Date .....

Expt. No. .....

Page No.....

Postgre SQL - Stores data and reads queries and returns information

Pg Admin - Graphical User Interface for connecting with Postgre SQL

Select statement -

Syntax -

SELECT column-name FROM table-name

Eg -

Table-1

c1	c2	c3
x	23	a
y	18	b
z	46	c

SELECT c1 FROM Table-1

SELECT * FROM Table-1 → displays all the columns from table.

Expt. No. .....

Date .....

Page No.....

Distinct Keyword -

Finds the distinct elements from column

Eg -

Table

Name	Colour-liked
Jack	Green
Jones	Blue
Amit	Green
Animesh	Blue

Syntax - SELECT DISTINCT Colour-liked FROM Table

Output -

Colour-liked
Green
Blue

Count function -

- COUNT returns the number of input rows that returns a specific condition of query.
- We can apply COUNT on specific column or pass COUNT(*)

Eg -

Table

Name	Colour-liked
Jack	Green
Jones	Blue
Amit	Green
Animesh	Blue

Syntax SELECT COUNT (Colour-liked) FROM Table

OR

SELECT COUNT (Name) FROM Table

OR

SELECT COUNT(*) FROM Table

Output -

Count
4

#Since, there are 4 rows irrespective of no. of columns the count function returns 4 as output

Count function + Distinct

It gives out the count of distinct elements present in the column.

Eg - Table

Name	Color_liked
Jack	Green
Jones	Blue
Amit	Green
Animesh	Blue

Syntax - SELECT COUNT (DISTINCT (Name)) FROM Table

Date .....

Expt. No. ..... Page No.....

Output →

Count
4

Here, all the names are distinct, hence count of distinct elements shown is 4.

SELECT COUNT (DISTINCT (Colour - Liked)) FROM Table.

Output →

Count
2

Here, there are only two unique colors, i.e., Green and Blue hence, the count of unique colours = 2.

Where statement -

Allows to specify conditions on columns for rows to be returned

- WHERE clause appears immediately after FROM clause of SELECT statement.

- The conditions are used to filter the rows returned from the SELECT statement.

* WHERE can be applied to different Comparison Operators such as =, >, <, >=, <=, !=

Logical operators such as AND, OR, NOT

Table

Eg -

Name	Choice
Zach	Green
David	Green
Claire	Yellow
David	Red

Syntax - SELECT Name, Choice FROM Table WHERE

Name = "David" ;

Output →

Name	Choice
David	Green
David	Red

SELECT Name, Choice FROM Table WHERE name = "David"

Choice = "Red" ;

Output →

Name	Choice
David	Red

* Note → Use single ' ' when giving strings instead of " "

Date .....

Expt. No. ..... Page No.....

Order By -

We can use ORDER BY to sort rows in either ascending or descending order

* We can use ORDER BY on multiple columns.
This makes sense when one column has duplicate entries

Eg -

Company	Name	Sales
Apple	Andrew	100
Groogle	David	500
Appll	Zach	300
Groogle	Claire	200
Xerox	Steven	100

Table

Syntax - SELECT Company, Name, Sales FROM Table
ORDER BY Company, Sales.

Company	Name	Sales
Apple	Andrew	100
Apple	Zach	300
Groogle	Claire	200
Groogle	David	500
Xerox	Steven	100

→ First Company is filtered in ascending order

⊙ Then Sales is filtered in ascending order corresponding to company

#When ORDER BY is performed whole row shifts from one position to other

By default, ORDER BY sorts in ascending order.

Syntax - SELECT * FROM Table
ORDER BY company, sales ASC/DESC;

Limit

- LIMIT command allows us to limit the number of rows returned for a query.
- LIMIT goes at the end of a query request and is last command to be executed.

Eg-

Payment

name	amount	payment_date
David	500	2006
Sam	1000	2007
Akshat	500	2002
Henry	200	2012

```
SELECT * FROM Payment
ORDER_BY payment_date DESC
LIMIT 2;
```

↓

No. of rows to be returned by LIMIT

Date .....

Expt. No. ..... Page No.....

Output –

Payment

Name	amount	payment_date
Henry	200	2012
Sam	1000	2007

Here, no. of rows returned = 2

Ordering of table is done in descending order ~~order~~ for payment_date.

Between

It is same as value >= low AND value <= high

value BETWEEN low AND high

Syntax – date BETWEEN '2007-01-01' AND '2007-02-01';

In()

We use IN() to make a condition that checks to see if a value ~~is~~ in included in a list of multiple options.

Syntax – SELECT color from Table
WHERE color IN ('red', 'blue', 'green')

⇒ returns color column where colour is either red or blue or green

We can also combine NOT with IN()

Eg - SELECT color FROM table
WHERE color NOT IN ('red', 'blue')

We have already been able to perform direct comparison against strings such as -

WHERE first_name = 'John'

But if we want to match against a general pattern in a string like -

All emails ending in '@gmail.com'

All names that begin with an 'A'.

Like

The LIKE operator allows us to perform pattern matching against a string data with the use of wildcard characters.

- Percent %

Matches any sequence of characters

- Underscore

Matches any single character

Date .....

Expt. No. .....

Page No.....

Syntax -

To find all names that begins with an 'A'

```
WHERE name LIKE 'A%'
```

To find all names that ends with an 'A'

```
WHERE name LIKE '%A'
```

LIKE is case sensitive.

When we want to remove case sensitive barrier of LIKE, we can use ILIKE as it is case insensitive.

Examples -

- To find someone's name has a particular string in his/her name

```
SELECT first_name FROM customer
WHERE first_name LIKE '%er%';
```

Output -

first_name
Jennifer
Heather
Cheryl
Katherine

To find a particular string in name with specified spaces.

```
SELECT first_name FROM customer
WHERE first_name LIKE '_hr%';
```

Output -

first_name
Cheryl
Theresa
Sherry
Sherri

Note - The first character is ignored because of the _ and then the pattern of character is searched.

NOT + LIKE

```
SELECT first_name FROM customer
WHERE first_name NOT LIKE '_hr%';
```

Output -

first_name
Jared
Mary
Linda
Barbara

Date .....

Expt. No. ..... Page No.....

Aggregate functions -

The main idea of Aggregate functions is to take multiple inputs and return a single output.

Common Aggregate Funtions -

AVG() - returns average value

COUNT() - returns number of values

MAX() - returns maximum value

MIN() - returns minimum value

SUM() - returns the sum of all values.

Aggregate Funtions are used just after SELECT or HAVING clause.

AVG() returns a floating point value (eg - 2.34861...)

Eg- SELECT AVG (replacement_cost)
FROM film;

Output -

avg
19.984000000000

* Round() -

SELECT ROUND (AVG (replacement_cost), 2)
FROM film;

Output -

avg
19.98

rounds the floating value to 2 decimal places.

Date .....

Expt. No. .....

Page No.....

Group By

We need to choose a categorical column to GROUP BY

Categorical columns are non-contiguous

Eg- Class 1, Class 2, Class 3, etc.

Eg-

Category	Data Value
A	10
A	5
B	2
B	4
C	12
C	6

A	10
A	5

B	2
B	4

C	12
C	6

Aggregate SUM Function

Category	Result
A	15
B	6
C	18

Aggregate AVG Function

Category	Result
A	7.5
B	3
C	9

GROUP BY clause must appear right after FROM or WHERE statement.

(x) In order to use a GROUP BY,
we need to select the column in which we wish to call the GROUP BY.

Eg- SELECT category, Abbr (data)
FROM table
GROUP BY category;

(*) Here, both of GROUP BY and SELECT has category.

VIM When performing GROUP BY on a particular column, we also use the aggregate function on one of the column.

Eg - SELECT (company, division) SUM (sales)
FROM frame-table
GROUP BY (company, division.

Here, notice that the aggregate function SUM is applied in the sales column.

GROUP BY should never include sales ~~in~~ as its component. Otherwise it will ~~does~~ throws an error.

Date .....

Expt. No. ..... Page No.....

If you want to sort results based on aggregate function, make sure to reference the entire

Eg - ~~SELECT finance - table~~

```
SELECT company, SUM(sales)
FROM finance - table
GROUP BY company
ORDER BY SUM(sales);
```

Here, instead of using ORDER BY only on sales, we use ORDER BY on SUM(sales).

Eg - Shop

Name	Items	Amount
Akshat	Apple	150
Akshat	Egg	60
Awinash	Banana	70
Ankit	Milk	30
Ankit	Orange	70

Syntax - SELECT Name FROM Shop
GROUP BY Name

Output →

Name
Akshat
Awnesh
Ankit

→ Shows output identical to

~~SELECT DISTINCT Name FROM~~
Shop;

```
SELECT name, SUM (amount)
FROM Shop
GROUP BY Name
ORDER BY SUM (amount)
```

Output -

Name	SUM
Akshat	210
Ankit	100
Avinash	70

Having -

A HAVING clause allows us to filter after an aggregation has already taken place.

Eg - SELECT company, SUM (sales)
FROM finance - table
WHERE company != 'Google'
GROUP BY company

If we want to filter based on SUM (sales), having clause will be used.

Date .....

Expt. No. ..... Page No.....

In the above example, we have filtered company column, i.e., google will never be included in the output column of company.

But if we want to filter the aggregate function column such as SUM() having clause can be used.

HAVING clause can be used only after GROUP BY statement is already executed.

As HAVING will filter the result of SUM(sales) column after it is concatinated with company column.

Eg- To get all the company having SUM of sales > 1000

```
SELECT company, (SUM|sales)
FROM finame - table
GROUP - BY company
HAVING (SUM|sales) > 1000
```

Eg- SELECT Name, SUM(Amount)
FROM Shop
GROUP BY Name
HAVING SUM(Amount) >= 100 ;

Output

Name	SUM
Akshat	210
Ankit	100