SQL

Complete Summary

Part – 2

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 – 2

5.1.4 Selection on Bags

To apply a selection to a bag, we apply the selection condition to each tuple independently. As always with bags, we do not eliminate duplicate tuples in the result.

Example 5.5: If R is the bag

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

then the result of the bag-selection $\sigma_{C \geq 6}(R)$ is

A	B	C
3	4	6
1	2	7
1	2	7

SELECT *
FROM R
WHERE C ≥ 6

That is, all but the first tuple meets the selection condition. The last two tuples, which are duplicates in R, are each included in the result. □

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

Next Topic:

SQL

NULL values

Why Use NULLs

The three meanings of NULL:

- Unknown values
- Inapplicable values
- Empty placeholders

Three meanings of null values

1. not applicable
2. not known
3. absent (not recorded)

The NULL Spouse Example

If *employee.spouse* is NULL, does it mean?

- The spouse's name is unknown.
- The employee is not married and therefore has no spouse.

NOTE:

Important rule to remember when we operate upon a NULL value:

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

50 + Null = Null

↙

arithmetic operation

Null × 0 = Null

Null − 0 = Null

Null − Null = Null

Null → (arithmetic opn) → anything

↓

Null or Not Null

Null

0 / Null = Null

50 + Null = Null

Unknown value / not applicable value

Null × 0 = Null

50 × Null = Null

Null ÷ Null = Null

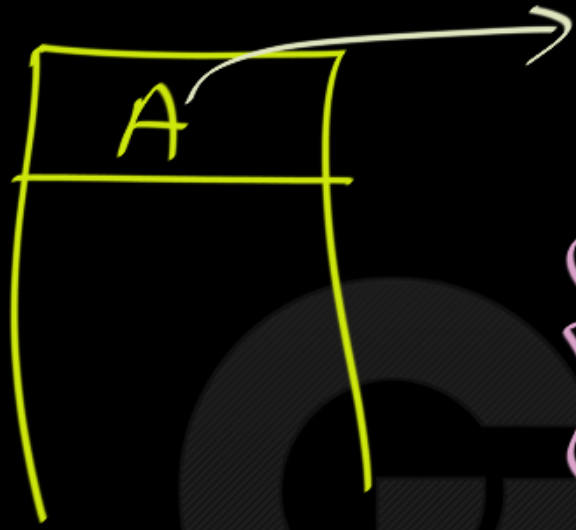
Null + anything = Null

Null → Arithmatic operation (+, −, ×, ÷, ...)

Null − 50 = Null

Null − 0 = Null

Q: R



Domain = Integer

Null − 0 = Null

Null × 0 = Null

Pitfalls Regarding Nulls

It is tempting to assume that NULL in SQL can always be taken to mean "a value that we don't know but that surely exists." However, there are several ways that intuition is violated. For instance, suppose x is a component of some tuple, and the domain for that component is the integers. We might reason that $0 * x$ surely has the value 0, since no matter what integer x is, its product with 0 is 0. However, if x has the value NULL, rule (1) of Section 6.1.6 applies; the product of 0 and NULL is NULL. Similarly, we might reason that $x - x$ has the value 0, since whatever integer x is, its difference with itself is 0. However, again the rule about operations on nulls applies, and the result is NULL.

Next Topic:

SQL

Aggregate Operators in case of NULL values

NOTE:

Aggregate functions ignore the NULL values in the Input relation.

Except count(*) which ALWAYS counts the total number of rows in the table.

R

A
2
-3
Null
0
Null

Count(A) = 3
Sum(A) = -1
Max(A) = 2
Min(A) = -3
Avg(A) = -1/3

→ behave in same way.

→ Ignore Null values.

Count(*) = 5

R

A
2
-3
Null
0
Null
2
-3

Count (Distinct A) = 3

Sum (Distinct A) = -1

Max (Distinct A) = max (A) = 2

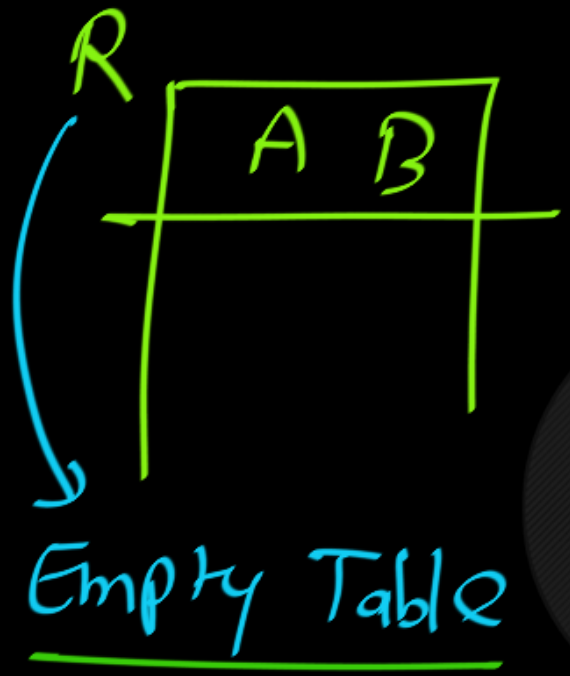
Min (Distinct A) = min (A) = -3

Avg (Distinct A) = -1/3

Count (*) = 7

Aggregate functions on Empty Table:

- 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.



Count(*) = 0

Count(A) = 0

Sum(A) = Null

Avg(A) = Null

Max(A) = Null

Min(A) = Null

Count(Distinct A) = 0

Sum(Distinct A) = Null

Min(Distinct A) = Null

Count(Distinct *)
Count (* A)

Invalid

attribute

Count (*) = number of Rows in Table.
(Always)

Example 6.33 : Suppose we have a relation $R(A, B)$ with one tuple, both of whose components are NULL:

Sum(A) = ?

A	B
NULL	NULL

① Ignores Null Values

Example 6.33: Suppose we have a relation $R(A, B)$ with one tuple, both of whose components are NULL:

Sum(A) = ? = Null

A	B
NULL	NULL

After ignoring Null values of A, we get

Empty Table ⟹ Sum(A) = Null

Count(*) = 1

Example 6.33: Suppose we have a relation $R(A, B)$ with one tuple, both of whose components are NULL:

A	B
NULL	NULL

Sum(A) = Null

Count(A) = 0

Min(A) = Null

Avg(Distinct A) = Null

R

A	B
Null	2
Null	Null
Null	2

Count(*) = 3

Count(* B) = Invalid

Count(A) = 0

Count(B) = 2

Count(Distinct B) = 1

Sum(A) = Null

Sum(Distinct B) = 2

3.7.4 Aggregation with Null and Boolean Values

Null values, when they exist, complicate the processing of aggregate operators. For example, assume that some tuples in the *instructor* relation have a null value for *salary*. Consider the following query to total all salary amounts:

select sum (*salary*)
from *instructor*;

The values to be summed in the preceding query include null values, since some tuples have a null value for *salary*. Rather than say that the overall sum is itself *null*, the SQL standard says that the **sum** operator should ignore *null* values in its input.

In general, aggregate functions treat nulls according to the following rule: All aggregate functions except **count (*)** ignore null values in their input collection. As a result of null values being ignored, the collection of values may be empty. The **count** of an empty collection is defined to be 0, and all other aggregate operations return a value of null when applied on an empty collection.

nulls can cause some unexpected behavior with aggregate operations. `COUNT(*)` handles *null* values just like other values, that is, they get counted. All the other aggregate operations (`COUNT`, `SUM`, `AVG`, `MIN`, `MAX`, and variations using `DISTINCT`) simply discard *null* values—thus `SUM` cannot be understood as just the addition of all values in the (multi)set of values that it is applied to; a preliminary step of discarding all *null* values must also be accounted for. As a special case, if one of these operators—other than `COUNT`—is applied to *only* null values, the result is again *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.

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.

Next Topic:

SQL

Null Values in SQL

Note:

So far, We Assumed that there were NO NULL Values in the tables.

Now We will study

"NULL Values in SQL & their effects on everything else"…

Note:

Arithmetic Operations:

$+, -, \div, \times$

Comparison Operations:

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

<> is symbol for "Not Equal"

Note:

1. Arithmetic Operation with NULL ALWAYS produces Null.

2. Comparison Operation with Null ALWAYS produces a new truth value, UNKNOWN.

Truth Values in Boolean logic:

True

False

Truth Values in SQL:

- True (T)
- False (F)
- Unknown (U)

Comparison op^n

Null < 50 = Unknown

Unknown → new Truth Value

Null + 50 = Null

arithmetic op^n

Null × 0 = Null

Null = 0 : Unknown

Comparison

Null ≤ Null : Unknown

0 ÷ Null : Null

Null Vs Unknown

Null
- a missing value
- or
- Not known value
- or
- Not Applicable value
- or
- Not Recorded value

Unknown
- a Truth Value
  - T
  - F
  - Unknown (U)

Null < 0 : Null ✗

Null < 0 : unknown ✓

→ a Truth Value

Null + 50 : unknown ✗

Null + 50 = Null ✓

In WHERE clauses, we must be prepared for the possibility that a component of some tuple we are examining will be NULL. There are two important rules to remember when we operate upon a NULL value.

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.

Ullman

Next Sub-Topic:

SQL

Three Truth values in SQL

True, False, Unknown

SQL's Three-Valued Logic:

SQL uses three-valued logic for evaluating queries on databases with nulls.

Three Truth values in SQL:

T, F, U

Note:

We know that:

Comparisons involving null values have the truth value "unknown".

6.1.7 The Truth-Value UNKNOWN

Comparison with NULL yields a third truth-value: UNKNOWN.

We must now learn how the logical operators behave on combinations of all three truth-values.

Trick to Remember Properties of Unknown :

True ≡ 1

False ≡ 0

Unknown ≡ 1/2

x and y = $\min(x, y)$

x or y = $\max(x, y)$

NOT x = $1 - x$

x	y	x and y	x or y	NOT x
T	T	T	T	F
T	F	F	T	F
T	U	U	T	F

T ≡ 1 ; NOT x = 1 − x ; NOT True = 1 − 1 = 0 ↑ False

T ≡ 1 ; and ≡ min ; T and T = min(1, 1) = 1

T ≡ 1 ; OR = max ; T or T = max(1,1) = 1 = True

T and U = min (1, 1/2) = 1/2 = U

1 min 1/2

T or U = max (1, 1/2) = 1 = True

x	y	x and y	x or y	NOT x
U	T	U	T	U
U	F	F	U	U
U	U	U	U	U

NOT U = 1 − U = 1 − 1/2 = 1/2 = U

U and U = min (1/2, 1/2) = 1/2 = U

x	y	x and y	x or y	NOT x
F	T	F	T	T
f	F	f	f	T
F	U	f	U	T

The rule is easy to remember if we think of TRUE as 1 (i.e., fully true), FALSE as 0 (i.e., not at all true), and UNKNOWN as 1/2 (i.e., somewhere between true and false). Then:

1. The AND of two truth-values is the minimum of those values. That is, x AND y is FALSE if either x or y is FALSE; it is UNKNOWN if neither is FALSE but at least one is UNKNOWN, and it is TRUE only when both x and y are TRUE.

2. The OR of two truth-values is the maximum of those values. That is, x OR y is TRUE if either x or y is TRUE; it is UNKNOWN if neither is TRUE but at least one is UNKNOWN, and it is FALSE only when both are FALSE.

3. The negation of truth-value v is $1 - v$. That is, NOT x has the value TRUE when x is FALSE, the value FALSE when x is TRUE, and the value UNKNOWN when x has value UNKNOWN.

x	y	x AND y	x OR y	NOT x
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

Figure 6.2: Truth table for three-valued logic

Next Sub-Topic:

SQL

(WHERE Clause) Tuple Selection In case of Null Values

Note:

WHERE clause only returns rows whose result is true, not false or UNKNOWN.

WHERE Clause: → Check Tuple by Tuple

- If Tuple selection Condition Evaluates to F/U then that Tuple will be "Rejected".
- ⇒ Tuple selected only if Condition becomes T.

Consider relations R with a single integer attribute A.

Query:

Select A
from R
where A <= 0 or A > 0

Find Number of Tuples in the output ? = 4 ✓

Consider relations R with a single integer attribute A.

Query:

Select A

from R ①

where A <= 0 or A > 0

②

Tuple selection Condition

Applies on Every Tuple individually & independently

(Null ≤ 0 or Null > 0)

or

U

Condition c

R

A	Condition c
0	T
-1	T
1	T
-1	T
Null	U
Null	U

Pass

fail

Example:

Consider relation R with a single numerical attribute A, and assume that R contains a single row with a null value in it.

Then the query

select S.A from S where S.A <= 0 or S.A > 0

returns nothing.

This is because both null <= 0 and null > 0 evaluate to unknown and so does their disjunction.

Example:

```
   x
--------
      30
      40
 (null)

SELECT * FROM inctest WHERE x >= 0;
 x
----
 30
 40

SELECT * FROM inctest WHERE x < 0;
 x
----
```

Empty Table

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." □

Next Sub-Topic:

SQL

Keywords:

IS NULL

IS NOT NULL

Consider relations R with a single integer attribute A.

Assume we want to know how many NULL values are there in the column A.

Is this Query Correct for this purpose??

```
Select count(*)
from R
where A = Null
```

Consider relations R with a single integer attribute A.

Assume we want to know how many NULL values are there in the column A.

Is this Query Correct for this purpose??

```
Select count(*)
from R
where A = Null
```

Invalid SQL Query

Because we can NOT use Null as a operand in the SQL Query.

NOTE:

SQL says
"we cannot use NULL explicitly as an operand."

To check Null Value :

SQL Keywords :

- Is Null
- Is Not Null

Consider relations R with a single integer attribute A.

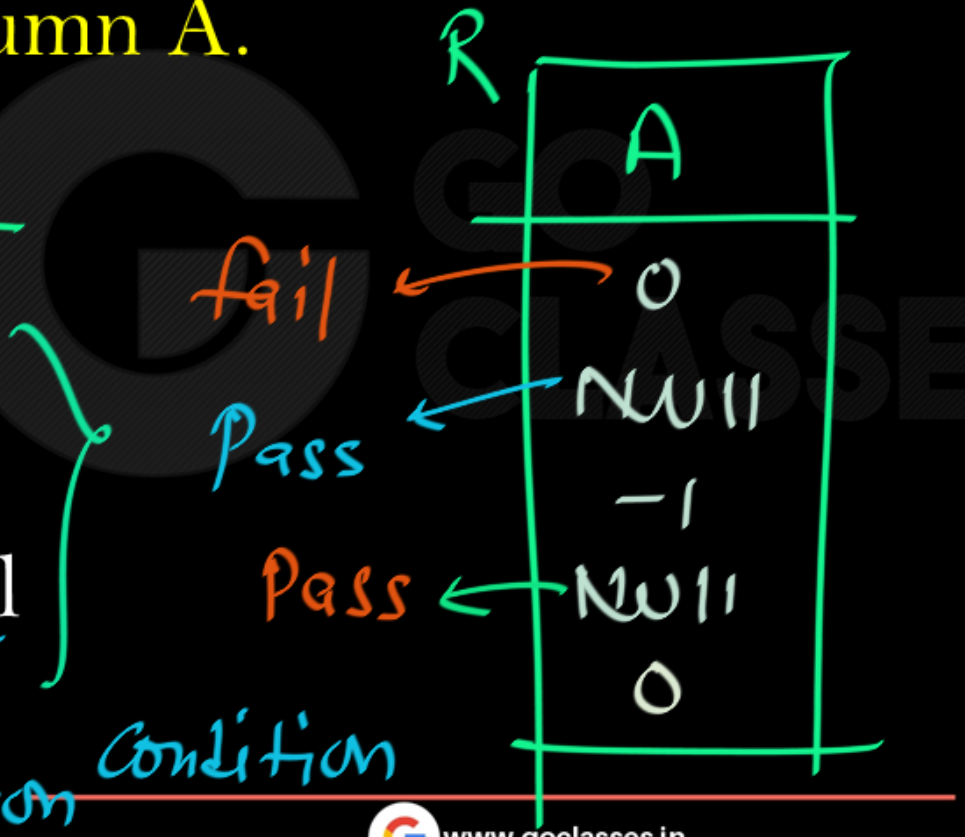
Assume we want to know how many NULL values are there in the column A.

Correct Query:

```
Select count(*)
from R
where A IS Null
```

(1) from R
(2) where A IS Null
(3) Select count(*)

Tuple selection condition



Number of Tuples in o/p = 2

Consider relations R with a single integer attribute A.

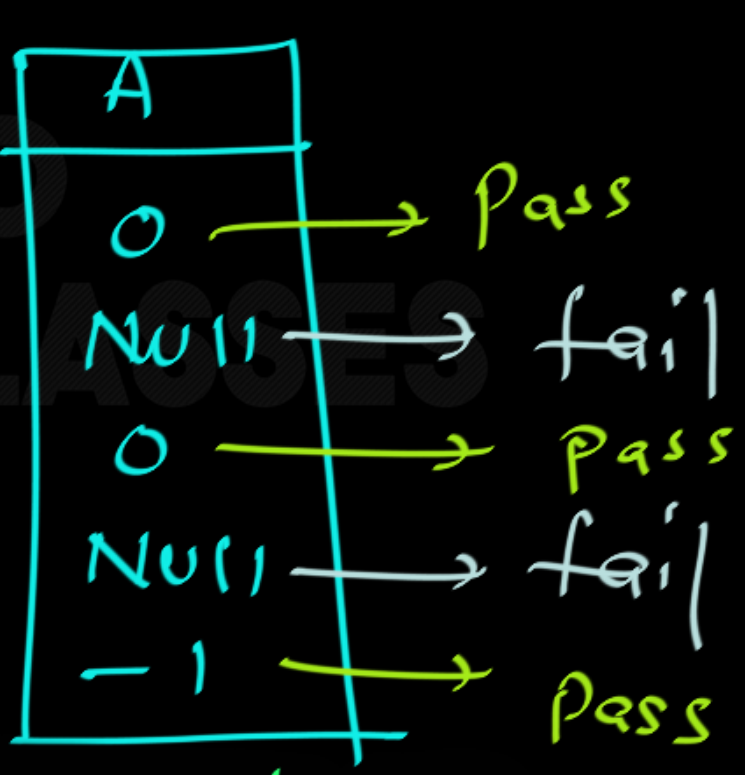
Assume we want to know how many Non-NULL values are there in the column A.

Correct Query:

```
Select count(*)
from R
where A IS NOT Null
```

count(*) → Number of Rows

1 2 3



Number of Tuples in o/p = 3

```
SELECT Count(A)
FROM R
WHERE A Is Null
```

o/p = ?

3

SELECT Count(A)

FROM R (1)

WHERE A Is Null

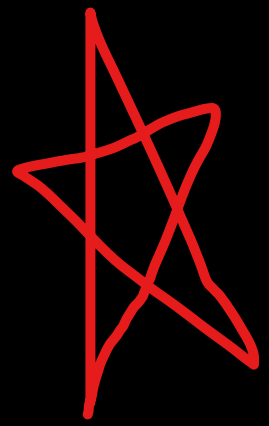
(2)

o/p = ?

Count(A)
0

R

A
Null → Pass
0 → fail
Null → Pass
0 → fail
-1 → fail
NULL → Pass



Number of Tuples in o/p = 1

The correct way to ask if x has the value NULL is with the expression x IS NULL. This expression has the value TRUE if x has the value NULL and it has value FALSE otherwise. Similarly, x IS NOT NULL has the value TRUE unless the value of x is NULL.

Next Sub-Topic:

SQL

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

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

all attributes

for Null values of C, it becomes Unknown.

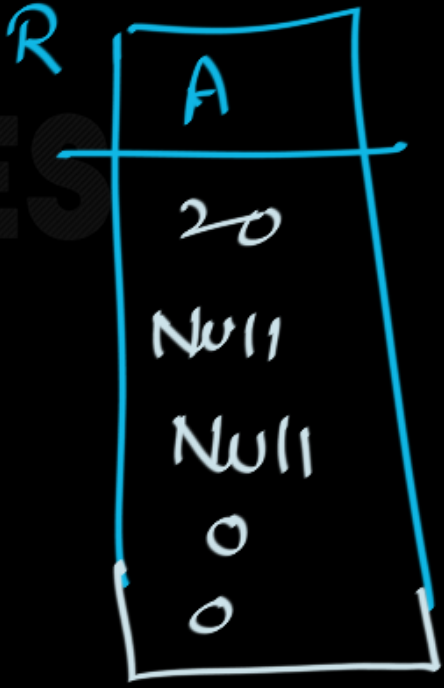
This condition is True for all Non-Null values of C

Q: We know that SQL doesn't allow NULL to be explicitly used as an operand. So, the following query is actually Invalid & illegal.

BUT for this question, Assume that NULL can be used as an Operand. Then how many tuples will be there in the output?

Consider relations R with a single integer attribute A.

```
Select count(*)
from R
where A = Null
```



Q: We know that SQL doesn't allow NULL to be explicitly used as an operand. So, the following query is actually Invalid & illegal.

BUT for this question, Assume that NULL can be used as an Operand. Then how many tuples will be there in the output? ⇒ 0

Consider relations R with a single integer attribute A.

```
Select count(*)
from R
where A = Null
```

(1) (2)

A = NULL → Comparison

for every Tuple it is checked

fail

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(*) : Doesn't ignore Null values
- MAX(value expression)
- SUM(value expression)
- COUNT(value expression) ✓

Which of the following is not an aggregate function ?

Select one:

- a. Min
- b. Avg
- c. Sum
- d. Average ✓

SQL Aggregate functions

Which of the following aggregate functions does not ignore null values?

- AVG
- SUM
- COUNT
- COUNT(*)

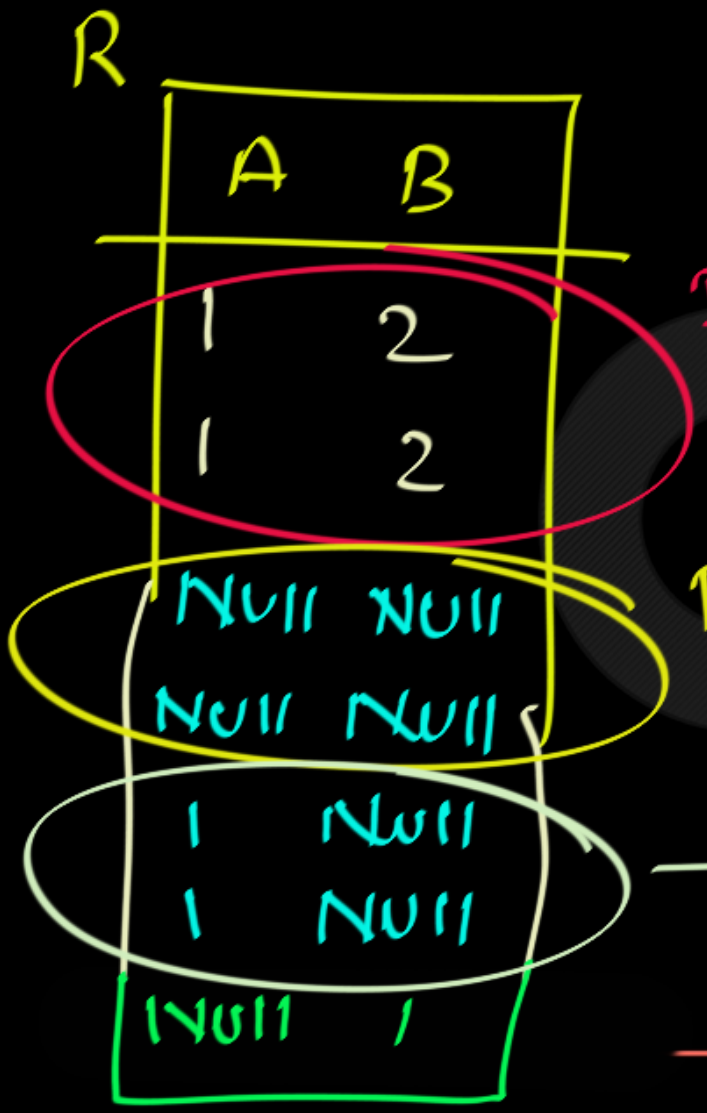
1. Aggregate functions are functions that take a collection of values as input and return a single value. All of them except ( ) ignore null values in their input collection.

A. avg(attr_name) B. count(*) C. max(attr_name) D. min(attr_name)

Next Sub-Topic:

SQL

"Duplicate Tuples" Definition In case of NULL Values



Duplicate Tuples

Duplicate Tuples

→ Duplicate Tuple

Duplicate Tuples :

To Human Eyes, They Look Duplicate.

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.

Korth/ Navath

```
SELECT DISTINCT *
FROM R;
```

⇒ Number of Tuples in o/p = 3

R

A	B
Null	Null
Null	Null
1	2
1	2
1	Null

Boolean expressions arise in many contexts in SQL, and the impact of *null* values must be recognized. For example, the qualification in the `WHERE` clause eliminates rows (in the cross-product of tables named in the `FROM` clause) for which the qualification does not evaluate to `true`. Therefore, in the presence of *null* values, any row that evaluates to `false` or to `unknown` is eliminated. Eliminating rows that evaluate to `unknown` has

As expected, the arithmetic operations $+, -, *$, and / all return *null* if one of their arguments is *null*. However, nulls can cause some unexpected behavior with aggregate operations. `COUNT(*)` handles *null* values just like other values, that is, they get counted. All the other aggregate operations (`COUNT`, `SUM`, `AVG`, `MIN`, `MAX`, and variations using `DISTINCT`) simply discard *null* values—thus `SUM` cannot be understood as just the addition of all values in the (multi)set of values that it is applied to; a preliminary step of discarding all *null* values must also be accounted for. As a special case, if one of these operators—other than `COUNT`—is applied to *only* null values, the result is again *null*.

Query 19. Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.

```
Q19:    SELECT    SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
        FROM      EMPLOYEE;
```

This query returns a *single-row* summary of all the rows in the EMPLOYEE table.

Next Sub-Topic:

SQL

Group By Clause
Having Clause

Next Sub-Topic:

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;

Now, we will study a new Clause "GROUP BY" clause, which is usually used because we want to do aggregation in groups..

Usefulness of Group By Clause:

Group-wise Summary

i.e. Group-wise Aggregation

Consider the relation:
Students(id, name, state, marks)

Find the average marks of all students, state wise i.e. for each state, average marks of all students of it.

Consider the relation:
Students(id, name, state, marks)

Find the average marks of all students, state wise i.e. for each state, average marks of all students of it.

Phrase for Grouping

```
SELECT State, Avg(marks)
FROM Students
GROUP BY State
```

Note: (Guideline)

for which attribute, we need to Groups, How to know?

Ans: for each A ----, ----

So; Group By A ;

Consider the relation: Students(id, name, state, marks)

Find the average marks of all students, state wise i.e. for each state, average marks of all students of it.

aggregate / summary

Need to GROUP(imaginary) the tuples..

State-wise Grouping, & summary for each Group(i.e. state)

Usefulness of Group By Clause:

Group-wise Summary i.e. Group-wise Aggregation

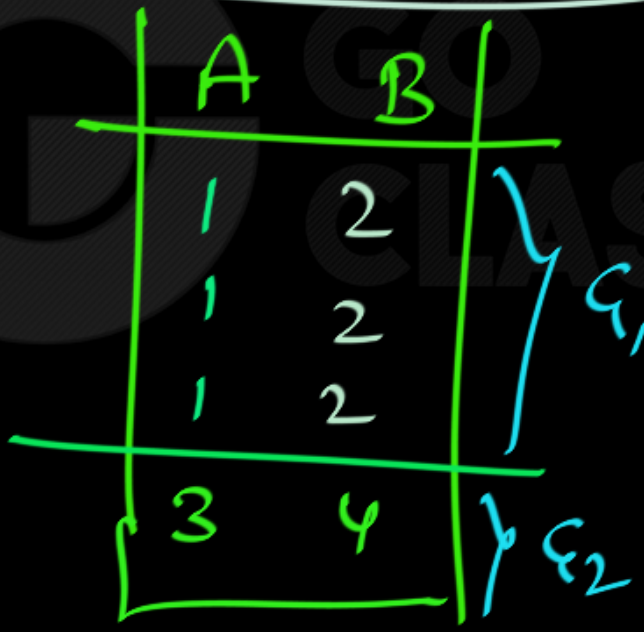
So; for every Group, one Tuple in the output. (when No Having Clause)

Visualization(Imaginary Partition of tuples into Groups)

R(A,B)

GROUP BY A;

A	B
1	2
3	4
1	2
1	2



: Virtual Grouping of Tuples

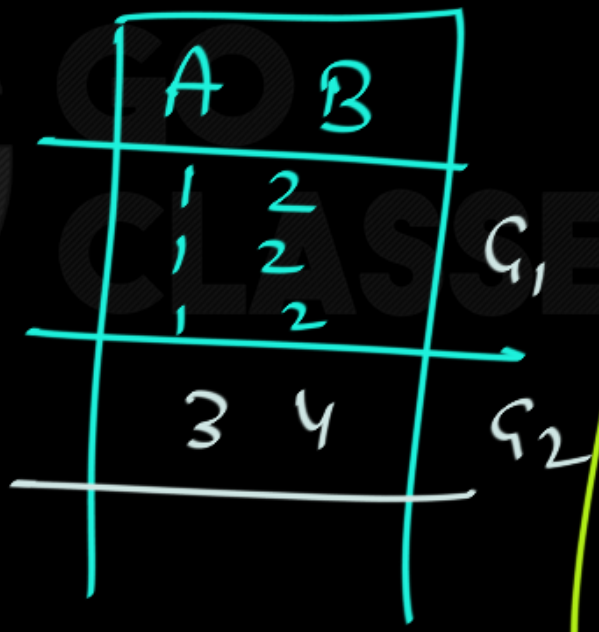
Imaginary Grouping

Visualization(Imaginary Partition of tuples into Groups)

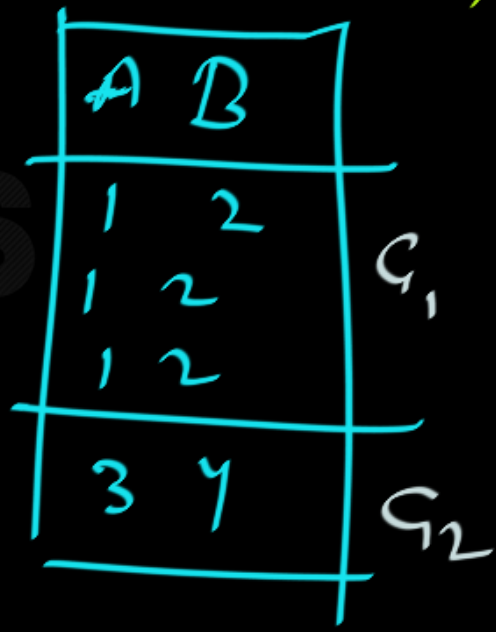
R(A,B)

A	B
1	2
3	4
1	2
1	2

GROUP BY A,B;



Group By B;



	studioName	
	Disney Disney Disney	
	MGM MGM	
	○ ○ ○	

GROUP BY studioName

Figure 5.4: A relation with imaginary division into groups

Visualization(Imaginary Partition of tuples into Groups)

R(A,B,C)

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

GROUP BY C;

How many Groups : 4

Group By A, C ;

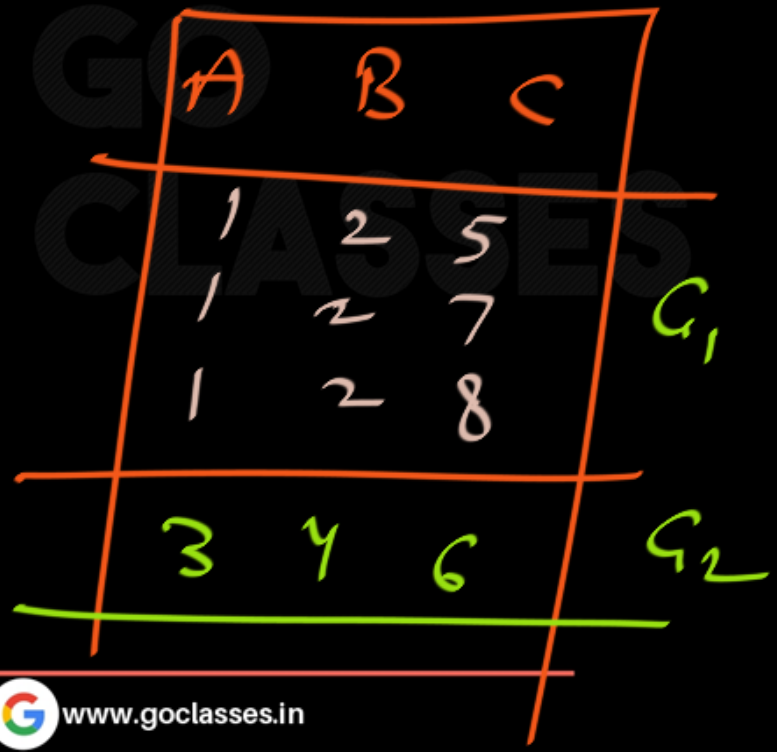
How many Groups : 4

Visualization(Imaginary Partition of tuples into Groups)

R(A,B,C)

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

GROUP BY A,B;



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.

- 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.

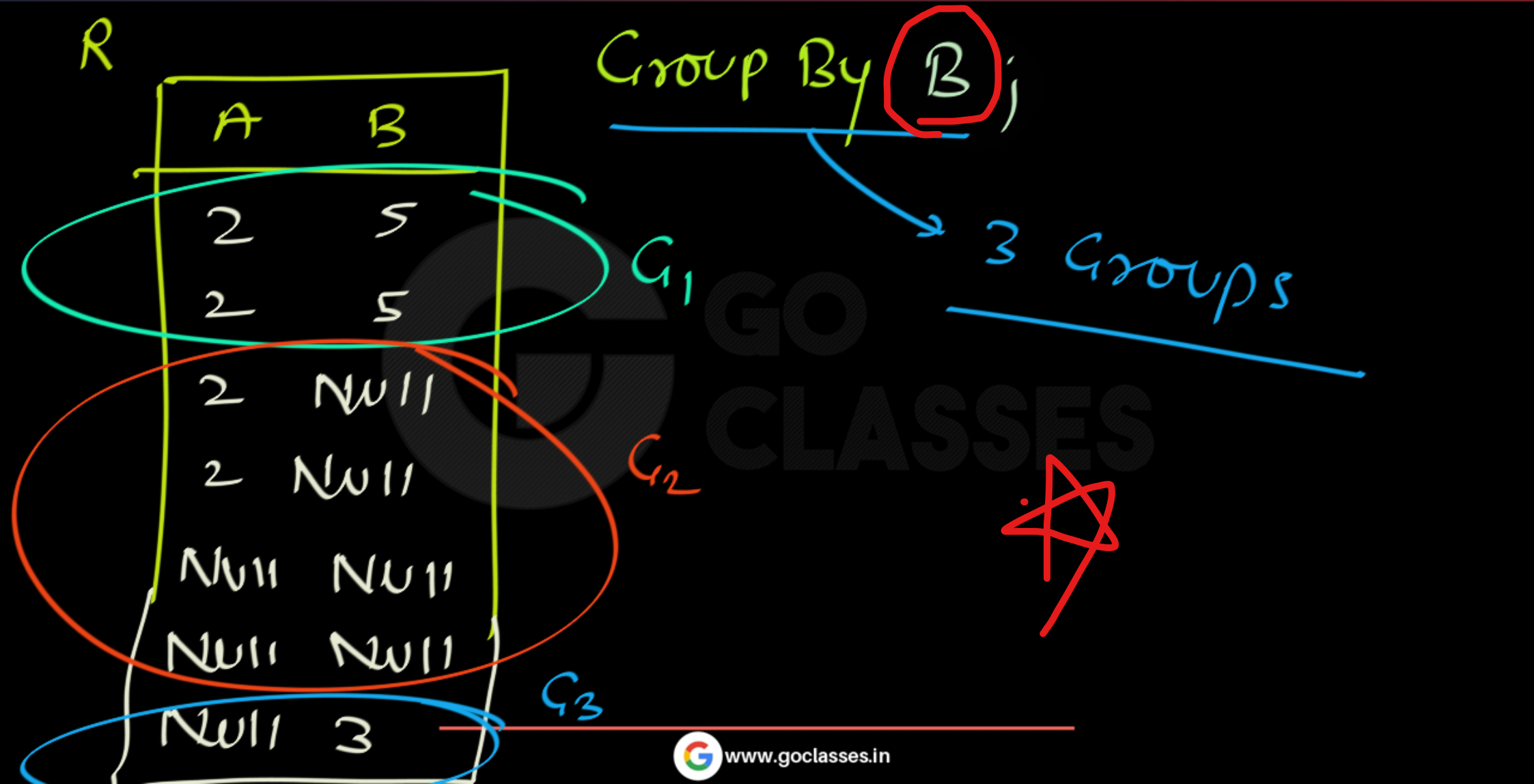
R

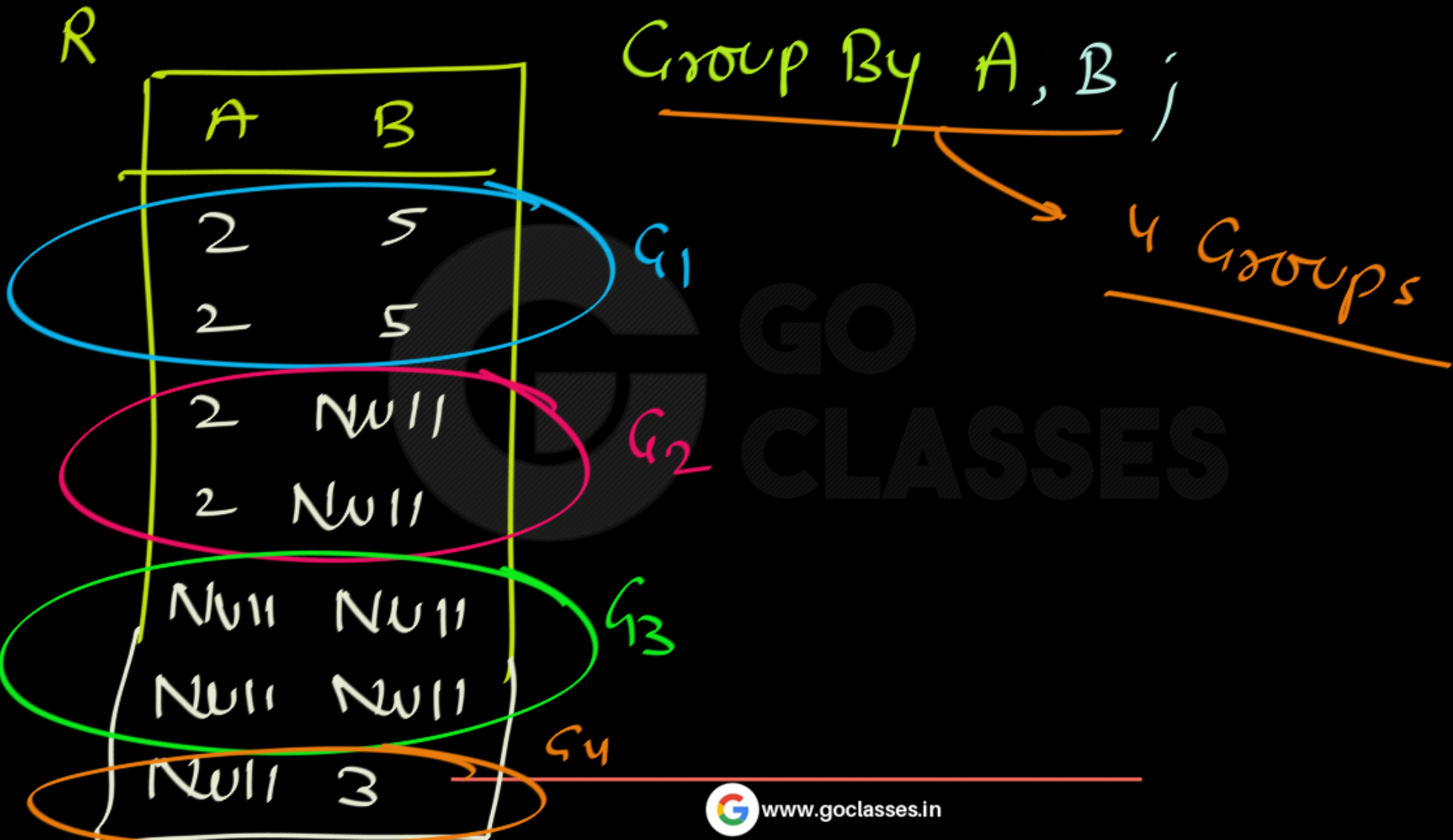
A	B
2	5
2	5
2	Null
2	Null
Null	Null
Null	Null
Null	3

Group By A :

→ G_1

→ G_2


R
A B
2 5
2 5
2 Null
2 Null
Null Null
Null Null
Null 3
Group By B ;
3 Groups
G1
G2
G3


R
A B
2 5
2 5
2 Null
2 Null
Null Null
Null Null
Null 3
G1
G2
G3
G4
Group By A, B ;
4 Groups

SQL Query Template with GROUP BY:

```
SELECT   L, Aggregation
FROM     - - -
WHERE    - - -
GROUP BY Attribute List
```

L

SQL Query Template with GROUP BY:

```
SELECT   A1, A2, A3, sum(A4), Count(A1),
FROM     - - -                  Count(*)
WHERE    - - -
GROUP BY A1, A2, A3
```

SQL Query Template with GROUP BY:

```
SELECT   A1, A2, A3, A4, Avg(A1)
FROM     - - -
WHERE    - - -
GROUP BY A1, A2, A3
```

A4 → Not Allowed in any SQL Standard

SQL Query Template with GROUP BY:

```
SELECT   A1, A2, Avg(A1), Count(A3)
FROM     - - -
WHERE    - - -
GROUP BY A1, A2, A3
```

NOT Allowed in SQL-92 standard.

SQL Query Template with GROUP BY:

users.soe.ucsc.edu/~eaugusti/archive/365-spring13/365-files/lectures/365-lec09-group-by.pdf

365-lec09-group-by.pdf 2 / 2 153%

```
SELECT B1, ..., Bk, AGG1(E1) AS X1, ..., AGGm(Em) AS Xm
FROM R
GROUP BY B1, ..., Bk;
```

SQL Query Template with GROUP BY:

users.soe.ucsc.edu/~eaugusti/archive/365-spring13/365-files/lectures/365-lec09-group-by.pdf

365-lec09-group-by.pdf 2 / 2 153%

```
SELECT B1, ..., Bk, AGG1(E1) AS X1, ..., AGGm(Em) AS Xm
FROM R
GROUP BY B1, ..., Bk;
```

Operational semantics of GROUP BY

```
SELECT ... FROM ... WHERE ... GROUP BY ...;
```

- Compute FROM ($\times$)
- Compute WHERE (σ)
- Compute GROUP BY: group rows according to the values of GROUP BY columns
- Compute SELECT for each group (π)
  - ☞ One output row per group in the final output

Conceptual Evaluation

Template:

```
SELECT  A, B, Sum(A), min(c), Count(*)
FROM    - -
WHERE   - --
GROUP By  A, B
```

Aggregation Happens Group-wise

Rules for GROUP BY clause:

Aggregation ALWAYS happens within Groups.

Aggregation Function in SELECT clause Applies on GROUP Level i.e. within groups only.

If there are groups, then the aggregation is done separately for each group.

Note:

When No Having clause, then for every Group, one tuple in o/p.

So;

Number of Tuples in o/p = Number of Groups

Note: Before creating Groups, Apply WHERE clause.

- output one tuple per group, with grouping attribute and aggregates

R

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

R group by A

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

A, count(*), max(B)

A	count	max
1	3	'c'
2	2	'b'
3	1	'a'

❖ Grouping

SELECT-FROM-WHERE can be followed by **GROUP BY** to:

- partition result relation into groups (according to values of specified attribute)
- summarise (aggregate) some aspects of each group
- output one tuple per group, with grouping attribute and aggregates

R

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

R group by A

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

A, count(*), max(B)

A	count	max
1	3	'c'
2	2	'b'
3	1	'a'

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

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

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

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

→ 3 Groups

No Having Clause

Tuples in o/p = 3

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

```
SELECT A, B, SUM(C), MAX(D)
① FROM  R
② GROUP BY A, B;
```

G_1, G_2, G_3

o/p :

	A	B	sum(C)	max(D)
G_1	a_1	b_1	9	12
G_2	a_2	b_1	8	11
G_3	a_3	b_2	4	10

#Tuples in o/p = 3

Number of tuples in the output??

```
SELECT A, sum(B), Count(A)
FROM R
WHERE B <> 3
GROUP BY A;
```

R

A	B	C
2	3	Null
2	3	Null
Null	Null	Null
2	3	4
2	4	3
3	4	2
3	4	1

Number of tuples in the output?? = 2

```
SELECT A, sum(B), Count(A)
FROM R
WHERE B <> 3
GROUP BY A;
```

(1) (2) (3)

→ 2 Groups

<> : NOT Equal

→ Comparison

o/p:

A	SUM(B)	Count(A)
2	4	1
3	8	2

A	B	C	B <> 3
2	3	Null	F
2	3	Null	F
Null	Null	Null	U
2	3	4	F
2	4	3	T
3	4	2	T
3	4	1	T

failed

Pass

Number of tuples in the output??

```
SELECT  C , Count(*)
FROM R
WHERE   C < 3 or C ≥ 3
GROUP BY C ;
```

R

A	B	C
~~2~~	~~3~~	~~Null~~
~~2~~	~~3~~	~~Null~~
~~Null~~	~~Null~~	~~Null~~
2	3	4
2	4	3
3	4	2
3	4	1

Number of tuples in the output?? ⇒ 4

```
SELECT  C , Count(*)
FROM R        (1)
WHERE   (2)   C < 3 or C ≥ 3
GROUP BY C ;
```

(3) → 4 groups

R

A	B	C	
2	3	Null	fail
2	3	Null	
Null	Null	Null	
2	3	4	Pass
2	4	3	
3	4	2	
3	4	1	

o/p:

C	Count(*)
4	1
3	1
2	1
1	1

Number of tuples in the output??

```
SELECT A, C, Avg(C)
FROM R

GROUP BY A, C;
```

R

A	B	C
2	3	Null
2	3	Null
Null	Null	Null
2	3	4
2	4	3
3	4	2
3	4	1

Number of tuples in the output?? = 6

```
SELECT A, C, Avg(C)
FROM R
GROUP BY A, C;
```

→ 6 groups

R

	A	B	C
G_1	2	3	Null
	2	3	Null
G_2	Null	Null	Null
G_3	2	3	4
G_4	2	4	3
G_5	3	4	2
G_6	3	4	1

O/P:

	A	C	Avg(C)
G_1	2	Null	Null
G_2	Null	Null	Null
G_3			

Example 6.33: Suppose we have a relation $R(A, B)$ with one tuple, both of whose components are NULL:

A	B
NULL	NULL

Then the result of:

```
SELECT A, COUNT(B)
FROM R
GROUP BY A;
```

Example 6.33: Suppose we have a relation $R(A, B)$ with one tuple, both of whose components are NULL:

A	B
NULL	NULL

Then the result of:

```
SELECT A, COUNT(B)
FROM R
GROUP BY A;
```

is the one tuple (NULL, 0). The reason is that when we group by A, we find only a group for value NULL. This group has one tuple, and its B-value is NULL. We thus count the bag of values {NULL}. Since the count of a bag of values does not count the NULL's, this count is 0.

Example 6.33: Suppose we have a relation $R(A, B)$ with one tuple, both of whose components are NULL:

A	B
NULL	NULL

Then the result of:

```
SELECT A, Sum(B), Count(*), Count(A), min(B)
FROM R
GROUP BY A;
```

Example 6.33: Suppose we have a relation $R(A, B)$ with one tuple, both of whose components are NULL:

A	B
NULL	NULL

Group G_1

Then the result of:

```
SELECT A, Sum(B), Count(*), Count(A), min(B)
FROM R
GROUP BY A;
```

1 Group

o/p:

A	sum(B)	Count(*)	Count(A)	min(B)
Null	Null	1	0	Null

Example 6.33: Suppose we have a relation $R(A, B)$ with one tuple, both of whose components are NULL:

A	B
NULL	NULL

On the other hand, the result of:

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

is the one tuple (NULL, NULL). The reason is as follows. The group for value NULL has one tuple, the only tuple in R. However, when we try to sum the B-values for this group, we only find NULL, and NULL does not contribute to a sum. Thus, we are summing an empty bag of values, and this sum is defined to be NULL. □

Suppose R(A,B) is a relation with a single tuple (NULL, NULL).

```
SELECT  A,  COUNT(B)
FROM    R
GROUP   BY  A;

SELECT  A,  COUNT(*)
FROM    R
GROUP   BY  A;

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

Suppose R(A,B) is a relation with a single tuple (NULL, NULL).

```
SELECT   A,   COUNT(B)
FROM     R
GROUP    BY   A;
```

A	Count(B)
Null	0

R

A	B
Null	Null

```
SELECT   A,   COUNT(*)
FROM     R
GROUP    BY   A;
```

A	Count(*)
Null	1

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

A	Sum(B)
Null	Null

Example of computing GROUP BY

```
SELECT age, AVG(pop) FROM User GROUP BY age;
```

User

uid	name	age	pop
142	Bart	10	0.9
857	Lisa	8	0.7
123	Milhouse	10	0.2
456	Ralph	8	0.3

Compute GROUP BY: group rows according to the values of GROUP BY columns

uid	name	age	pop
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
456	Ralph	8	0.3

Compute SELECT for each group

age	avg_pop
10	0.55
8	0.50

NOTE:

GROUP BY, without Aggregation, is Valid, But Useless…

GROUP BY, without Aggregation, is equivalent to SELECT DISTINCT.

```
SELECT A
FROM R
WHERE A<>3
GROUP BY A;
```

R

A	B
2	1
2	3
2	Null
Null	2
Null	Null
Null	Null
3	4
Null	3

```
SELECT A
FROM R (1)
WHERE A<>3 (2)
GROUP BY A;
```

R

A	B
2	1
2	3
2	Null
Null	2
Null	Null
Null	Null
3	4
Null	3

G_1

failed

o/p:

A
2

```
SELECT A
FROM R
WHERE A<>3
GROUP BY A;
```

R

A	B
2	1
2	3
2	Null
Null	2
Null	Null
Null	Null
3	4
Null	3

o/p

A
2

```
SELECT DISTINCT A
FROM R
WHERE A<>3;
```

o/p

A
2

SELECT A
FROM R
GROUP BY A

SELECT DISTINCT A
FROM R

R

A	B
2	1
2	3
2	Null
Null	2
Null	Null
Null	Null
3	4
Null	3

o/p

A
2
Null
3

o/p

A
2
Null
3

Definition of Duplicate Tuples :

A	B
Null	2
Null	2

Duplicate

Null	Null
Null	Null

Duplicate

"GROUP BY" clause is usually used because we want to do aggregation in groups..

NOTE that because of our wish to "AGGREGATE" within groups of tuples… Group By is useful & was introduced…

Apart from aggregation, you can use Group by But it is almost useless with aggregation…

NOTE: Group By without Aggregation

So far, we have seen Group By is used with Aggregation in Select Clause… What happens for the following query?

Relation R(A,B) & the following query:

```
SELECT A
FROM R
GROUP BY A;
```

NOTE: Group By without Aggregation

So far, we have seen Group By is used with Aggregation in Select Clause…

That's how Group By is USED & is USEFUL..

We can apply Group By without aggregation in select clause, BUT that would be useless & same as SELECT DISTINCT ..

While queries involving GROUP BY generally have both grouping attributes and aggregations in the SELECT clause, it is technically not necessary to have both. For example, we could write

```
SELECT studioName
FROM Movies
GROUP BY studioName;
```

This query would group the tuples of Movies according to their studio name and then print the studio name for each group, no matter how many tuples there are with a given studio name. Thus, the above query has the same effect as

```
SELECT DISTINCT studioName
FROM Movies;
```

SQL Query Template with GROUP BY:

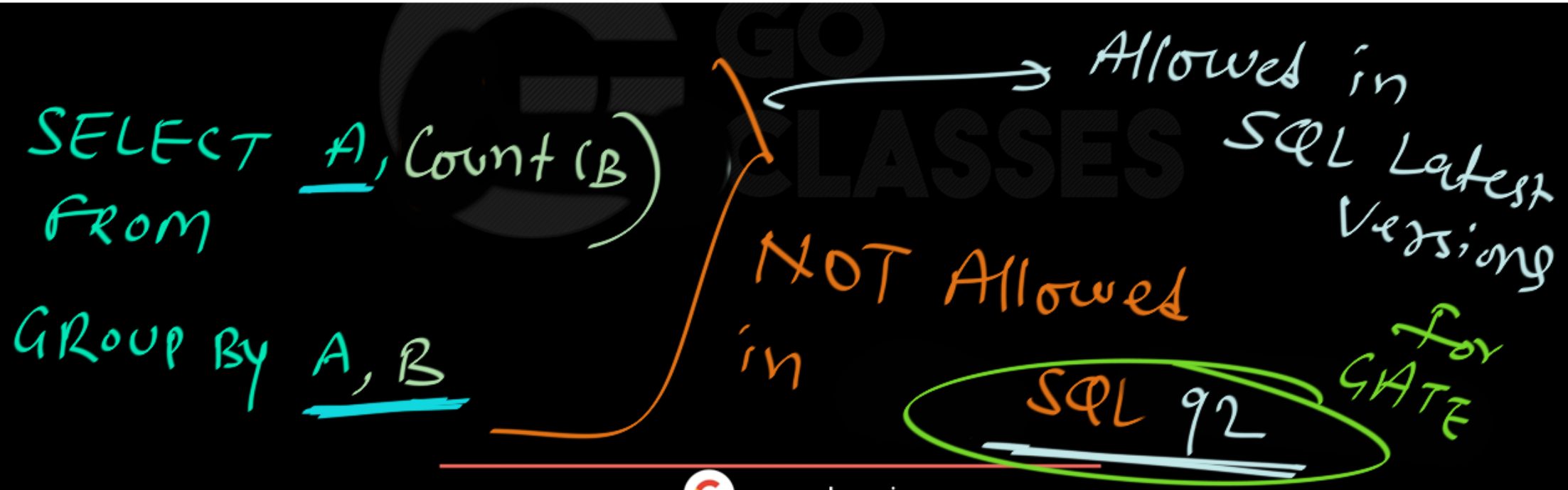
```
SELECT b1, ..., bk, AGG1(e1) AS x1, ..., AGGm(em) AS xm
FROM R
GROUP BY b1, ..., bk
```

SELECT A, B, Count(*)
FROM
GROUP BY A

NOT Allowed in any SQL Version

SQL Query Template with GROUP BY:

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



MUST Rules for GROUP BY clause:

SELCET Clause Restrictions in presence of GROUP BY Clause:

Every "unaggregated" attribute of SELECT Clause MUST appear in the Group By clause.

In Select Clause, ANY attribute(s) of the FROM clause can be Aggregated.

**SQL-92 Rules for GROUP BY clause:

SELECT Clause Restrictions in presence of GROUP BY Clause:

The GROUP BY clause specifies the grouping attributes, which should also appear in the SELECT clause.

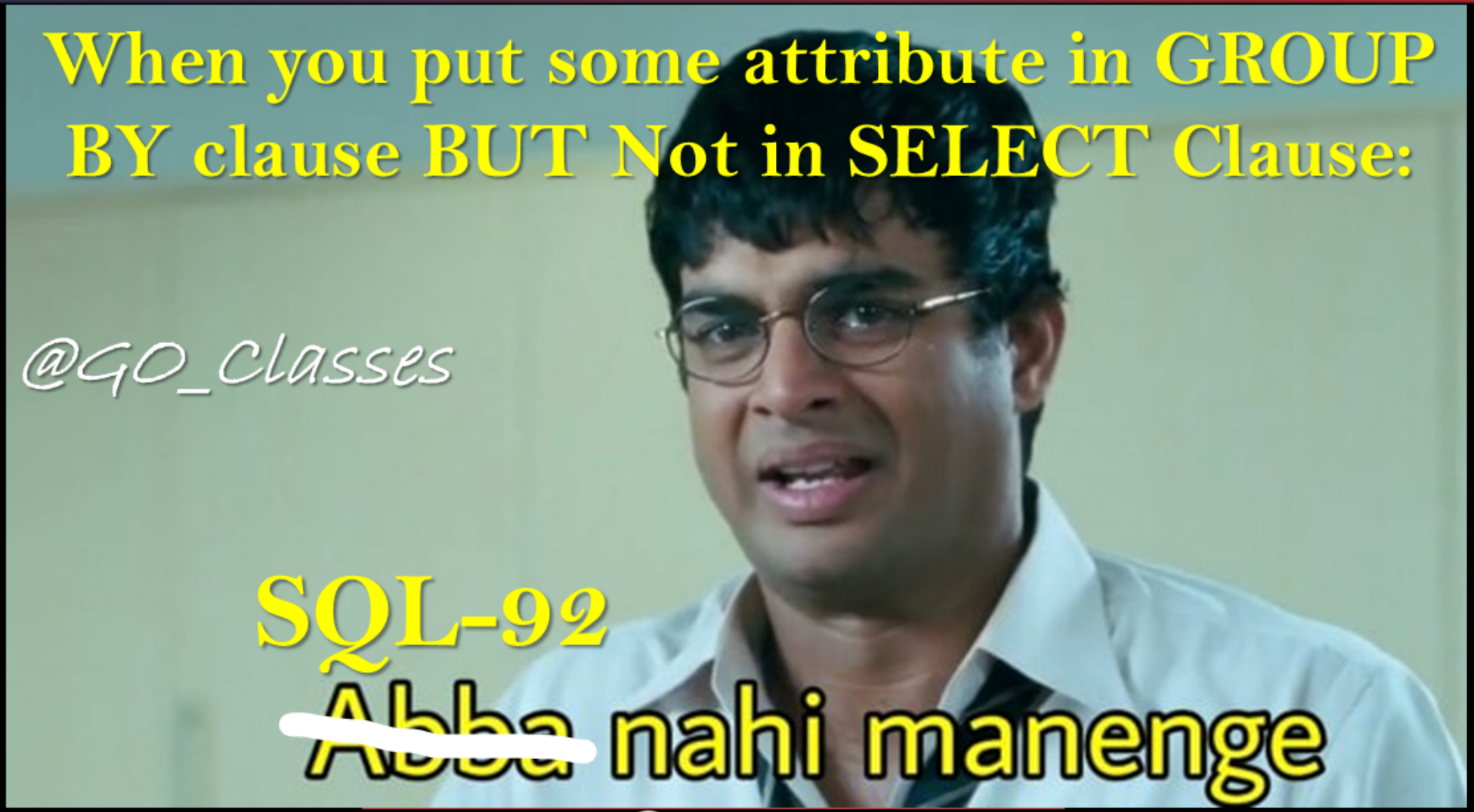
i.e. ALL Group By Clause attributes must appear in SELECT Clause unaggregated.

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.


When you put some attribute in GROUP BY clause BUT Not in SELECT Clause:
@GO_Classes
SQL-92
Abba nahi manenge

NOTE:

For GATE or other exams, **Follow SQL-92 Standard.**

Grouping

- `SELECT … FROM … WHERE …`
  `GROUP BY` *`list_of_columns`*`;`

- Example: compute average popularity for each age group
  - `SELECT age, AVG(pop)`
    `FROM User`
    `GROUP BY age;`

Semantics of GROUP BY

```
SELECT … FROM … WHERE … GROUP BY …;
```

- Compute `FROM` ($\times$)
- Compute `WHERE` (σ)
- Compute `GROUP BY`: group rows according to the values of `GROUP BY` columns
- Compute `SELECT` for each group (π)
  - For aggregation functions with `DISTINCT` inputs, first eliminate duplicates within the group

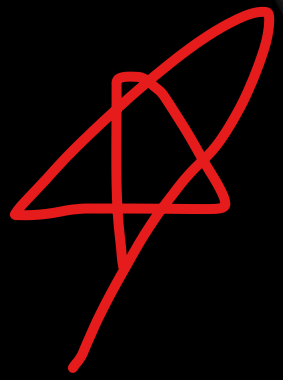
☞ Number of groups = number of rows in the final output

① when No HAVING clause

② After WHERE

NOTE: If there is NO Group By Clause:

If the GROUP BY clause is omitted, the entire table is regarded as a single group.



```
SELECT  SUM(A)
FROM    R;
```

⇒ One Tuple in o/p

```
SELECT SUM(A), B
FROM R;
```

⇒ Invalid

If there is NO Group By Clause:

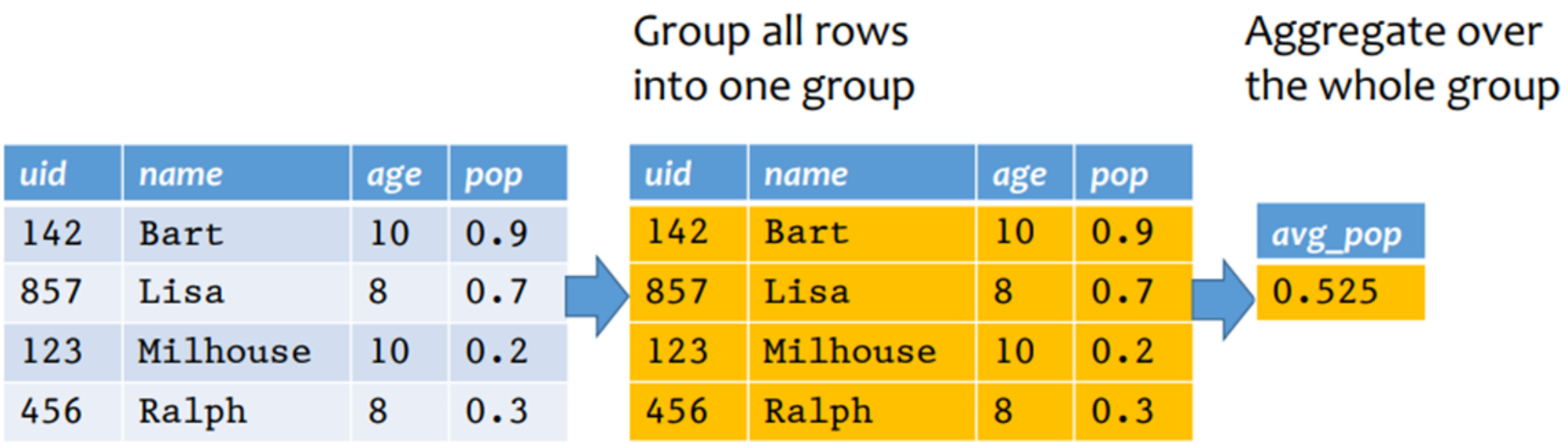
If the GROUP BY clause is omitted, the entire table is regarded as a single group.

That's why, Aggregation without Group By Clause gives SINGLE Tuple in the Output.

Aggregates with no GROUP BY

- An aggregate query with no GROUP BY clause = all rows go into one group

```
SELECT AVG(pop) FROM User;
```



Restriction on SELECT

- If a query uses aggregation/group by, then every column referenced in SELECT must be either
  - Aggregated, or
  - A GROUP BY column

Examples of invalid queries

- WRONG!
```
SELECT uid, age
FROM User GROUP BY age;
```
  - Recall there is one output row per group
  - There can be multiple *uid* values per group
- WRONG!
```
SELECT uid, MAX(pop) FROM User;
```
  - Recall there is only one group for an aggregate query with no GROUP BY clause

Next Sub-Topic:

SQL

Group By Clause

Having Clause

SQL

Complete Summary

To be Continued...