~~sql~~ \sql
connect _ root@ localhost

DATE

Part :- 3 SQL

- Shreyash Jagtap
- 2020BTEMEO675
- TYmech

* Cheat sheet of SQL :

* SQL SELECT STATEMENT *

① SELECT * FROM tbl
select all rows & colourms from table tbl.

② SELECT C_1, C_2 FROM tbl
select column c1, C2 and all rows from table tbl.

③ SELECT c_1, c_2 FROM tbl WHERE conditions ORDER BY c1 ASC, C_2 DESC
Select coloumn C_1, C_2 with where conditions and from table tbl order result by column c1 in ascending order and c2 in decending order.

④ SELECT DISTINCT c_1, c_2 FROM tbl
Select distinct rows by coloumn c1 & C_2 from table tbl.

⑤ SELECT c1, aggregate (expr) FROM tbl GROUP BY c1
select coloumn c1 and use aggregate function on expression expr, group coloums by c1.

PAGE

DATE

⑥ SELECT C_1, aggregate(expr) As C_2 FROM tbl GROUP By C_1 HAVING $C_2 > V$

Select column C_1 & C_2 as column alias of the result of aggregate function on expr. Filter group of records with C_2 greater than value V.

* SQL UPDATE TABLE

① INSERT INTO tbl (C_1, C_2 - -) VALUES (V_1, V_2 - -)

Insert data into table tbl.

② INSERT INTO tbl (C_1, C_2 - -) SELECT C_1, C_2 - - FROM tbl2 WHERH condⁿ.

Insert data from tbl2 into tbl.

③ UPDATE t. SET $C_1 = V_1$, $C_2 = V_2$ - - - WHERE condⁿ.

Update data in table tbl

④ DELETE FROM tbl WHERE condⁿ.

Delete records from table tbl basedon WHERE condⁿ.

⑤ TRUNCATE TABLE tbl

Drop table tbl and recreate it, all data is lost.

DATE

* SQL TABLE STATEMETS *

① CREATE TABLE tbl (
    C1 datatype (length)
    C2 datatype (length)

    PRIMARY KEY (C1)
)

Create table tbl with primary key is C1

② DROP TABLE tbl
Remove table tbl from database

③ ALTER TABLE tbl
ADD COLUMN C1 datatype (length)
Add column C1 to table tbl

④

DATE

- SELECT Statement →
  - example syntax for SELECT statement.

    SELECT column_name FROM table_name

e.g. 1. SELECT C_1, C_3 FROM table_1

2. SELECT * FROM table_1

↓

due to this all colums will be selected

- In general it is not good practice to use an asterisk (*) in the SELECT statement if you don't really need colums.
- It will automatically query everything which increase the trafic betn the database server and the application which can slow down the retrevl of results.

DATE

SELECT DISTINCT

Sometimes a tables contains a column that has duplicated values and you may find yourself in a situation where you only want to list the uniquely, distint values.

The DISTINCT Keyword can be used to return only the distinct values in a column

The DISTINCT Keyword operates on a column. The syntax looks like this:

Syntax:

SELECT DISTINCT column FROM table

, It will work with or without parenthesis . later on when we learn about adding more calls such as COUNT and DISTINCT together, the parenthesis will be necessary.

PAGE

classmate

DATE

- ## COUNT

The COUNT function returns the number of input rows that match the specific condition of a query
We can apply COUNT on a specific column or just pass COUNT (*), we will soon see this should return the same result.

Note: COUNT is mush more useful when combined with other commands, such as DISTINCT.

- ## SELECT WHERE - Part one

- SELECT and WHERE are the most fundametal SQL statements and you will find yourself using them often!
- The WHERE statments allows us to specify conditions on columns for the rows to be returned.

Syntax:

~~Basi~~

```
SELECT column1, column2
FROM table
WHERE conditions;
```

DATE

- The WHERE clause appears immediately after the FROM clause of the SELECT statement.
- The conditions are used to filter the rows returned from the SELECT Statement.
- Postgre SQL provides a variety of standard operators to construct the condn

  - Comparision opreators
  - logical operator AND, OR, NOT

e.g.

```
SELECT name, choise FROM table
WHERE name='David' and choice='Red';
```

DATE

ORDER BY

You can use 'ORDER BY' to sort rows based on a column value, in either ascending or descending order.

Syntax:

```
SELECT column_1, column_2
FROM tables
ORDER BY column_1 ASC/DESC
```

ASC ↑ to sort in ascending order

DESC ↑ to sort in a descending order.

- If you leave it blank ORDER BY uses ASC by default.
- You can also ORDER BY multiple columns

DATE

LIMIT →

The LIMIt command allows us to limit the number of rows returned for a query.

LIMIT also becaure useful in comination with ORDER By.

Note: LIMIT goes at the end of the of a query request and is the last command to be executed.

BETWEEN

· The BETWEEN operator can be used to match a value agaist a range of values;
  · values BETWEEN Low AND high.

· You can also combine BETWEEN with the NOT logical operator.
  · value NOT BETWEEN low AND high

· The 'BETWEEN' operator can also be used with date. Note that you need to format dates in the ISO 8601 Standerd formate, which is YYYY-MM-DD

· date BETWEEN '2007-01-01' AND '2007-02-01'.

PAGE

classmate

DATE

- IN

In ceartain cases you want to check for multiple possible values optains, for e.g, if a users names shown up IN a list of known names.

We can use the IN operator to create a cond^n that checks to see if a value in included in a list of multiple options.

The genral syntax:

value IN (option 1, option 2, --- option_n)

e.g

```
SELECT color FROM table
WHERE color IN ('red', 'blue');
```

we can also use NOT operators

e.g. WHERE color NOT IN ('red', 'blue');

DATE

LIKE and IKIKE

We've already been able to perform direct comparisons against strings such as:

- WHRE first_name = 'John';

But what if we want to match against a general pattern in a string?

- All emails ending in '@gmails.com'.
- All names that begins with an 'A'.

The LIKE operator allows us to perform pattern matching agaist string data with the use of wild card characters's.

- Precent %
  Matches any sequence of characters.

- Under score _
  matches any single characters.

eg
- All names that begin with an 'A'.
  - WHERE name LIKE 'A%'.
- All names that end with an 'a'.
  - WHERE name LIKE '%a'.

Notice that LIKE is case-sensitive, we can use IKIKE which is case-insensitive.

PAGE

classmate

DATE

Section 3 → Group By Statements →

- ~~Group~~ * Introduction to Group By *

- Most common Aggregate function →

- AVG() - returns average value
- COUNT() - returns number of values
- MAX() - returns maximum values
- MIN() - returns minimum values
- SUM() - returns the sum of all values

Note

- Aggregate function calls happen only in the SELECT clause or the HAVING clause.

Special Notes:

- AVG() returns a floating point value many decimal point (eg 2.342418)
- You can use ROUND() to specify precision after the decimal.
- COUNT() simply returns the number of rows, which means by convention we just use COUNT(*).

DATE

Group by - Part-one

- group by allows us to aggregate columns per some category.

We need to choose a categorical column to GroupBy

Categorical columns are non-continous.

General syntax:

```
SELECT category_col, AGG(data_col)
FROM table
GROUP BY category_col
```

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

DATE

e.g.

```
SELECT company, division, SUM(sales)
FROM Finace_table
GROUP BY company, division
ORDER BY SUM(sales)
```

In select statement, columns must either have an aggregate function or be in the GROUP By call.

"Emp"

DATE

eg1

E_id	E_name	Dept	Salary
1	Ram	HR	10000
2	Amrit	MRKT	20000
3	Ravi	HR	30000
4	Nitin	MRKT	40000
5	Varun	IT	50000

Q. write a query to display all the dept name along with no of Emps working in that?

O/P:

HR	2
MRKT	2
IT	1

```
Select dept, Count (*) From Emp
group by dept;
```

Q. write a query to display all the dept names where no. of employes are less than 2?

```
Select E-name from Emp
where dept In (Select dept from Emp group by
                dept
                having count (*) <2);)
```

PAGE

classmate

DATE

① write a query to display highest salery department wise and name of emp who is taking that sallary?

```
Select E.name from emp
where salary
In ( (Select max(salary) from emp group by dept) )
```

note:

```
SELECT DATE(payment_date) FROM payment;
```

↓

just date will dis not the time of that.

DATE

HAVING

The HAVIG caluse allows us to filter after an aggregation has already taken place.
~~let's take a look back at one~~

e.g.
```
SELECT company, Sum(sales)
FROM finance_table
WHERE company != 'Google'
GROUP BY company
HAVING Sum(sales) > 1000
```

HAVING allows us to use the aggregate result as a filter along with a 'GROUP BY'.

note: Aggregate function are not allowed in WHERE. So, we have to use HAVING clause there

DATE

Section 5- Joins

- Introduction of joins →

Joins will allows us to combine information from multiple tables!

- As statement →

As clause which allaus us to create an "alias" for column or result.
i.e it change the name of coloum in the result table

e.g.
```
SELECT SUM(amount) AS net_revenue
FROM payment;
```

O/P

net_revenue
-
-
-

The AS operator gets executed at the very end of a query, meaning that we can not use the ALIAS inside a WHERE operator!

DATE

e.g

```
SELECT COUNT(amount) FROM payment;
```

o/p :

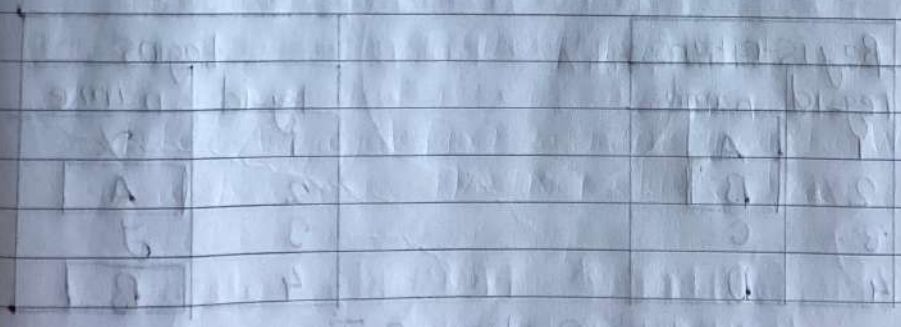
count
14596

```
SELECT COUNT(amount) AS num_transactions
FROM payment;
```

o/p

num_transactions
14596

note: new name of the col not use in where statement or having clause



DATE

- Inner joins →

What is the join operations →

Joins allows us to combine multiple tables together.

The main reason for the different Join types is to decide how to deal with information only present in one of the joined tables.

e.g.
The respective id columns indicate what order they registered or logged in onsite.

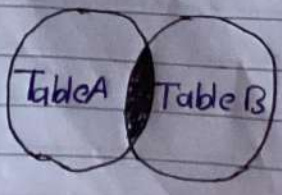
Registrations	
reg_id	name
1	A
2	B
3	C
4	D

Logins	
log_id	name
1	X
2	A
3	Y
4	B

→

simple Syntax of Inner joins →

```
SELECT * FROM ~~TABLE~~ TableA
INNER JOIN TableB
ON TableA.col_match = TableB.col_match
```



classmate

PAGE

DATE

SELECT * FROM Registration
INNER JOINS Logins
ON Registrations.name = Logins.name

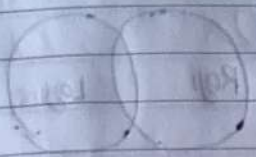
O/P :

reg_id	name	log_id	name
1	A	2	A
2	B	4	B

Note: IF we interchange the position of Registration and Logins then result (O/P) will not change.

Also if you see just JOIN without the INNER, PostgreSQL will treat it as an INNER JOIN.

e.g. Select * From payment
Inner join customer customer
ON Payment.customer_id = customer_id



PAGE

classmate

DATE

FULL OUTER JOINS →

There are Few different types of OUTER JOINS

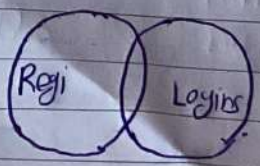
They will allow us to specify how to deal with values only present in one of the table being joined

Simple Syntax →

```
SELECT * FROM TableA
FULL OUTER JOIN TableB
ON TableA.col_match = TableB.col_match
```

e.g.

```
SELECT * FROM Registrations
FULL OUTER JOIN Logins
ON Registration.name = Logins.name
```



DATE

O/P:

reg_id	name	log_id	name
1	A	2	A
2	B	4	B
3	C	null	null
4	D	null	null
null	null	1	X
null	null	3	Y

- Full outer joins with WHERE
  get rows unique to either table
  (rows not found in both tables)

simply syntax:

```
SELECT * FROM TableA
FULL OUTER JOIN TableB
ON Table.colmatch = TableB.col_match
WHERE TableA.id IS null OR
TableB.id IS null
```

O/P: execpt first two lines ---

PAGE

DATE

- Left outer joins →

A LEFT OUTER JOINS results in the set of records that are in the left table, if there is no match with right table, the results are null.

simple syntax:

```
SELECT * FROM TableA
LEFT OUTER JOIN TableB
ON TableA. col_match = TableB. col_match ;
```

note: orders matters for left outer joins.

e.g.
```
SELECT * FROM Registrations
LEFT OUTER JOIN Logins
ON Registrations.name = Logins.name
```

O/P:

reg_id	name	log_id	name
1	A	2	A
2	B	4	B
3	C	null	null
4	D	null	null

DATE

1. What if we only wanted entries unique to Table A? Those rows found in Table A and not found in Table B.

Simple syntax:

```
SELECT * FROM TableA
LEFT OUTER JOINS TableB
ON TableA.COL_match = TableB.col_match
WHERE TableB.id is null
```

```
SELECT * FROM Registrations
LEFT OUTER JOINS Logins
ON Registrations.name = Logins.name
WHERE Logins.log_id IS null.
```

O/P:-

reg_id	name	log_id	name
3	C	null	null
4	D	null	null

PAGE

DATE

• Right Joins →

Simple Syntax :

```
SELECT * FROM TableA
RIGHT JOIN TableB
ON TableA.col_match = TableB.col_match
```

e.g.

classmate

PAGE

DATE

UNION

PAGE

classmate

DATE

continoue.

Advanced SQL commands →

Displaying current time information

Note: These will be more useful when creating our own tables and database, rather than when querying a database.

TIME	Contains only time
DATE	contains only date
TIMESTAMP	contains date and time
TIMESTAMPTZ	contains date, time and timezone

careful considerations should be made when desinging a table and database and choosing a time data type.

Depending on the situation you may or may not need the full level of TIMSTAMPTZ.

Remember, you can always remove historical information but you can't add it!

SHOW TIMEZONE →

Select:
- NOW() →
- TIMEOFDAY() →
- CURRENT_DATE →
- CURRENT_TIME →

CLASSMATE

PAGE

DATE

e.g. UPDATE account
SET last_login = CURRENT_TIMESTAMP
WHERE last_login is NULL;

- Using another table's values (UPDATE join)
UPDATE TableA
SET original_col = TableB.new_col
FROM tableB
WHERE tableA.id = TableB.id

e.g. suppose you have phone no. in student table. So you want to update the phone number in student class Tab
You can perform this update operation by joining the table.

e.g2
UPDATE account_jobl
SET hire_date = account.created_on
FROM account
WHERE account_jobl.userid = account.usedid

- RETURNING key word
↑
It Return the required table after updating.

- DELE
rows fro
. we
other tab
e.g
we c
- ALTER
existing

DATE

DELETE

We can use the DELETE clause to remove rows from a table.

For example:

```
DELETE from table
WHERE row_id=1
```

- We can delete rows based on their presence in other tables.

e.g.

```
DELETE FROM tableA
USING tableB
WHERE tableA id= tableB.id
```

We can delete all rows from the table

e.g. DELETE FROM table

ALTER Table

The ALTER clause allows for changes to an existing table structure, such as

- Adding, dropping, or renaming columns
- changing A column's data type
- Set default values for a column
- Add check constraints.
- Rename table.

DATE

genral syntax:
ALTER TABLE table_name action

eg. Adding columns
- ALTER TABLE table_name
  ADD COLUMN new_colTYPE

Removing columns.
- ALTER TABLE table_name
  DROP COLUMN col_name

- DROP

drop allows for the complete removal of column in a table

In postgsql this will also automatically remove all of its indexes and constraints involving the column.

however it will not remove columns used in views, triggers or stored procedures without the additional CASCADE clause.

General syntax:
- ALTER TABLE table_name
  DROP COLUMN col_name

Remove all dependencies.
- ALTER TABLE table_name
  DROP COLUMN col_name CASCADE

classmate

PAGE

DATE

- Check for existence to avoid error

```
ALTER TABLE table_name
DROP COLUMN IF EXISTS col_name;
```

Drop multiple columns

```
ALTER TABLE table_name
DROP COLUMN col_one,
DROP COLUMN col_two;
```

PAGE

classmate

DATE

continoue

- Timestamps and extract - Part two

Time based datatype →
- EXTRACT()
- AGE()
- TO_CHAR()

- EXTRACT() →
  Allows you to 'extract' or obtain a sub-com-ponent of a date value

  EXTRACT (year from date-col)

- AGE()
  calculates and returns the currents age given a time_stamp.
  Useage:
  - AGE (date_col)

  Return
  13 year 1 mon 5 dys 01:34:13.003423

- TO_CHAR()
  General function to convert data types to text.
  Useful for timestamp formating.

  Usage:
  TO_CHAR (date_col, 'mm-dd-yyyy')

DATE

~~Year~~ • YEAR • MONTH • DAY • WEEK • QUARTER

DATE

Note: ∅

dow Keyword return numeric value (1 to 7) representing the day of the week for a specified date or datetime.

↓

Found similar keyword for month

eg.

```
SELECT COUNT(*)
FROM payment WHERE EXTRACT
(dow FROM payment_date) = 1
```

. Mathematical functions and operators.

DATE

- String functions and operators →

PgSQL also provides a variety of string functions operations that allow us to edit, combine and alter text data columns.

note: go for docum...

DATE

- subquery →
  A subquery allows you to construct complex queries, essentially performing a query on the result of another query.
  The syntax is straight forward and involves two SELECT statements.
  ① we have find the student grade was greater than AVG?

```
e.g. SELECT student_grade
     FROM test_scores
     WHERE grade > (SELECT AVG(grade)
                    FROM test_scores)
```

The subquery is performed first since it is inside the parenthesis.
We can also use the 'IN' operator in conjunction with a subquery to check agaist multiple results returned.

```
e.g. SELECT student_grade
     FROM test_scores
     WHERE student IN
      (SELECT student
       FROM honor_roll_table)
```

DATE

- The EXISTS operator is used to test for existence of rows in a subquery.
- Typically a subquery is passed in the EXISTS() function to check if any rows are returned with the subquery.
- The result of EXISTS is a boolean value True or false.

Syntax:

```
SELECT column_name
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name
 WHERE condition);
```

e.g. select payment_id, amount from payment
where amount < (select AVG(amount) from payment);

2.

```
Select film_id, title
from film
where film_id IN
(SELECT inventory.film_id FROM rental
 Inner join inventory ON inventory.inventory_id = rental.inventory_id
WHERE return_date BETWEEN '2005-05-29' AND '2005-05-30')
```

DATE

Q. Find the first name & last name of the customer whoes at least one payment (amount) is greater than 11?

```
select first_name, last_name
from customer AS C
where EXISTS
 (Select *FROM payment as p WHERE p.customer
 _id = C.customer AND amount >11)
```

also used joining here.

self join

A self-join is a query in which a table is joined to it'self.

Self joins are useful for comparing values in a column of rows within the same table.

- The self join can be viewed as a join of two copies of the same table.
- The table is not actually copied, but SQL perform the command as though it were.
- There is no special keyword for a self join it's simply standard JOIN syntax with the same table in both parts.

compositekey → a composite key is made by the combination of two or more coloumns in a table and can be used to uniquely identify each row in the table when columns are combined uniqueness or it can also be understood as a primary key made by the combination of two or more attribute to uniquely identify every row in a table

- However, when using a self join it is necessary to use an alias for the table otherwise the table names would be ambiguous.

Syntax:

```
Select tableA.col, tableB.col
From table AS tableA
Join
```

```
Select tableA.col,
From table AS tableA
Join table AS tableB ON
tableA.some_col = tableB.other_col
```

employees.

emp_id	name	report-id
1	Andrew	3
2	bob	3
3	charlie	4
4	david	1

we want results showing the employee name & their reports recipient name

```
SELECT empname, repot.name
FROM employee Asemp
Join employee AS reporton
emp.emp_id = report.report-id
```

name	rep
Andrew	charlie
bob	charlie
charlie	David
David	Andrew

DATE

e.g 1.
Find the student id who is enrolled in at least two courses.

Join = Crossproduct + condition

Study

S_id	C_id	Since
S_1	C_1	2016
S_2	C_2	2017
S_1	C_2	2017

```
Select T1.S_id
From Study as T1,
     Study as T2
where T1.sid = T2.S_id
   AND
     T1.C_id != T2.C_id
```

Self join

T_1	T_2
$S_1 C_1$	$S_1 C_1$
$S_1 C_1$	$S_2 C_2$
$S_2 C_1$	$S_1 C_2$
$S_2 C_2$	$S_1 C_1$
$S_2 C_2$	$S_2 C_2$
$S_2 C_2$	$S_1 C_2$
$S_1 C_2$	$S_1 C_1$
$S_1 C_2$	$S_2 C_2$
$S_1 C_2$	$S_1 C_2$

PAGE

DATE

Creating data-base and tables.

- Data-type → go to docu ---

DATE

(Restriction).

- Constraints →

Constraints are the rules enforced on data columns on table.

These are used to prevent invalid data from being entered into the database.

This ensures the accuracy and reliability of the data in the database.

- constaints can be divided into two main categories

- columns constraints:
  - constraints the data in a column to adhere to certain condn.

- Table Constraints:
  - applied to the entire table rather than to an individual column.

- The most common constraints used:

- NOT NULL Constriant
  - ensure that a column cannot have NULL value.

- UNIQUE constraint.
  - ensure that all values in a column are diff.

classmate

PAGE

DATE

- Primary Key : Not Null + Unique.
  Uniquely identifies each row/record in database table.

- Foreign Key :
  Constrains data based on columns in other tables.

- CHECK constraint
  ~~all values~~ ensure that all values in a column satisfy certain conditions.

- EXCLUSION constraint
  - ensure that if any two rows are compared on the specified column or expression using the specified operator, not all of these comparisons will turn true.

- Default
  The Default constraint is used to fill a column with a default and fixed value. The value will be added to all new records when no other value is provided.

- CHECK (condition)
  - to check a condition when inserting or updating data.

- REFERENCES
  To constrain the value stored in the column that must exist in a column in another table.

DATE

- UNIQUE (column_list)
  - forces the values stored in the columns listed inside the partheses to be unique
- PRIMARY KEY (column_list)
  - Allows you to define the primary key that consists of multiple columns.

∞ ∞

DATE

- Create table →

```
CREATE TABLE table_name
    (column_name TYPE column_constraint,
    column_name TYPE column_constraint,
    table_constraint table_constraint);
    INHERITS existing_table_name;
```

- Common simple syntax:

```
CREATE TABLE table_name
    (column_name TYPE column_constraint,
    column_name TYPE column_constraint);
```

- Serial data type →

In SQL sequence is a special kind of data-base object that generates a sequence of integers.

A sequence is often used as the primary key column in a table

It will create a sequence object and set the next value generated by the sequence as the default value for the column.

This is prefect for primary key, because it lays unique integar entries for you automatically upon insertion.

DATE

If a row is later removed, the column with the SERIAL data type will <u>not</u> adjust, marking the fact that a row was removed from the sequence. For e.g

1, 2, 3, 5, 6, 7

You know row 4 was removed at some point.

TABLE

DATE

e.g. 1. CREATE TABLE players
(player_id SERIAL PRIMARY KEY,
age SMALLINT NOT NULL);

eg. 2.

```
CREATE TABLE information(
info_id SERIAL PRIMARY KEY,
title VARCHAR(500) NOT NULL,
Person VARCHAR(50) NOT NULL UNIQUE
)
```

CHECK constraint ⟶

The CHECK constraint allows us to create more customized constraints that adhere to a certain condn.

Such as making, sure all inserted integer values fall below a certain threshold.

Genral syntax ⟶

```
CREATE TABLE example
(
    ex_id SERIAL PRIMARY KEY,
    age SMALLEST CHECK (age > 21),
    parent_age SMALLINT CHECK (parent_age
        > age)
);
```

DATE

Conditional expressions and procedures →

cases →

We can use the CASE statement to only execute SQL code when certain condn are met.
These is very similar to IF/else statement in other programming languages.
There are two main ways to use a CASE statement either a general CASE or a CASE expression
Both method can lead the same result.

genral syntax →

```
CASE
    WHEN condition1 THEN result1
    WHEN condition 2 THEN result 2
    ELSE some_other_result
END
```

eg.

```
SELECT a
CASE
    WHEN a = 1 THEN 'one'
    WHEN a= 2 THEN 'Two'
ELSE 'other'
END
FRom test
```

→

a	case
1	one
2	two

DATE

The CASE Expression syntax First evaluates an expression then compares the result with each value in the WHEN clauses sequentially.

Case expression syntax →

```
CASE expression
    WHEN value1 THEN result1
    WHEN value2 THEN result2
    ELSE some_other_result
END
```

Rewriting our previous example,

```
SELECT a,
  CASE a
  WHEN 1 THEN 'one'
  WHEN 2 THEN 'two'
  ELSE 'OTHER'
  END AS lable
FROM test;
```

a	lable
1	one
2	two

DATE

COALESCE

The COALESCE function accepts an unlimeted number of arguments. It returns the first argument that is not null.

If all arguments are null, the COALESCE function will return null.

COALESCE (arg1, arg-2, ---arg-32)

example

- SELECT COALESCE (1, 2)
  1
- SELECT COALESCE (NULL, 2, 3)
- 2

The COALESCE function become useful when querying a table that contain null values and substituting it with another value.

- Product

Item	Price	Discount
A	100	20
B	300	NULL
C	200	10

Whatis the final price after discount

DATE

SELECT item, (price - discount) AS Final
FROM table

Item	Final
A	80
B	null
C	190

Doesn't work for item B, should be 300.

~~Item Fin~~

SELECT item, (price - COALESCE (discount, 0))
AS Final FROM table

Item	Final
A	80
B	300
C	190

Keep the COALESCE function in mind in case you encounter a table with null values that you want to perform operations on!

PAGE

DATE

CAST →

The CAST operator let's you convert from one data type into another

Keep in mind not every instance of a data type can be CAST to another the data for, it must be resonable to convert the data,

e.g "5" to an integer will work,
"five" to an integer will not.

Syntax:

```
SELECT CAST ("5" AS INTEGER)

SELECT "5"::INTEGER
```

you can then use this in a SELECT query with with a column name instead of a single instance.

- SELECT CAST (date AS TIMESTAMP)
  FROM table

DATE

NULLIF

The NULLIF function takes in 2 inputes and return NULL if both are equal, otherwise it return the first argument passed.

- NULLIF (arg1, arg2)

e.g

NULLIF (10,10)
return NULL

This becomes very useful in cases where a NULL value would cause an error or unwanted result.

PAGE

classmate

DATE

View

A view is a database object that is of a stored query.

A view can be accessed as a virtual table in postgre SQL.

Notice that a view does not store data physically, it simply stores the query.

you can also updating and alter existing views

genral syntax :-

```
CREATE VIEW name AS
query ----
```

for removeing the view →

```
DROP VIEW IF EXISTS (~~cust~~ name ---)
```

classmate

PAGE

DATE

Import and Export

Import/Export functionality of pg admin which allows us to import data from a csv file to an already existing table.

There are some imp. notes to keep in mid when using Import/Export.

not every outside data file will work, variations in formatting, macros, data types, etc may prevent the import command from reading the file, at which point, you must edit your file to be compatible with SQL.