SQL

Complete Summary

Part – 3

Instructor:

Deepak Poonia

MTech, IISc Bangalore

GATE CSE AIR 53; AIR 67;
AIR 107; AIR 206; AIR 256

DBMS Complete Course:

https://www.goclasses.in/courses/Database-Management-Systems

NOTE :

Complete Discrete Mathematics & Complete Engineering

Mathematics Courses, by GO Classes, are **FREE** for ALL learners.

Visit here to watch : https://www.goclasses.in/s/store/

SignUp/Login on Goclasses website for free and start learning.

GO CLASSES

FREE

COMPLETE COURSE

Engineering Mathematics

GATE CS IT

Freely Available Features

FREE

- All Video Lectures
- Quizzes
- Homeworks, Practice Sets
- Annotated Notes
- Toppers' Hand Written Notes
- GATE PYQs Video Solutions

SACHIN MITTAL SIR

www.goclasses.in

/ GoClasses

GO CLASSES

FREE

COMPLETE COURSE

Discrete Mathematics

GATE CS IT

Freely Available Features

FREE

- All Video Lectures
- Quizzes, Homeworks
- Annotated Notes
- Toppers' Hand Written Notes
- Summary Lectures
- GATE PYQs Video Solutions

DEEPAK POONIA SIR

www.goclasses.in

/ GoClasses

GO CLASSES

FREE

COMPLETE COURSE

Linear Algebra

GATE DA

Freely Available Features

FREE

- All Video Lectures
- Quizzes
- Homeworks, Practice Sets
- Annotated Notes
- Toppers' Hand Written Notes
- GATE PYQs Video Solutions

SACHIN MITTAL SIR

www.goclasses.in

/ GoClasses

We are on Telegram. **Contact us** for any help.

Link in the Description!!

Join GO Classes **Doubt Discussion** Telegram Group :

Username:

@GATECSE_GOCLASSES

We are on Telegram. **Contact us** for any help.

Join GO Classes Telegram Channel, Username: **@GOCLASSES_CSE**

Join GO Classes **Doubt Discussion** Telegram Group :

Username: **@GATECSE_Goclasses**

(Any doubt related to Goclasses Courses can also be asked here.)

Join GATEOverflow Doubt Discussion Telegram Group :

Username: **@GateOverflow_CSE**

GO CLASSES


GO CLASSES
GO CLASSES
GO CLASSES
GO CLASSES
GO CLASSES
GO CLASSES

SQL

Complete Summary

Part – 3

Recap:

SQL

Group By Clause

GROUP BY

- SELECT ... FROM ... WHERE ...
  GROUP BY *list_of_columns*;

- Example: find the average GPA for each age group
  - SELECT age, AVG(GPA)
    FROM Student
    GROUP BY age;

SQL Query Template with GROUP BY:

```
SELECT b1, ..., bk, AGG1(e1) AS x1, ..., AGGm(em) AS xm
FROM R
GROUP BY b1, ..., bk
```

SQL: Structured Query Language
Grouping Queries

SQL SELECT statement has two more clauses to support grouping operations: GROUP BY and HAVING clauses.

GROUP BY Clause

The syntax of a GROUP BY clause is:

GROUP BY <AttributeName>, ..., <AttributeName>

The GROUP BY clause is added to the SELECT statement after the WHERE clause (or, if there is no WHERE clause, after the FROM clause).

GROUP BY clause causes the DBMS to separate the Cartesian Product of all tables referenced in the FROM clause into groups of tuples.

Each group of tuples *must agree on all values* of attributes listed in the GROUP BY clause.

Describe the output of the following SQL Query.

Consider, for example the relational table

`Student(firstName, lastName, gpa, class, grade, school)`

The following query:

```
SELECT school, COUNT(*)
FROM Student
GROUP BY school;
```

Grouping operation is used to **allow for computation and reporting of aggregate operations over groups** in the SELECT statement.

Consider, for example the relational table

```
Student(firstName, lastName, gpa, class, grade, school)
```

The following query:

```
SELECT school, COUNT(*)
FROM Student
GROUP BY school;
```

Valid Query

Will output the number of students enrolled in each school.

Next Sub-Topic:

SQL

Having Clause

Having Clause:

Used to filter groups based on the group properties.

AFTER Groups are formed, we can eliminate some groups using HAVING clause.

Having Clause applies on one Group at a time.

Output??

```
SELECT A, Min(B)
FROM R
GROUP BY  A;
HAVING Count(*) > 1
```

R

A	B
1	'a'
2	'b'
3	'a'
1	'b'
2	'a'
1	'c'

Output??

A	min(B)
1	'a'
2	'a'

```
SELECT A, Min(B)
FROM R (1)
GROUP BY A; (2)
HAVING Count(*) > 1
(3)
```

Count(*) = 3 ← G_1

Count(*) = 2 ← G_2

G_3 → Count(*) = 1

R	
A	B
1	'a'
2	'b'
3	'a'
1	'b'
2	'a'
1	'c'

Group G_1 passes
Group G_2 passes
Group G_3 fails

A	B	C	D
a1	b1	1	7
a1	b1	2	8
a2	b1	3	9
a3	b2	4	10
a2	b1	5	11
a1	b1	6	12

```
SELECT A, B, SUM(C), MAX(D)
FROM  R
GROUP BY A, B;
```

HAVING B = 'b1' AND Sum(C) > 8

R

A	B	C	D
a1	b1	1	7
a1	b1	2	8
a2	b1	3	9
a3	b2	4	10
a2	b1	5	11
a1	b1	6	12

G_1 → B = 'b1' ; sum(C) = 9

G_2

G_3

B = 'b1' ; sum(C) = 8

```
(4) SELECT A, B, SUM(C), MAX(D)
(1) FROM  R
(2) GROUP BY A, B;
(3) HAVING B = 'b1' AND Sum(C) > 8
```

o/p : A B sum(C) max(D)

a1	b1	9	12

check for every Group independently

only G_1 passes

Note: when we have Group by & Having Clause ;

Number of Tuples in o/p = Number of Groups that pass

Note: 'WHERE' clause is applied before creating Groups.

Note: when we have Group by & No Having Clause ;

Number of Tuples in o/p = Number of Groups

Note: 'WHERE' clause is applied before creating Groups.

WHERE Vs HAVING:

for Tuples	for Groups
(1) Filters Tuples	(1) Filter Groups
(2) Applies on Tuples individually & independently	(2) Applies on Groups individually & independently
(3) Whole tuple selected or Rejected	(3) Whole Group Selected or Rejected

HAVING

- Used to filter groups based on the group properties (e.g., aggregate values, GROUP BY column values)
- SELECT … FROM … WHERE … GROUP BY … HAVING *condition*;
  - Compute FROM ($\times$)
  - Compute WHERE (σ)
  - Compute GROUP BY: group rows according to the values of GROUP BY columns
  - Compute HAVING (another σ over the groups)
  - Compute SELECT (π) for each group that passes HAVING

HAVING examples

- List the average popularity for each age group with more than a hundred users
  - ```
    SELECT age, AVG(pop)
    FROM User
    GROUP BY age
    HAVING COUNT(*) > 100;
    ```

- Find average popularity for each age group over 10
  - ```
    SELECT age, AVG(pop)
    FROM User
    GROUP BY age
    HAVING age > 10;
    ```

Rules for Having Clause:

(1) Can NOT use Having without Group by.

(2) Group By A_1, A_2, A_3

HAVING A_1, A_3 $agg_1(A_4), agg_2(A_1)$

Aggregation on any attribute

Subset of Group By can be used unaggregated

Not Allowed :

Group By A, B, C

Having A, D, Sum(D), Count(*), Avg(A)

D → Not Allowed

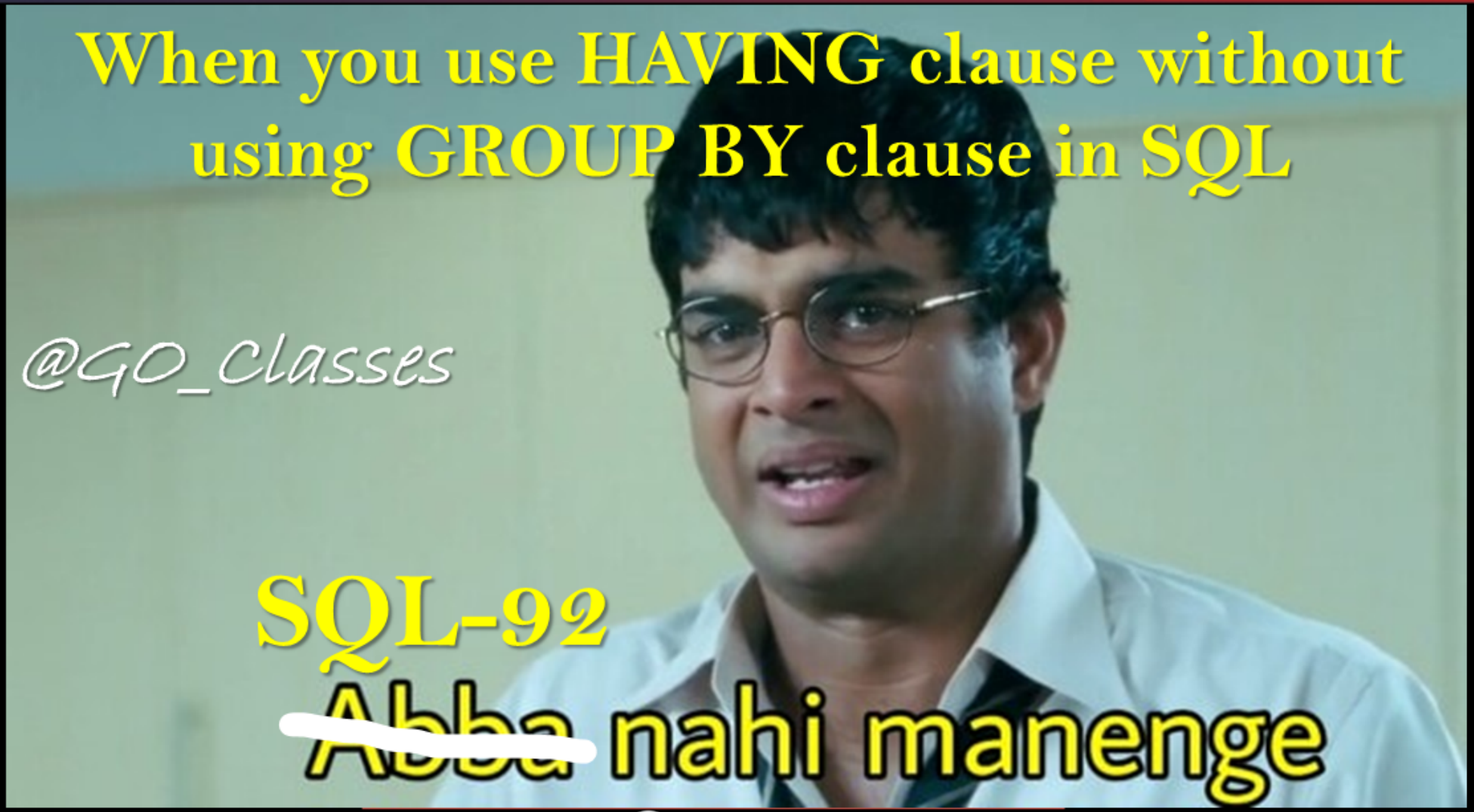
Rules for Having Clause:

There are several rules we must remember about `HAVING` clauses:

- An aggregation in a `HAVING` clause applies only to the tuples of the group being tested.
- Any attribute of relations in the `FROM` clause may be aggregated in the `HAVING` clause, but only those attributes that are in the `GROUP BY` list may appear unaggregated in the `HAVING` clause (the same rule as for the `SELECT` clause).

**SQL-92 Rules for Having Clause:

Can NOT have Having Clause in the Absence of GROUP BY clause.


When you use HAVING clause without using GROUP BY clause in SQL
@GO_Classes
SQL-92
Abba nahi manenge

Describe the output of the following SQL Query.

Consider, for example the relational table

`Student(firstName, lastName, gpa, class, grade, school)`

For example,

```
SELECT school, COUNT(*)
FROM Student
GROUP BY school
HAVING AVG(gpa) > 3.0;
```

Consider, for example the relational table

```
Student(firstName, lastName, gpa, class, grade, school)
```

For example,

```
SELECT school, COUNT(*)
FROM Student
GROUP BY school
HAVING AVG(gpa) > 3.0;
```

returns the number of students enrolled in each school, which has average student GPA over 3.0.

HAVING Clause

The GROUP BY clause transforms the Cartesian Product from a relation of tuples into a relation of groups of tuples.

The HAVING clause is to the groups is what the WHERE clause is to the individual tuples. It provides a condition, and filters out the groups that fail it.

The syntax of HAVING clause is:

```
SELECT b1, ..., bk, AGG1(e1) AS x1, ..., AGGm(em) AS xm
FROM R
GROUP BY b1, ..., bk
HAVING C;
```

(Q31) Find the age of the youngest sailor for each rating level.

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

(Q31) Find the age of the youngest sailor for each rating level.

Group by

Phrase for Grouping

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

```
SELECT rating, min(age)
FROM Sailors
Group by rating
```

Next Sub-Topic:

SQL

Order By Clause

If we want the FINAL Output "to be displayed" in SORTED order, we can use ORDER BY clause.

By default, Ordering is in the Ascending Order.

Can use DESC for Descending Order.

```
SELECT <attribute and function list>
FROM <table list>
[ WHERE <condition> ]
[ GROUP BY <grouping attribute(s)> ]
[ HAVING <group condition> ]
[ ORDER BY <attribute list> ];
```

Conceptual Evaluation :

1. From
2. WHERE
3. Group by
4. Having
5. SELECT
6. DISTINCT
7. Order by

SELECT A, B, - - - -

≡

ORDER By A, B DESC

R

A	B
1	'a'
2	'b'
3	'a'
1	'b'
2	'a'
1	'c'

r_1 r_2 r_3 r_4 r_5 r_6

SELECT *
ORDER By A ;

≡ ORDER By A Asc ;

1	c
1	a
1	b
2	a
2	b
3	a

} fine

} fine

Multiple Results possible.

R

	A	B
r_1	1	'a'
r_2	2	'b'
r_3	3	'a'
r_4	1	'b'
r_5	2	'a'
r_6	1	'c'

SELECT *
FROM R
order by B $\equiv$ order by B Asc

3	a
1	a
2	a
2	b
1	b
1	c

multiple Results Possible.

R

A	B
1	'a'
2	'b'
3	'a'
1	'b'
2	'a'
1	'c'

r_1 r_2 r_3 r_4 r_5 r_6

Order by A, B DESC ;

1 c
1 b
1 a
2 b
2 a
3 a

must

must

No other order possible

R

A	B
1	'a'
2	'b'
3	'a'
1	'b'
2	'a'
1	'c'

r_1 r_2 r_3 r_4 r_5 r_6

Order by B Desc, A Asc;

1	c
1	b
2	b
1	a
2	a
3	a

→ No other order possible.

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.

3.4.4 Ordering the Display of Tuples

SQL offers the user some control over the order in which tuples in a relation are displayed. The **order by** clause causes the tuples in the result of a query to appear in sorted order. To list in alphabetic order all instructors in the Physics department, we write:

select *name*
from *instructor*
where *dept_name* = 'Physics'
order by *name*;

By default, the **order by** clause lists items in ascending order. To specify the sort order, we may specify **desc** for descending order or **asc** for ascending order. Furthermore, ordering can be performed on multiple attributes. Suppose that we wish to list the entire *instructor* relation in descending order of *salary*. If several

instructors have the same salary, we order them in ascending order by name. We express this query in SQL as follows:

```
select *
from instructor
order by salary desc, name asc;
```

The order-item-list is a list of order-items; an order-item is a column name, optionally followed by one of the keywords ASC or DESC. Every column mentioned in the ORDER BY clause attributes must also appear in the select-list of the query; otherwise it is not clear what columns we should sort on.

The keywords ASC or DESC that follow a column control whether the result should be sorted—with respect to that column—in ascending or descending order; the default is ASC. This clause is applied as the last step in evaluating the query.

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 in 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.

HW: Number of tuples in the Output ??

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1
```

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
~~71~~	~~zorba~~	~~10~~	~~16.0~~
64	horatio	7	35.0
29	brutus	1	33.0
58	rusty	10	35.0

```
SELECT  S.rating,  MIN (S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1
```

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
71	zorba	10	16.0
64	horatio	7	35.0
29	brutus	1	33.0
58	rusty	10	35.0

2

rating	age
1	33.0
7	45.0
7	35.0
8	55.5
10	35.0

3

rating	m-age	count
1	33.0	1
7	35.0	2
8	55.0	1
10	35.0	1

rating	
7	35.0

Next Topic:

SQL

Nested Queries

Nested Query:

Query within Query.

6.3 Subqueries

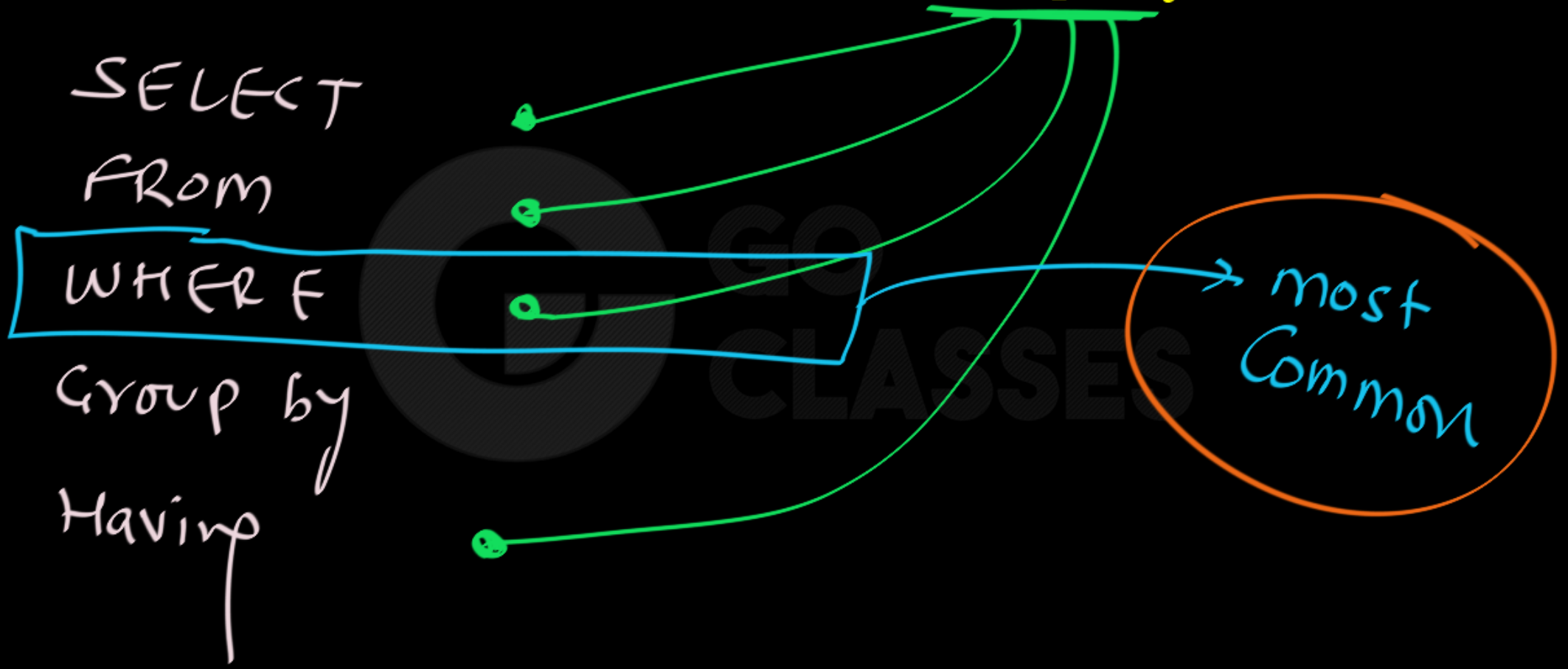
In SQL, one query can be used in various ways to help in the evaluation of another. A query that is part of another is called a *subquery*. Subqueries can have subqueries, and so on, down as many levels as we wish.

Main Query

```
SELECT  M.StarID, M.Name
FROM    MovieStar M
WHERE   M.StarID IN (SELECT  S.StarID
                     FROM  StarsIn S
                     WHERE  MovieID=28)
```

Subquery

Where can we insert a Subquery ??



Nested Queries

- A very powerful feature of SQL:

```
Select  A1,A2,…,An
From    R1,R2, …,Rm
where   condition
```

- A nested query is a query that has another query embedded with it.
  - A **SELECT, FROM, WHERE, or HAVING** clause can itself contain an SQL query!
  - Being part of the WHERE clause is the most common

Two types of Sub-queries:

1. Simple Subquery:

Query without Co-relation with surrounding query..(NO dependence from outer query)

2. Co-related Subquery:

Query with Co-relation with surrounding query..(Dependence from outer query)

Movie(MovieID, Title, Year)
StarsIn(MovieID, StarID, role)
MovieStar(StarID, Name, Gender)

Simple subquery

```
SELECT  M.StarID, M.Name
FROM    MovieStar M
WHERE   M.StarID IN (SELECT  S.StarID
                     FROM  StarsIn S
                     WHERE  MovieID=28)
```

Subquery

Main query

Movie(MovieID, Title, Year)
StarsIn(MovieID, StarID, role)
MovieStar(StarID, Name, Gender)

Scoping Rule :
any attribute belongs to Closest surrounding query.

```
SELECT [redacted]
FROM   movie
WHERE  [redacted] IN (SELECT S.StarID
                      FROM StarsIn S
                      WHERE MovieID=28)
```

Main Query

Correlated Subquery

```
SELECT cName, state
FROM College C1
WHERE exists (SELECT *
              FROM College C2
              WHERE C2.state = C1.state AND
                    C2.cName <> C1.cName);
```

attribute of outer Query

```
SELECT cName, state
FROM College C1
WHERE exists (SELECT *
              FROM College C2
              WHERE C2.state = C1.state AND
                    C2.cName <> C1.cName);
```

"Evaluation of Subquery" in the WHERE clause:

"Evaluation of Subquery" in the WHERE clause:

1. Simple Subquery in the WHERE clause:
(Query without Co-relation)

The simplest subqueries can be evaluated once and for all, and the result used in a higher-level query.

"Evaluation of Subquery" in the WHERE clause:

2. Co-related Subquery:

Query with Co-relation with surrounding query..(Dependence from outer query)

Re-Computed for EVERY Row of Outer Table.

Take rows of Outer query one-by-one & evaluate the corelated subquery each time.

Nested Query concept is BEST Understood with the help of Examples.

But first, let us Study some SQL keywords.

Next Sub-Topic:

SQL

Understanding Some SQL Keywords

Some SQL Keywords

Set Comparator Keywords:

IN, ALL, EXISTS, ANY

NOT IN, NOT ALL, NOT EXISTS, NOT ANY

Next Sub-Topic:

SQL

Some SQL Keywords

IN, NOT IN

IN ≡ ∈ ≡ membership keyword

NOT IN : ∉

S : a Tuple ; R : a subquery

S IN R : True iff Tuple S is in R.

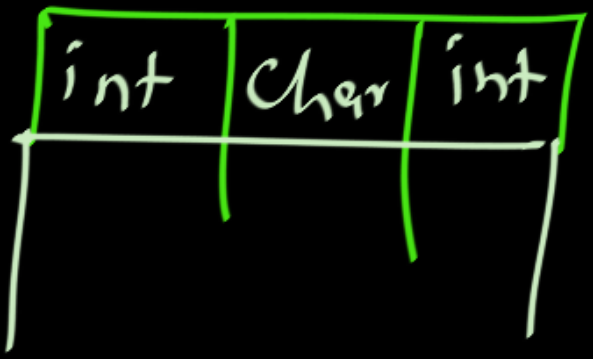
S NOT IN R : True iff Tuple S is NOT in R

S : a Tuple ; R : a subquery

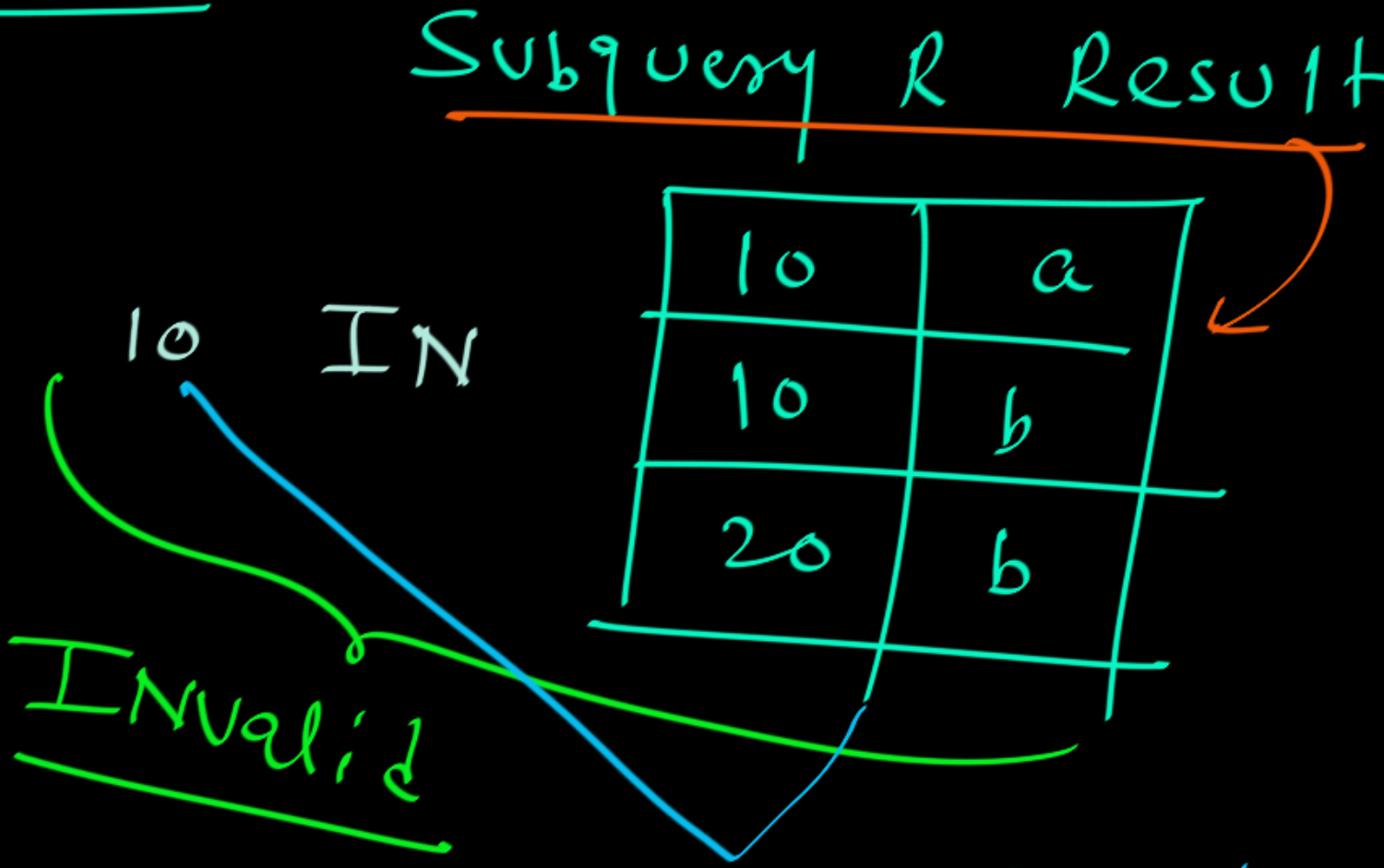
S IN R
S NOT IN R

} S, R must be Compatible.

Tuple S (10, 'a', 20) then R



Note :



1. IN , NOT IN keywords:

(set membership keywords)

The IN operator allows us to test whether a value or tuple is in a given set/bag;

s IN R is true if and only if **s** is equal to one of the tuples in **R**. Likewise,

s NOT IN R is true if and only if **s** is equal to no tuple in **R**.

s `IN` R is true if and only if s is equal to one of the values in R. Likewise, s `NOT IN` R is true if and only if s is equal to no value in R.

- With `IN` ($\in$) and `NOT IN` ($\notin$) it is possible to check whether an attribute value appears in a set of values computed by another SQL **subquery.**

R

A	B
10	a
20	a
20	c
30	a

```
SELECT *
FROM R
WHERE (A, B) IN R;
```

Invalid

must be a subquery

R

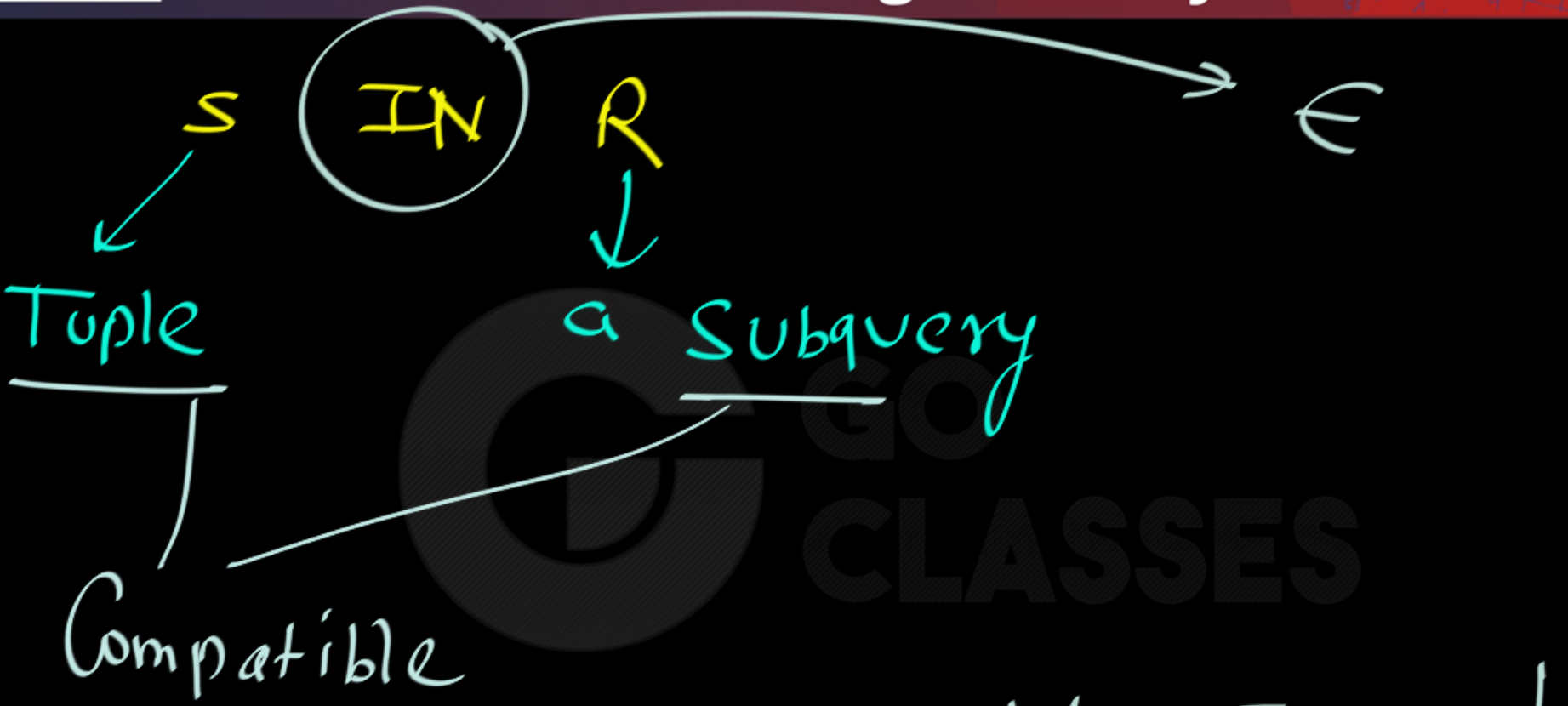
A	B
10	a
20	a
20	c
30	a

1. SELECT *
   FROM R
2. WHERE (A, B) IN (SELECT * FROM R)

Applies Tuple by Tuple

Subquery ≡ R

Final o/p ≡ Same as R



NOT IN ≡ ∉

STUDENTS

SID	FIRST	LAST	EMAIL
101	Ann	Smith	...
102	Michael	Jones	(null)
103	Richard	Turner	...
104	Maria	Brown	...

RESULTS

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

EXERCISES

CAT	ENO	TOPIC	MAXPT
H	1	Rel.Alg.	10
H	2	SQL	10
M	1	SQL	14

```
SELECT   FIRST, LAST
FROM     STUDENTS
WHERE    SID NOT IN (SELECT   SID
                     FROM     RESULTS
                     WHERE    CAT = 'H')
```

What is the output of this Query ??

```
SELECT   FIRST, LAST
FROM     STUDENTS
WHERE    SID NOT IN (SELECT   SID
                     FROM     RESULTS
                     WHERE    CAT = 'H')
```

Scoping Rule

STUDENTS

SID	FIRST	LAST	EMAIL
101	Ann	Smith	...
102	Michael	Jones	(null)
103	Richard	Turner	...
104	Maria	Brown	...

pass

EXERCISES

CAT	ENO	TOPIC	MAXPT
H	1	Rel.Alg.	10
H	2	SQL	10
M	1	SQL	14

RESULTS

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

```
SELECT   FIRST, LAST
FROM     STUDENTS
WHERE    SID NOT IN (SELECT   SID
                     FROM     RESULTS
                     WHERE    CAT = 'H')
```

{101, 101, 102, 102, 103}

Bag

Tuple by Tuple

Subquery : Simple

Execute once

o/p:

Maria	Brown

Students without any homework result.

```
SELECT  FIRST, LAST
FROM    STUDENTS
WHERE   SID NOT IN (SELECT  SID
                    FROM    RESULTS
                    WHERE   CAT = 'H')
```

FIRST	LAST
Maria	Brown

Q: Consider the following Query on relation R (A,B).

```
SELECT *
FROM R
WHERE (A,B)  IN ( R )
```

What is the output of it??

Q: Consider the following Query on relation R (A,B).

```
SELECT *
FROM R
WHERE (A,B) IN ( R )
```

Invalid

must be a Subquery

What is the output of it??

Consider the following Query: R (A,B)

```
SELECT *
FROM R
WHERE (A,B) IN ( SELECT *
                 FROM R )
```

Simple Subquery

≡ R

o/p: Same as R ✓

The IN operator allows us to test whether a value is in a given set of elements; an SQL query is used to generate the set to be tested.

There are a number of SQL operators that we can apply to a relation R and produce a boolean result. However, the relation R must be expressed as a subquery. As a trick, if we want to apply these operators to a stored table `Foo`, we can use the subquery `(SELECT * FROM Foo)`. The same trick works for union, intersection, and difference of relations.

IN subqueries

- *x* IN (*subquery*) checks if *x* is in the result of *subquery*
- Example: users at the same age as (some) Bart
  - ```
    SELECT *
    FROM User
    WHERE age IN (SELECT age
                  FROM User
                  WHERE name = 'Bart');
    ```

What's Bart's age?
    ```

Next Sub-Topic:

SQL

Co-Related Sub-queries or Correlated Nested Queries

"Evaluation of Subquery" in the WHERE clause:

2. Co-related Subquery:

Query with Co-relation with surrounding query..(Dependence from outer query)

Re-Computed for EVERY Row of Outer Table.

Take rows of Outer query one-by-one & evaluate the corelated subquery each time.

2. Suppose relations R(A,B) and S(B,C,D) are as follows:

R =

A	B
1	2
3	4
5	6

S =

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

Example:

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

2. Suppose relations R(A,B) and S(B,C,D) are as follows:

R =

A	B
1	2
3	4
5	6

S =

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

Example:

main Query

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

Subquery : Corelated Subquery

2. Suppose relations R(A,B) and S(B,C,D) are as follows:

R =

A	B
1	2
3	4
5	6

S =

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

Example:

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

Tuple by Tuple

for every Tuple of R, Calculate inner query.

2. Suppose relations R(A,B) and S(B,C,D) are as follows:

fails

R =

A	B
1	2
3	4
5	6

S =

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

Example:

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

1 2

Checking for Tuple (1,2)

Count(*)
1

2. Suppose relations R(A,B) and S(B,C,D) are as follows:

Passes

R =

A	B
1	2
3	4
5	6

S =

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

Example:

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

3 → A

4 → R.B

Check for Tuple (3,4)

Count(*)
2

2. Suppose relations R(A,B) and S(B,C,D) are as follows:

R =

A	B
1	2
3	4
5	6

S =

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

Passed

Example:

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

A → 5

R.B → 6

Check for Tuple (5,6)

count(*)
0

2. Suppose relations R(A,B) and S(B,C,D) are as follows:

R =

A	B
1	2
3	4
5	6

S =

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

Example:

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

o/p:

A	B
3	4
5	6

Next Sub-Topic:

SQL

Some SQL Keywords

EXISTS , NOT EXISTS

EXIST : NOT a SQL keyword

EXISTS : keyword

Syntax :

EXISTS R → Subquery

↳ True iff R is NOT Empty.

NOT EXISTS R → Subquery : True iff R is Empty.

1. EXISTS , NOT EXISTS keywords:

(existence of a tuple)

EXISTS R is a condition that is true if and only if R is not empty.

NOT EXISTS R is true if and only if R is empty.

EXISTS R is a condition that is true if and only if R is not empty.

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

Example:

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

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

Example:

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

Subquery : Corelated

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

Example:

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

Applies Tuple by Tuple of R

Example:

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

Check Tuple (1,2)

True

Passes

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

C	D
4	12
6	14
8	16

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

A	B
1	2
3	4
5	6
7	8

R =

C	D
2	10
4	12
6	14
8	16

S =

Pass

Example:

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

Check Tuple (3,4)

True

C	D
6	14
8	16

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

Passes

Example:

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

Check Tuple (5,6)

True

6

C	D
8	16

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

fails

Example:

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

check Tuple (7,8)

false

8

Empty Table

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

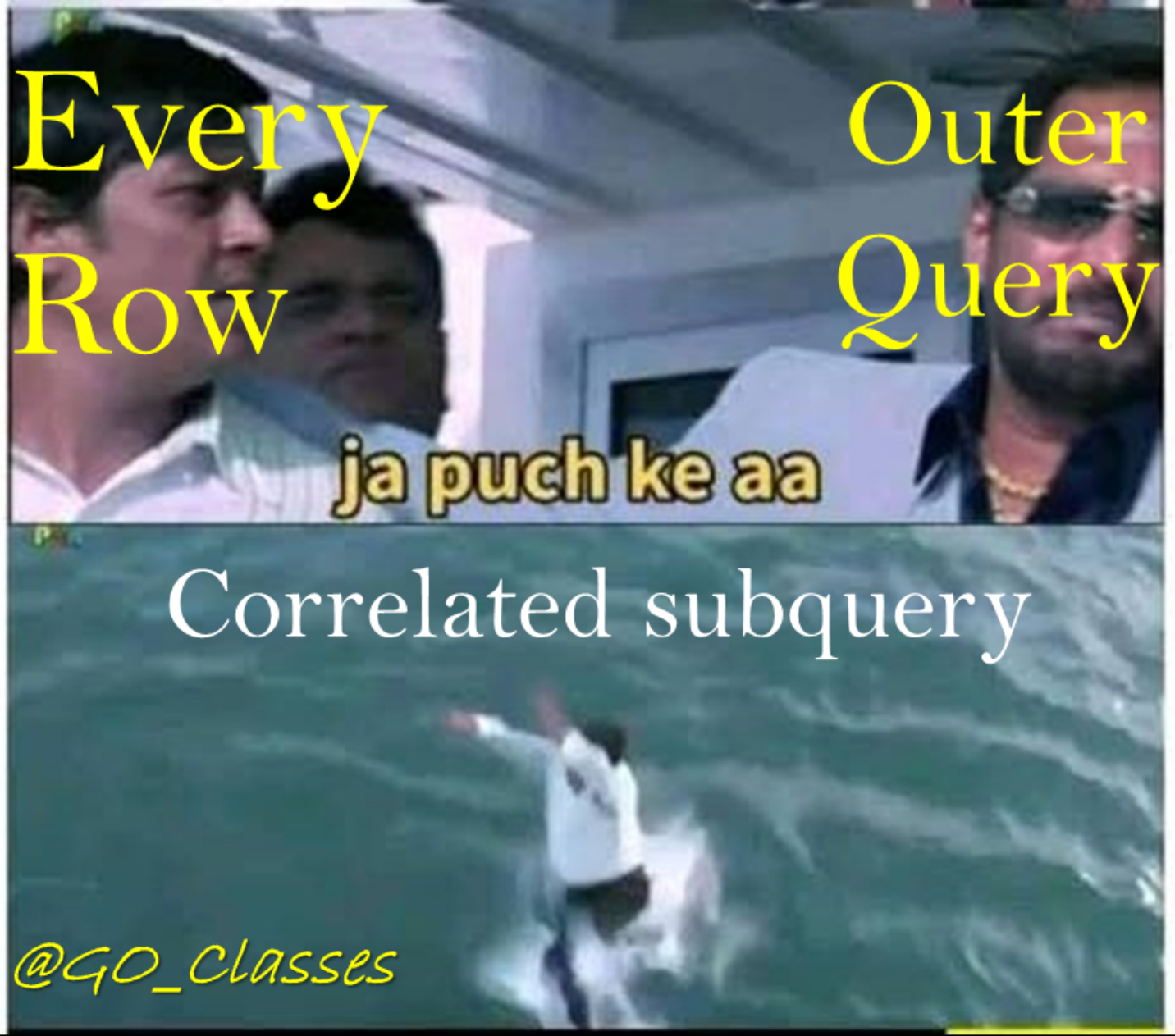
C	D
2	10
4	12
6	14
8	16

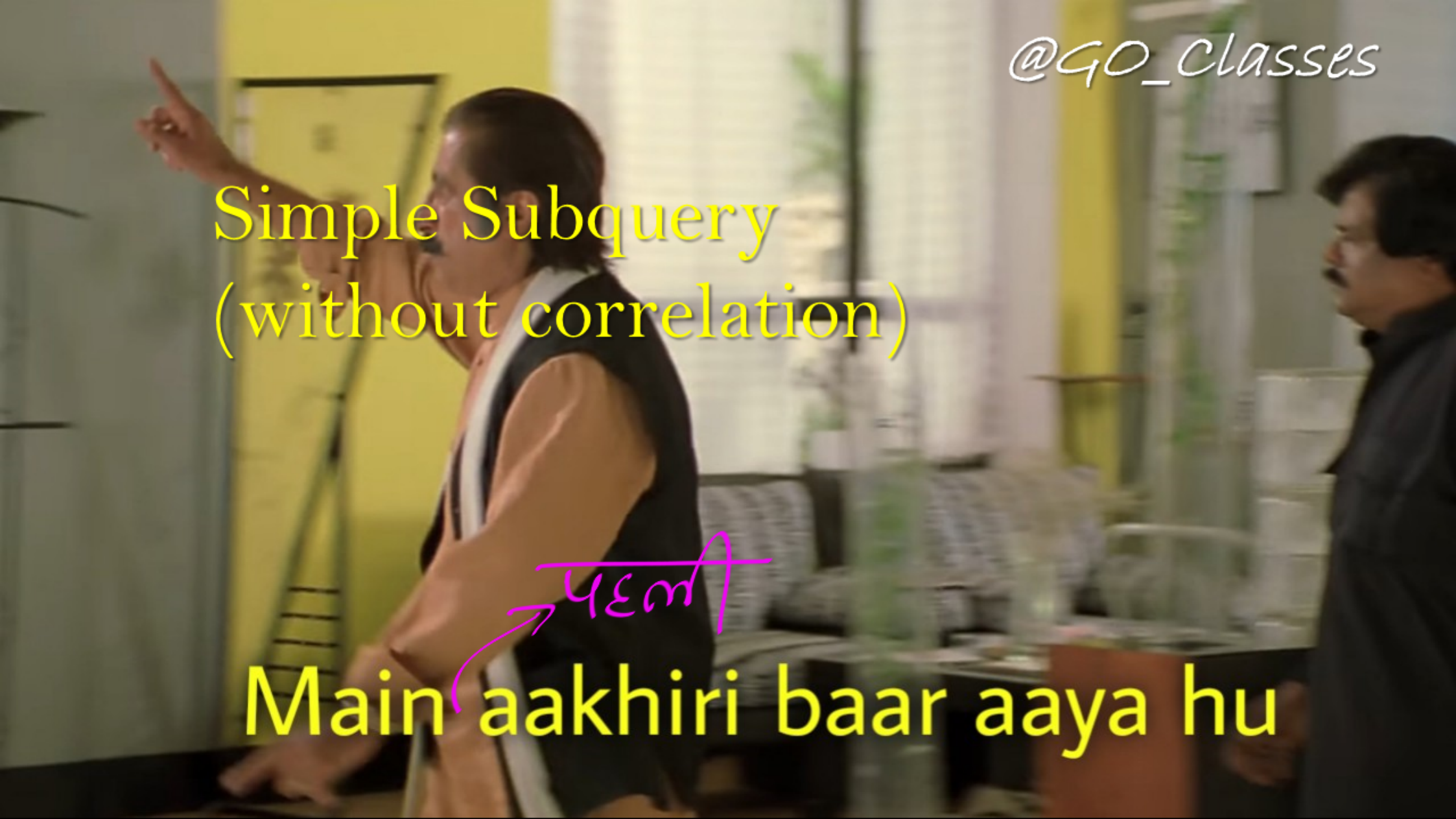
o/p:

A	B
1	2
3	4
5	6

Example:

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




@GO_Classes
Simple Subquery
(without correlation)
पहली
Main aakhiri baar aaya hu

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

Example:

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

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

Example:

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

Main query

Correlated Subquery

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

Passes

Example:

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

Tuple by Tuple

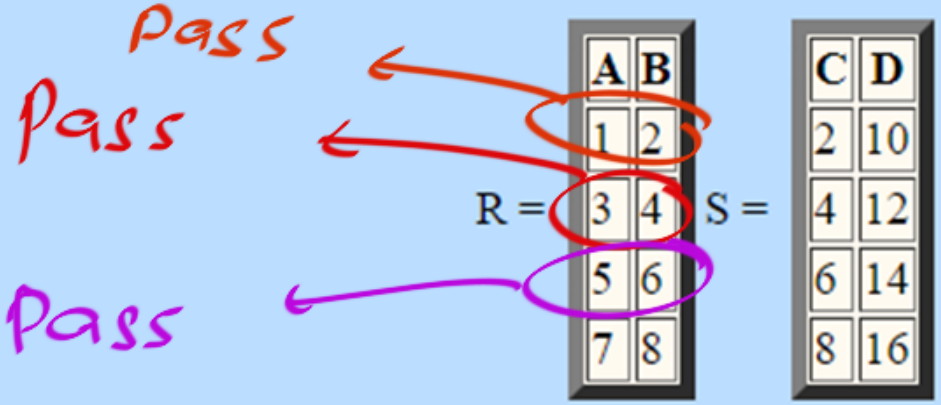
(1,2) Tuple

True

2

count(*)
3

6. Here are two relations, R(A,B) and S(C,D). Their current values are:



Example:

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

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

Pass

Example:

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

Check Tuple (7,8)

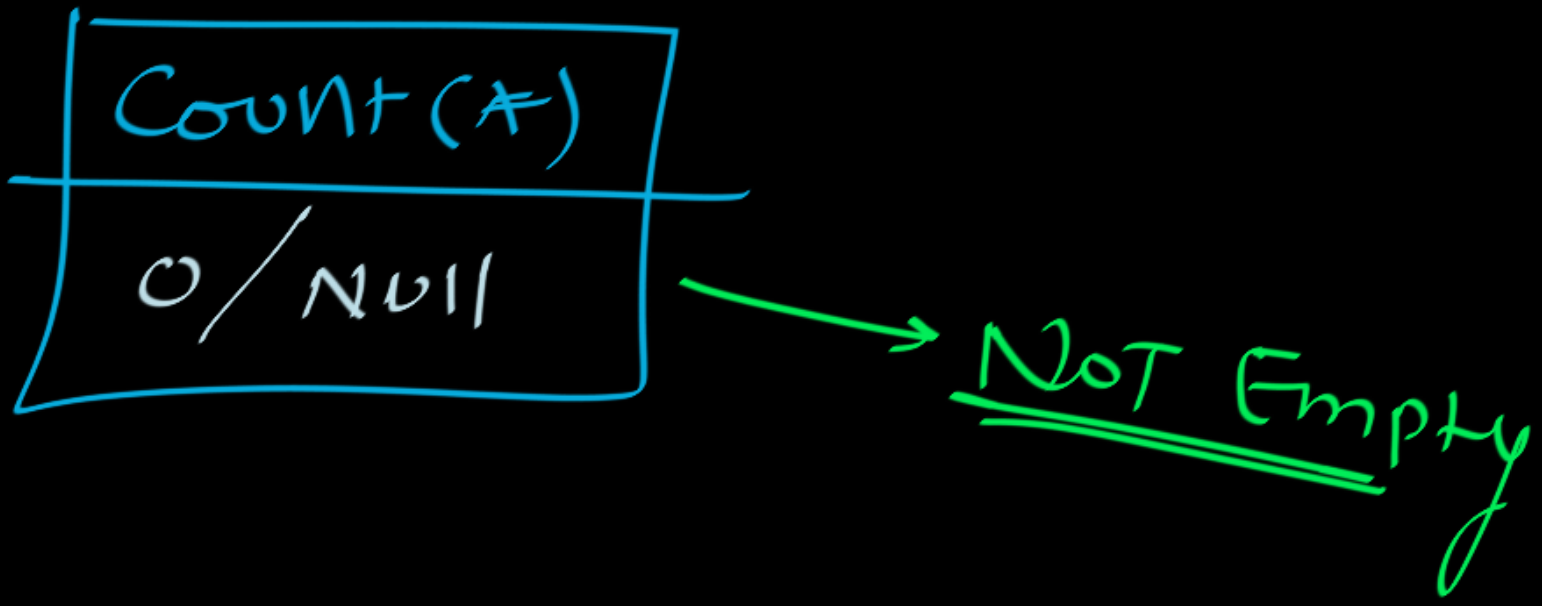
True

8

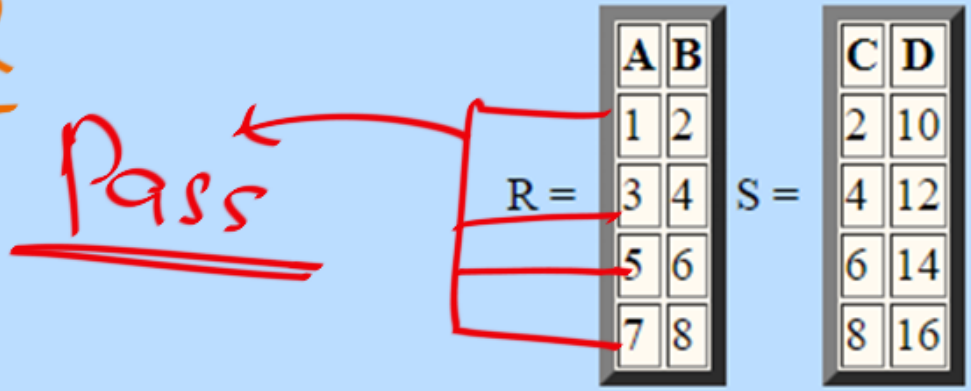
count(*)
0

Not Empty

Empty Table : Containing No Tuple



6. Here are two relations, R(A,B) and S(C,D). Their current values are:



Pass

Example:

o/p = R

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

Always True

Never gives Empty Table

Whatever A :

→ Empty

Count (*)
Count (A)
Sum (A)
Avg (A)
Min (A)
Max (A)

→ Never Give Empty Result.

Count(*) = 0
Count (A) = 0
Sum (A) = Null
Min (A) = Null

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

Example:

```
SELECT *
FROM R
WHERE NOT EXISTS ( SELECT *
                   FROM S
                   WHERE B < C)
```

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

Pass

o/p

A	B
7	8

Example:

```
SELECT *
FROM R
WHERE NOT EXISTS ( SELECT *
                   FROM S
                   WHERE B < C)
```

Tuple by Tuple

True when Subquery Result is Empty

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

Example:

```
SELECT *
FROM  R
WHERE  NOT EXISTS ( SELECT count(*)
                    FROM S
                    WHERE  B < C)
```

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

Empty Table

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

All fail

Example: o/p =

```
SELECT *
FROM R
WHERE NOT EXISTS ( SELECT count(*)
                   FROM S
                   WHERE B < C)
```

Always False

Never Empty Table

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

Example:

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

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

Example:

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

No aggregate

a Table

Invalid Query

Invalid

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

Example:

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

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

Simple aggregates

Example:

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

Simple Column, Simple Tuple

DBMS Converts this Table into a Value

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

Example:

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

Subquery : Correlated

6. Here are two relations, R(A,B) and S(C,D). Their current values are:

R =

A	B
1	2
3	4
5	6
7	8

S =

C	D
2	10
4	12
6	14
8	16

Pass

Example:

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

Tuple by Tuple

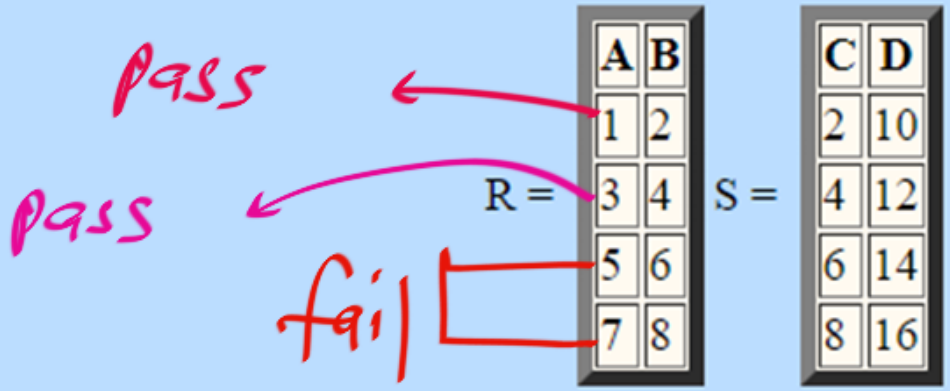
Count(*)
3

≥ 2

Value 3

DBMS internally

6. Here are two relations, R(A,B) and S(C,D). Their current values are:



Example:

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

o/p

A	B
1	2
3	4

Next Sub-Topic:

SQL

Some SQL Keywords

UNIQUE, NOT UNIQUE

Closely related to EXISTS is the UNIQUE predicate.

When we apply UNIQUE to a subquery, it returns true if no row appears twice in the answer to the subquery, that is, there are no duplicates;

Syntax:

Unique R → Subquery : True iff R has No Duplicate Tuples.

NOT Unique R → Subquery : True iff R has Some Duplication.

Next Sub-Topic:

SQL

Some SQL Keywords

ALL , ANY/SOME

ALL

ANY / SOME

Keywords

Suggestion:

~~ANY~~ SOME

Syntax:

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

$S \; < \; ALL \; R$

Universal Quantifier ($\forall$)

If R Empty then "ALL" is Always True.

Syntax:

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

$S \; < \; \cancel{\text{ANY}} \; R$: S must be less than Some value of R

Some

at least one

Syntax:

Some → at least one

R → If R Empty then "Some" is Always false

x → Existential Quantifier

Syntax:

NOT $s < ALL\ R$: True if s is $\geq$ some value in R.

Syntax:

NOT S $\leq$ ~~Any~~ Some R : S is greater than All values of R.

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 `>` `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 `<>` `ALL` R is the same as s `NOT IN` R.

4. s `>` `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 `=` `ANY` R is the same as s `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 `>=` `ALL` R is true if and only if s is not the maximum value in R, and `NOT` s `>` `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.

Next Sub-Topic:

SQL

Some more points...

Note that :

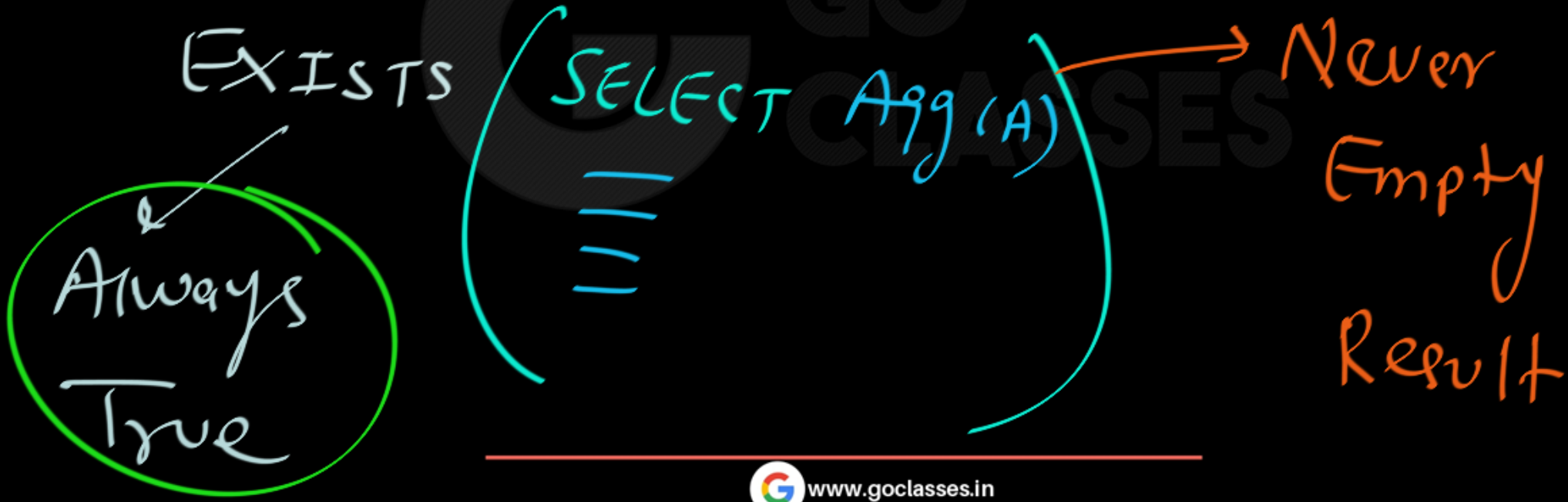
IN and NOT IN are equivalent to = ANY and <> ALL, respectively.

S IN R $\equiv$ S = ANY R

S NOT IN R $\equiv$ S <> ALL R

NOTE:

EXISTS and Aggregate in the inner query : Take Care



Clicker nested question

co-related query - apply subquery on each tuple

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

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 )
```

Correlated Subquery

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

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 )
```

fails

t1:

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

tuple t1 : 4 ≤ ALL

all Runs made on Sunday: 4, 9, 2, 7, 2, 4

GO Classes

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 )
```

3 Tuples in o/p

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

We want those Tuples where Runs are Least on that Day.

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

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

Correct

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 pairs such that the team scored the minimum number of runs for that day.

Semantics of subqueries

- ```
  SELECT *
  FROM Users AS u
  WHERE EXISTS (SELECT * FROM User
                WHERE name = 'Bart'
                AND age = u.age);
  ```

- For each row `u` in `User`
  - Evaluate the subquery with the value of `u.age`
  - If the result of the subquery is not empty, output `u.*`
- The DBMS query optimizer may choose to process the query in an equivalent, but more efficient way
  ```

Scoping rule of subqueries

- To find out which table a column belongs to
  - Start with the immediately surrounding query
  - If not found, look in the one surrounding that; repeat if necessary
- Use *table_name*.*column_name* notation and AS (renaming) to avoid confusion

Next Sub-Topic:

SQL

Joins in SQL

5.1.6 Joins of Bags

Joining bags presents no surprises. We compare each tuple of one relation with each tuple of the other, decide whether or not this pair of tuples joins successfully, and if so we put the resulting tuple in the answer. When constructing the answer, we do not eliminate duplicate tuples.

Joins in Rel. Alg ≡ Joins in SQL

↳ Duplicates NOT Eliminated

A	B
1	2
1	2

(a) The relation R

B	C
2	3
4	5
4	5

(b) The relation S

A	$R.B$	$S.B$	C
1	2	2	3
1	2	2	3
1	2	4	5
1	2	4	5
1	2	4	5
1	2	4	5

(c) The product $R \times S$

Figure 5.3: Computing the product of bags

Table R : i attributes, m tuples

S : j attributes, n tuples

R x S : i + j attributes, mn tuples

A	B
1	2
1	2

(a) The relation R

B	C
2	3
4	5
4	5

(b) The relation S

$R \bowtie S$:

Natural Join :

A	B	C
1	2	3
1	2	3

That is, tuple $(1, 2)$ of R joins with $(2, 3)$ of S. Since there are two copies of $(1, 2)$ in R and one copy of $(2, 3)$ in S, there are two pairs of tuples that join to give the tuple $(1, 2, 3)$. No other tuples from R and S join successfully.

A	B
1	2
1	2

(a) The relation R

B	C
2	3
4	5
4	5

(b) The relation S

As another example on the same relations R and S, the theta-join

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

produces the bag

A	$R.B$	$S.B$	C
1	2	4	5
1	2	4	5
1	2	4	5
1	2	4	5

Clicker outer join question

- Given:
  Compute:
  SELECT R.A, R.B, S.B, S.C, S.D
  FROM R FULL OUTER JOIN S
  ON (R.A > S.B AND R.B = S.C)
- Which of the following tuples of R or S is dangling (and therefore needs to be padded in the outer join)?

A. (1,2) of R
B. (3,4) of R
C. (2,4,6) of S
D. All of the above
E. None of the above

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

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

Clicker outer join question

- Given:
  Compute:
  SELECT R.A, R.B, S.B, S.C, S.D
  FROM R FULL OUTER JOIN S
  ON (R.A > S.B AND R.B = S.C)
- Which of the following tuples of R or S is dangling (and therefore needs to be padded in the outer join)?

A. (1,2) of R
B. (3,4) of R
C. (2,4,6) of S
D. All of the above
E. (4,7,9) of S

Dangling Tuple

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

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

Dangling

4 Tuples in o/p

R ⟗ S

R.A > S.B & R.B = S.C

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

Null Null 4 7 9

Dangling Tuple:

In the Join operations,

A tuple that fails to pair with any tuple of the other relation in a join is said to be a dangling tuple.

Clicker outer join question

- Given:
  Compute:
  SELECT R.A, R.B, S.B, S.C, S.D
  FROM R FULL OUTER JOIN S
  ON (R.A > S.B AND R.B = S.C)
- Which of the following tuples of R or S is dangling (and therefore needs to be padded in the outer join)?

A. (1,2) of R
B. (3,4) of R
C. (2,4,6) of S
D. All of the above
E. None of the above

A is correct

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

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

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

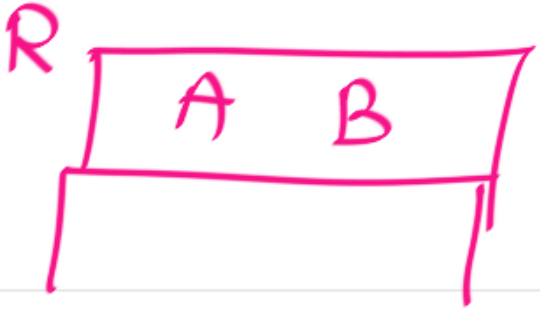
5.39.1 Sql: UGC NET CSE | August 2016 | Part 2 | Question: 20 top

Consider a database table R with attributes A and B. Which of the following SQL queries is illegal ?

A. SELECT A FROM R;
B. SELECT A, COUNT(*) FROM R;
C. SELECT A, COUNT(*) FROM R GROUP BY A;
D. SELECT A, B, COUNT(*) FROM R GROUP BY A, B;

ugcnetcse-aug2016-paper2 databases sql

UGC NET CSE | August 2016 | Part 2 | Question: 20



asked Sep 24, 2016 • recategorized Oct 14, 2018 by **Pooja Khatri**

1,862 views

2

Consider a database table R with attributes A and B. Which of the following SQL queries is illegal ?

A. SELECT A FROM R;
B. SELECT A, COUNT(*) FROM R;
C. SELECT A, COUNT(*) FROM R GROUP BY A;
D. SELECT A, B, COUNT(*) FROM R GROUP BY A, B;

Invalid

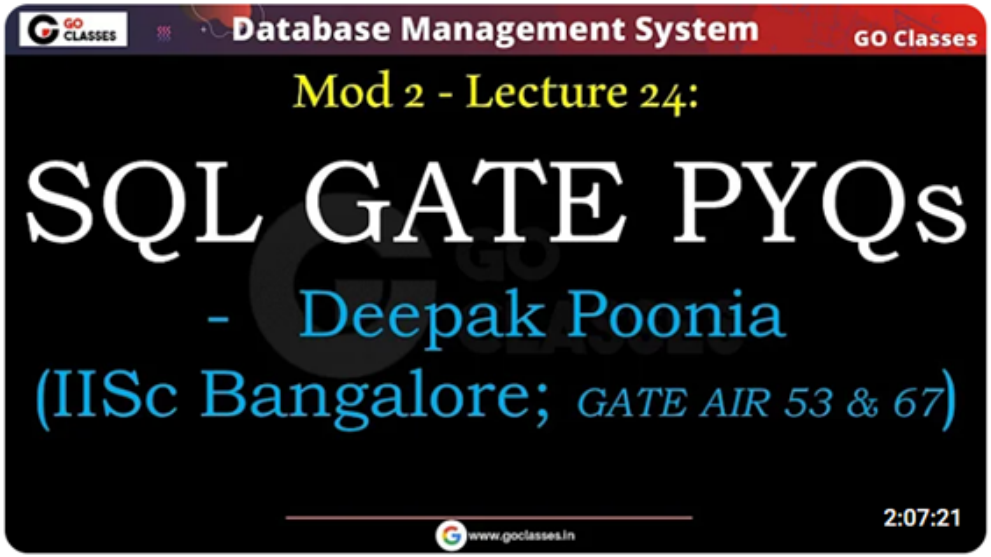
SELECT A, Count(*)
FROM R

illegal

Databases #ugcnetcse-aug2016-paper2 #databases #sql

Premium IN

go classes sql



SQL GATE Questions - Complete Analysis | DBMS GATE CS PYQs | GO Classes | Deepak Poonia

4.4K views • 1 year ago

GO Classes for GATE CS

SQL GATE Questions - Complete Analysis | DBMS GATE CS PYQs | GO Classes | Deepak Poonia Crack #GateCSE

11 chapters SQL GATE Questions Session Begins | GATE CSE 2000 | Question: 2.26 - SQL...