☐ SUM(value expression)
- ☒ COUNT(value expression)

Another issue in the presence of *null* values is the definition of when two rows in a relation instance are regarded as *duplicates*. The SQL definition is that two rows are duplicates if corresponding columns are either equal, or both contain *null*. Contrast

this definition with the fact that if we compare two *null* values using =, the result is unknown! In the context of duplicates, this comparison is implicitly treated as true, which is an anomaly.

☆ → Two tuples are considered duplicate even when they contain NULL value in some attributes
 → But using '=' inside where clause doesn't return true when asking NULL = NULL

One Exception:

R

A	B
NULL	1
NULL	1
NULL	NULL
NULL	NULL
2	NULL

SELECT DISTINCT *
FROM R;

⇒ 3 tuples in Answer

DISALLOWING NULL VALUE

5.6.5 Disallowing Null Values

We can disallow *null* values by specifying NOT NULL as part of the field definition, for example, *sname* CHAR(20) NOT NULL. In addition, the fields in a primary key are not allowed to take on *null* values. Thus, there is an implicit NOT NULL constraint for every field listed in a PRIMARY KEY constraint.

eg Retrieve the names of all employees who do not have supervisors

→
 SELECT name
 FROM emp
 WHERE sup_id IS NULL

Grouping of Tuples

Q1) Average marks of all students. Student (id, name, state, marks)
SELECT avg(marks)
FROM student

Q2) Find Avg marks of all students of Gujarat
SELECT AVG(marks)
FROM students
WHERE state = "Gujarat";

Q3) Find avg marks of all students statewise (i.e. desired o/p)

state	avg-marks
SI	M1
SO	M2

SELECT state, avg(marks)
FROM students
GROUP BY state;

- state attribute is mandatory in select clause, otherwise invalid query
- As per SQL 92 standard, group by attribute should be part of SELECT clause also
- However, all modern implementations, do not care
- Gate exam & all standard exams follow SQL 92

Q4) ★ Avg marks of students, statewise, but only for those state, where average mark > 5.

SELECT state, AVG(marks)
FROM students
GROUP BY state
WHERE AVG(marks) > 5

} = INVALID
→ where clause applies individually & independently on each tuple

SELECT state, AVG(marks)
FROM students
GROUP BY state
HAVING AVG(marks) > 5

} → filter after grouping
→ HAVING works on each group individually & independently.

Note: If Group By is used then
for EVERY group, at most
one tuple will be in output.

(at most because of HAVING clause) No having clause ⇒ Exactly one tuple per group

→ Using group by without aggregation is equivalent to simple select distinct

★

SELECT A
FROM R
GROUP BY A

≡

SELECT DISTINCT A
FROM R

eg:-

R(A,B)	
1	2
1	3
5	7
1	3
6	7
Null	Null
Null	Null
6	Null

```
SELECT A, AVG(B)
FROM R
GROUP BY A
HAVING COUNT(B) > 0
```

```
SELECT A, B, AVG(B)
FROM R
GROUP BY A
HAVING COUNT(B) > 0
```

o/p(A, AVG(B))	
1	8/3
5	7
6	7

} ⇒ INVALID

Conclusion:

③ Conceptual Evaluation:

- ① FROM
- ② WHERE
- ③ GROUP BY
- ④ HAVING
- ⑤ SELECT
- ⑥ ORDER BY

MUST Rules for GROUP BY clause:

About Null Values:

If NULLs exist in the grouping attribute, then a separate group is created for all tuples with a NULL value in the grouping attribute.

simple

SELECT clause in presence of GROUP BY clause:

① aggregated attributes
② unaggregated attributes

```
SELECT A, sum(c), count(A)
GROUP BY A, B;
```

SQL 92 Issue

SELECT clause in presence of GROUP BY clause:

① aggregated attributes
② unaggregated attributes

```
SELECT A, B, sum(c), count(A)
GROUP BY A;
```

NEVER Allowed

Note:

WHERE → Some Tuples gone
GROUP BY → Now create groups

GROUP BY and the SQL/92 standard

The SQL/92 standard for GROUP BY requires the following:

- A column used in an expression of the SELECT clause must be in the GROUP BY clause. Otherwise, the expression using that column is an aggregate function.
- A GROUP BY expression can only contain column names from the select list, but not those used only as arguments for vector aggregates.

The results of a standard GROUP BY with vector aggregate functions produce one row with one value per group.

NOTE: Group By with Multiple Relations in FROM Clause:

R(A,B); S(B,C)

```
SELECT A,C, COUNT(B)
FROM R,S
WHERE A <> S.B
GROUP BY A,C;
```

R		S	
A	B	B	C
1	2	1	2
1	2	1	4
1	3	4	3
Null	Null	5	Null
Null	5	2	1
7	2	2	2

→ # Tuples in o/p?

① where is evaluated

R x S	A	RB	SB	C
1	1	2	4	3
2	1	2	5	Null
3	1	2	2	1
4	1	2	2	2
5	1	2	4	3
6	1	2	5	Null
7	1	2	2	1
8	1	2	2	2
9	1	3	4	3
10	1	3	5	Null
11	1	3	2	1
12	1	3	2	2
13	7	2	1	2
14	7	2	4	3
15	7	2	5	Null
16	7	2	2	1
17	7	2	2	2

tuples = 9 in o/p

It is also possible to use a GROUP BY clause in a query about several relations. Such a query is interpreted by the following sequence of steps:

- Evaluate the relation R expressed by the FROM and WHERE clauses. That is, relation R is the Cartesian product of the relations mentioned in the FROM clause, to which the selection of the WHERE clause is applied.
- Group the tuples of R according to the attributes in the GROUP BY clause.
- Produce as a result the attributes and aggregations of the SELECT clause, as if the query were about a stored relation R .

eg-
$$\begin{array}{c|c} R(A, B) \\ \hline \text{NULL} & \text{NULL} \end{array}$$

$Q \rightarrow \text{SELECT } A, \text{COUNT}(B)$
 $\text{FROM } R$
 $\text{GROUP BY } A$

$$\left. \begin{array}{c} \text{o/p } (A, \text{COUNT}(B)) \\ \hline \text{NULL} \quad | \quad 0 \end{array} \right\}$$

When tuples have nulls, there are a few rules we must remember:

- The value NULL is ignored in any aggregation. It does not contribute to a sum, average, or count of an attribute, nor can it be the minimum or maximum in its column. For example, COUNT(*) is always a count of the number of tuples in a relation, but COUNT(A) is the number of tuples with non-NULL values for attribute A.
- On the other hand, NULL is treated as an ordinary value when forming groups. That is, we can have a group in which one or more of the grouping attributes are assigned the value NULL.
- When we perform any aggregation except count over an empty bag of values, the result is NULL. The count of an empty bag is 0.

Database Management System GO Classes

GROUP BY A, B
HAVING A > 0

Valid

Aggregates attributes

Unaggregated attributes must be subset of Group by clause

Any attribute

Here, if we replace A > 0 with C > 0, then Query is INVALID

eg. Find the age of youngest sailors for each rating level — $\text{SELECT rating, min(age)}$
 FROM sailors
 GROUP BY rating

eg. Find the age of youngest sailors for each rating level above 7 — $\text{SELECT rating, min(age)}$
 FROM sailors
 GROUP BY rating
 $\text{HAVING rating} > 7$

Equivalent — $\text{SELECT rating, min(age)}$
 FROM sailors
 $\text{WHERE rating} > 7$
 GROUP BY rating

both give same result

★ → crash course question touching everything

Q: Suppose that we want to count the total number of employees whose salaries exceed \$40,000, in each department, but only for departments where more than five employees work.

Well framed.

→ $\text{SELECT dep, count(salary)}$
 FROM employees
 $\text{WHERE salary} > 40000$
 GROUP BY dep
 $\text{HAVING count(*)} > 5$

ORDER BY CLAUSE

new keywords — ORDER BY
— ASC (redundant)
— DESC

6.1.8 Ordering the Output

We may ask that the tuples produced by a query be presented in sorted order. The order may be based on the value of any attribute, with ties broken by the value of a second attribute, remaining ties broken by a third, and so on, as in the τ operation of Section 5.2.6. To get output in sorted order, we may add to the select-from-where statement a clause:

ORDER BY <list of attributes>

The order is by default ascending, but we can get the output highest-first by appending the keyword DESC (for “descending”) to an attribute. Similarly, we can specify ascending order with the keyword ASC, but that word is unnecessary.

eg. → Order professors with descending order of salary, if two or more instructors has same salary, order them by ascending order of name.

Q → `SELECT *`
`FROM instructors`

`ORDER BY salary DESC, name ASC;`
→ redundant → can be omitted.

☆ → Every column mentioned in the order by clause must also appear in select clause

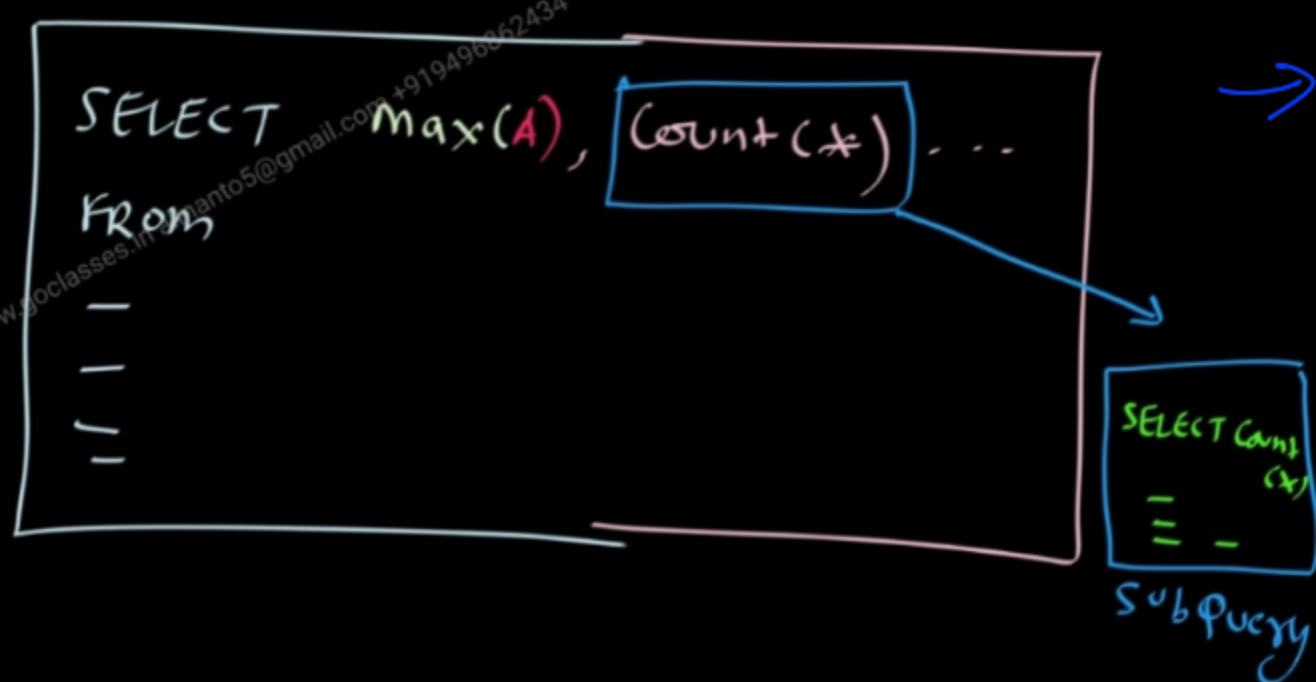
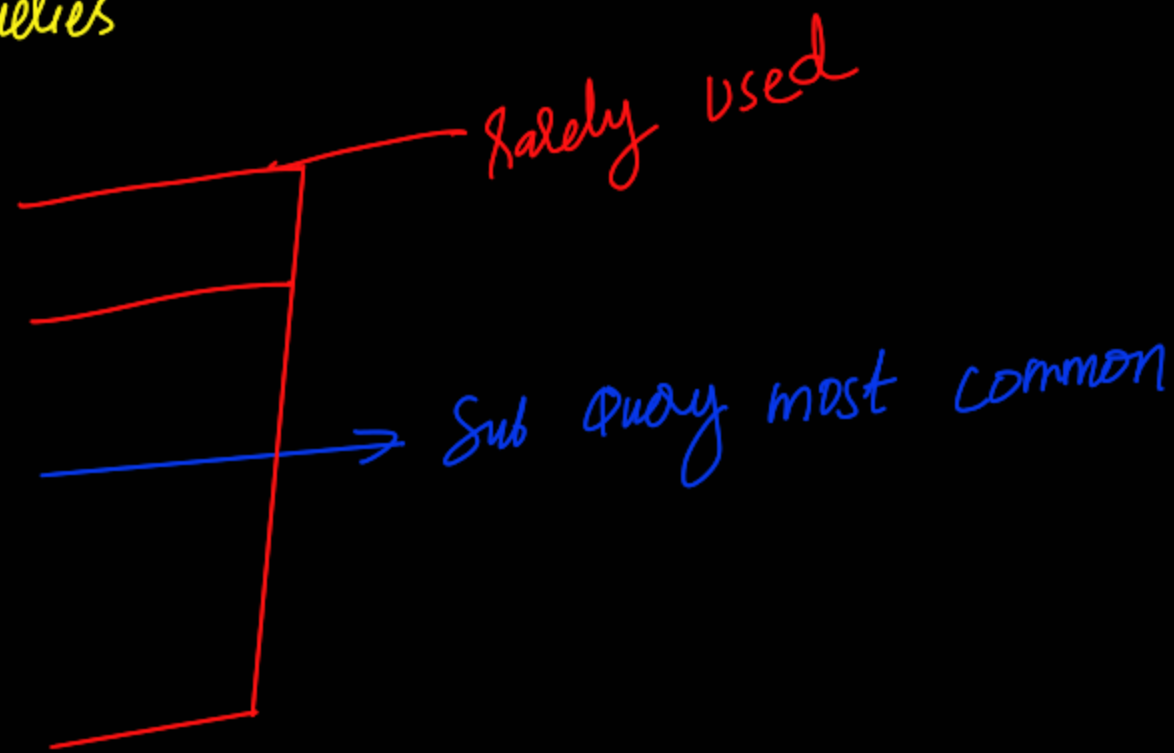
Database Management System

Conceptual Order of Evaluation of a Select Statement:

1. First the product of all tables in the **FROM** clause is formed.
2. The **WHERE** clause is then evaluated to eliminate rows that do not satisfy the search_condition.
3. Next, the rows are grouped using the columns in the **GROUP BY** clause.
4. Then, Groups that do not satisfy the search_condition in the **HAVING** clause are eliminated.
5. Next, the expressions in the **SELECT** clause target list are evaluated.
6. If the **DISTINCT** keyword is present in the select clause, duplicate rows are now eliminated.
7. Finally, the resulting rows are sorted according to the columns specified in the **ORDER BY** clause.

Nested Queries → Sub Queries

`SELECT`
`FROM`
`WHERE`
`GROUP BY`
`HAVING`
`ORDER BY`



→ when sub query in select clause → Subquery result must be a simple tuple & simple column

Two types of Sub Queries

- ① Simple Sub Query — without correlation with outer query — Evaluated once for entire outer query
- ② Correlated Sub Query — using attributes of outer query — Recomputed for every Row of outer Table.

Set comparison keywords

IN, NOT IN

set membership

$S \rightarrow \text{tuple}$
 $R \rightarrow \text{relation} \rightarrow \text{result of sub query}$
 $\rightarrow \text{Cannot be relation directly from DB}$
 $\rightarrow \text{if required use (SELECT * FROM R)}$
 $S \text{ IN } R \rightarrow \text{true if tuple S is in relation R}$
 $S \text{ NOT IN } R \rightarrow \text{True if tuple S is not in relation R.}$
 Relation R need to be compatible with tuple S.
 (same schema)

(Unary) EXISTS, NOT EXISTS

emptiness of sub query

EXISTS R $\rightarrow$ true if R only if R is non empty
 NOT EXISTS R $\rightarrow$ true if R only if R is empty.



$\rightarrow$ EXISTS treats NULL tuple as non empty. $\star$
 $\rightarrow$ Therefore EXISTS (SELECT AVG(marks) FROM R) $\frac{R(\text{marks})}{\text{NULL}}$
 returns true

(Unary) UNIQUE, NOT UNIQUE

uniqueness of sub query
 $\rightarrow$ NO duplicate Row
 $\rightarrow$ empty Relation

UNIQUE R $\rightarrow$ False if duplicate tuples
 $\rightarrow$ True if empty OR
 $\rightarrow$ NO duplicate tuples.

ANY, SOME, ALL

5.4.3 Set-Comparison Operators

We have already seen the set-comparison operators EXISTS, IN, and UNIQUE, along with their negated versions. SQL also supports op ANY and op ALL, where op is one of the arithmetic comparison operators $\{<, <=, =, <>, >=, >\}$. (SOME is also available, but it is just a synonym for ANY.)

3. $s > \text{ALL } R$ is true if and only if s is greater than every value in unary relation R . Similarly, the $>$ operator could be replaced by any of the other five comparison operators, with the analogous meaning: s stands in the stated relationship to every tuple in R . For instance, $s <> \text{ALL } R$ is the same as $s \text{ NOT IN } R$.

4. $s > \text{ANY } R$ is true if and only if s is greater than at least one value in unary relation R . Similarly, any of the other five comparisons could be used in place of $>$, with the meaning that s stands in the stated relationship to at least one tuple of R . For instance, $s = \text{ANY } R$ is the same as $s \text{ IN } R$.

The EXISTS, ALL, and ANY operators can be negated by putting NOT in front of the entire expression, just like any other boolean-valued expression. Thus, NOT EXISTS R is true if and only if R is empty. NOT $s >= \text{ALL } R$ is true if and only if s is not the maximum value in R , and NOT $s > \text{ANY } R$ is true if and only if s is the minimum value in R . We shall see several examples of the use of these operators shortly.

SOME - SYNONYMOUS with ANY

ALL $\rightarrow \forall$ Quantified
 ANY $\rightarrow \exists$ Quantified

$S \# \text{ALL}(R)$

if R is empty

$\forall x \text{ ALL} \rightarrow \text{True}$
 $\exists x \text{ ANY} \rightarrow \text{False}$

$\# \rightarrow (< >, =, \geq, \leq, >, <)$

5 Important Rules of Set Comparison

① $S \text{ IN } R \equiv S = \text{ANY } R$

② $S \text{ NOT IN } R \equiv S <> \text{ALL } R$

③ $\text{EXIST}(\text{SELECT COUNT}(*))$ always returns true

④ $\text{EXIST}(\text{SELECT aggregate}(x))$ always return true

⑤ $\begin{cases} \text{UNIQUE}(R) \rightarrow \text{True} \\ S \# \text{ALL}(R) \\ S \# \text{ANY}(R) \rightarrow \text{False} \end{cases} \left. \begin{array}{l} R \rightarrow \text{empty} \\ \text{relation} \end{array} \right\}$

eg- Name of sailor who reserved some boat

```
SELECT S.sname
FROM SAILORS S, RESERVES R
WHERE S.sid = R.sid
```

```
SELECT S.sname
FROM SAILORS
WHERE sid IN
```

(SELECT sid FROM RESERVES)

→ Calculates once & for all
→ does not depend on outer
Query
→ Simple Subquery.

```
SELECT S.sname
FROM SAILORS
WHERE sid IN (SELECT * FROM RESERVES)
```

not compatible (expects single column value)

X
INVALID

```
SELECT S.sname
FROM SAILORS
WHERE sid IN RESERVES
```

X
INVALID

Even if we ignore compatibility issue
still invalid → Reserves cannot be
used directly → has to be subquery

eg: Sid of sailors who have not reserved any boat

```
SELECT S.sname
FROM SAILORS
WHERE sid NOT IN (SELECT sid FROM RESERVES)
```

→ Simple Subquery
→ Calculated once & for all

eg: Names of sailors who reserved boat 103.

```
SELECT S.sname
FROM SAILORS
WHERE sid IN (SELECT sid
FROM RESERVES
WHERE bid = 103)
```

Simple subquery.

→ We can have as many levels of nesting as we want

eg: Name of sailors → reserved red boat

```
SELECT S.sname
FROM SAILORS
WHERE sid IN (SELECT sid
FROM RESERVES
WHERE bid IN (SELECT bid
FROM BOATS
WHERE color = "red"))
```

eg: Sailors who have not reserved red boat

```
SELECT S.sname
FROM SAILORS
WHERE sid NOT IN (SELECT sid
FROM RESERVES
WHERE bid IN (SELECT bid
FROM BOATS
WHERE color = "red"))
```


Q: What is the Output??

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN ( SELECT R.sid
                  FROM Reserves R
                  WHERE R.bid IN ( SELECT B.bid
                                   FROM Boats B
                                   WHERE B.color = 'red' )
                  NOT )
```

- A. Sailors who booked NO red boat.
- B. Sailors who not booked some red boat.
- C. Sailors who booked some Non-red boat. ✓
- D. Sailors who booked All Non-red boat.

www.goclasses.in

Q: What is the Output??

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN ( SELECT R.sid
                  FROM Reserves R
                  WHERE R.bid IN ( SELECT B.bid
                                   FROM Boats B
                                   WHERE B.color = 'red' )
                  NOT )
```

- A. Sailors who booked NO red boat.
- B. Sailors who Not booked some red boat.
- C. Sailors who booked NO Non-red boat. ✓
- D. Sailors who booked All Non-red boat.

www.goclasses.in

Correlated Sub Query.

eg)

```
SELECT *
FROM R
WHERE A > (SELECT COUNT(*)
           FROM S
           WHERE R.B = S.B)
```

R =

A	B
1	2
3	4
5	6

S =

B	C	D
2	4	6
4	6	8
4	7	9

O/P →

A	B
3	4
5	6

Correlated Sub Query
 → Recomputed for each tuple in Main Query
 → Used attribute from outer Query inside subquery.

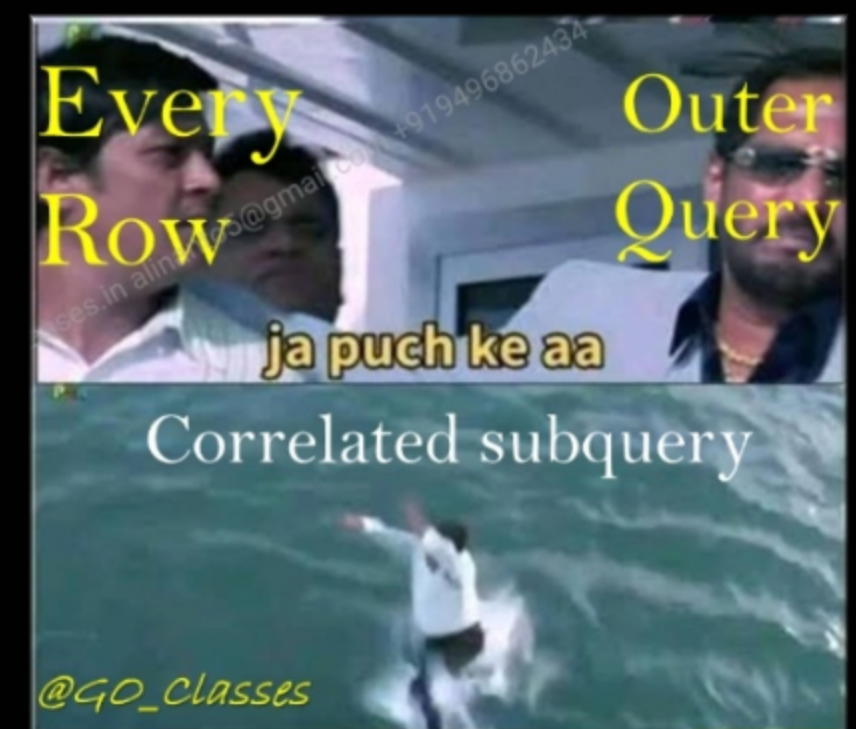
eg) Example:

```
SELECT *
FROM R
WHERE EXISTS ( SELECT *
               FROM S
               WHERE B < C )
```

Used attribute from outer Query.
 Correlated Sub Query.

O/P →

A	B
1	2
3	4
5	6



Example:

```
SELECT *
FROM R
WHERE EXISTS ( SELECT count(*)
               FROM S
               WHERE B < C )
```

Number of Tuples in o/p ??

→ Count * always returns 1 tuple
 → EXISTS (SELECT Count *) always returns true.

≡ SELECT * FROM R } All tuples = 4

Example:

```
SELECT *
FROM R
WHERE ( SELECT *
        FROM S
        WHERE B < C ) >= 2 ;
```

INVALID
 If * replaced with count(*) Valid

O/P →

A	B
1	2
3	4

eg) Names of sailors who have reserved boat 103. Write Correlated Query.

```
SELECT S.sname
FROM Sailors S
WHERE EXISTS (SELECT *
FROM Reserves R
WHERE S.sid = R.sid AND
R.bid = 103)
```



if we replace * with count (*) then our result is all sailor names
→ remember, EXIST (SELECT count (*) always returns true)

eg. Name of sailors → Not reserved boat 103

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXIST (SELECT *
FROM Reserves R
WHERE S.sid = R.sid AND
R.bid = 103)
```

eg. Name sailors → Atleast one boat → write correlated Query

```
SELECT S.sname
FROM Sailors S
WHERE EXIST (SELECT *
FROM Reserves R
WHERE S.sid = R.sid AND)
```

eg. Name sailors → Almost one boat → write correlated Query

```
SELECT S.sname
FROM Sailors S
WHERE UNIQUE (SELECT sid
FROM Reserves R
WHERE S.sid = R.sid AND)
```

eg. Sailors who rating is better than some/every sailor called Horatio

```
SELECT S.sid
FROM Sailors S
WHERE S.rating > ANY (SELECT S2.rating
FROM Sailors S2
WHERE S2.sname = "Horatio")
```

ANY for some sailor
ALL for every sailor

eg. Find sailor with highest rating

```
SELECT S.sid
FROM Sailors S
WHERE S.rating ≥ ALL (SELECT S2.rating
FROM Sailors S2)
```


Division operation in SQL

eg) Name of sailors who have reserved all boats

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS
```

(Boats which are not reserved by S.sid)

```
SELECT B.bid
FROM Boats B
WHERE NOT EXISTS
```

(Reservation tuple such that B.bid = R.bid AND S.sid = R.sid)

↓

```
SELECT *
FROM RESERVES R
WHERE B.bid = R.bid AND S.sid = R.sid
```

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS
```

```
( SELECT B.bid
  FROM Boats B
  WHERE NOT EXISTS ( SELECT *
                    FROM RESERVES R
                    WHERE B.bid = R.bid AND S.sid = R.sid ) )
```

In general $\frac{R(A,B)}{S(B)} = Q(A)$

```
SELECT R1.A
FROM R AS R1
WHERE NOT EXISTS
```

```
( SELECT S.B
  FROM S
  WHERE NOT EXISTS
```

list of tuples in S that do not pair with current tuple of consideration in R1

```
( SELECT *
  FROM R AS R2
  WHERE R2.A = R1.A AND R2.B = S.B )
```

→ List of tuples in R2 where current tuple in R1 pair with current tuple in S
→ if this is empty then current tuple in R1 do not pair with current tuple of S.

$\frac{R(A,B)}{S(B)} = Q(A)$

Hayle's Implementation

$\pi_A(R) - \pi_A[\pi_A(R) \times S - R]$

→ All people

→ All people paired with all boats

people who have not reserved all boats

Equivalent division as above

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS
```

(Boats which are not reserved by S.sid)

```
SELECT B.bid
FROM Boats B
```

All boats

EXCEPT

```
SELECT R.bid
FROM Reserves R
WHERE S.sid = R.sid
```

Boats reserved by S.sid

≡

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS
```

{ (SELECT B.bid FROM Boats B)

EXCEPT (SELECT R.bid FROM Reserves R WHERE S.sid = R.sid) }

In general $\frac{R(A) \div S(B)}{S(B)} = Q(A)$

```
SELECT R1.A
FROM R AS R1
WHERE NOT EXISTS
```

All tuples in S

{ (SELECT S.B FROM S)

EXCEPT

{ (SELECT R2.B FROM R AS R2 WHERE R2.A = R1.A)

All tuples in S that pair with current tuple in R1.A

→ if this is empty that means the current tuple in R1.A pairs with all tuples in S ⇒ R1.A is part of Q(A)

Clicker nested question

Determine the result of:

```
SELECT Team, Day
FROM Scores S1
WHERE Runs <= ALL
(SELECT Runs
FROM Scores S2
WHERE S1.Day = S2.Day)
```

Which of the following is in the result:

- A. (Carp, Sun)
- B. (Bay Stars, Sun)
- C. (Swallows, Mon)
- D. All of the above ✓
- E. None of the above

Scores:				
Team	Day	Opponent	Runs	
Dragons	Sun	Swallows	4	
Tigers	Sun	Bay Stars	9	
Carp	Sun	Giants	2	
Swallows	Sun	Dragons	7	
Bay Stars	Sun	Tigers	2	
Giants	Sun	Carp	4	
Dragons	Mon	Carp	6	
Tigers	Mon	Bay Stars	5	
Carp	Mon	Dragons	3	
Swallows	Mon	Giants	0	
Bay Stars	Mon	Tigers	7	
Giants	Mon	Swallows	5	

Team	Day
Carp	Sun
Bay Stars	Sun
Swallows	Mon

Consider the following database schema:
Product (pid, name, brand, price, color), where pid is the primary key.
Consider the following SQL queries:

Query P:

```
SELECT brand
FROM Product
WHERE color = 'green'
GROUP BY brand
HAVING COUNT(*) < 3
```

Query Q:

```
SELECT DISTINCT brand
FROM Product p
WHERE (SELECT COUNT(*)
FROM Product q
WHERE q.brand = p.brand
AND q.color = 'green') < 3
```

→ Brands without green color are eliminated at this stage & don't go forward to group by having clause

→ Brands without green color are also in this query.

Which of the following is correct?

- A. Query P, Q are equivalent.
- B. Query P, Q are not equivalent But if we ignore the duplicates in the result of P, Q then P, Q are equivalent.
- C. Query P, Q are not equivalent even if we ignore the duplicates in the result of P, Q. ✓
- D. Result of query P is always a superset of the result of query Q.

