→ `WHERE` clause

a. It can be used with any of the operators
b. It is used to filter the row returned by `SELECT` statement
c. It allows you to specify a condition, that each row must meet in order to be included in the result set
d. Syntax:

```
SELECT <column_name> FROM <table_name> WHERE <condition>;
```

e. This condition can include one or more expression that compares column values to literal value or other column
f. We can use operators like relational operators, logical operators in `WHERE` condition

→ Operators in SQL

a. It has following operators in precedence order of high to low
    i. <u>Relational operators</u>
        1. > greater than
        2. >= greater than or equal to
        3. < less than
        4. <= less than or equal to
        5. != or <> is not equal to
        6. = is equal to [don;t put '==']
    ii. <u>Logical Operators</u>
        1. NOT Logical NOT
            a. It"ll give the output which is not present in the condition
            b. Example:

```
SELECT * FROM employees WHERE SALARY NOT IN (24000, 17000);
```

        2. AND logical AND
            a. It"ll give output where both the conditions match
            b. Example:

```
SELECT * FROM employees WHERE department_id=100 OR SALARY>6000;
```

        3. OR logical OR
            a. It"ll give output where any of the conditions match
            b. Example:

```
SELECT * FROM employees WHERE department_id=100 OR SALARY>6000;
```

Query 1:

```
SELECT * FROM employees WHERE DEPARTMENT_ID=60 OR DEPARTMENT_ID=20 AND
SALARY=17000;
```

- In Query 1, `DEPARTMENT_ID=60` or `DEPARTMENT_ID=20` will return employees who either belong to dept 20 or 60.
- The condition `SALARY=17000` will be connected to what previous condition of dept id being 60 or 20, which means condition will show employee whose `DEPARTMENT_ID` is

60, hence, it'll lead to incorrect result as the condition is applied to `employees` who belong to either of the employees

Query 2:

```
SELECT * FROM employees WHERE (DEPARTMENT_ID=60 OR DEPARTMENT_ID=20)AND
SALARY>17000;
```

-AND operator will get precedence over OR operator
-In Query 2, in this condition, `DEPARTMENT_ID=60` or `DEPARTMENT_ID=20` are grouped together within parenthesis.
-This ensures that OR condition will be evaluated first and then only condition `SALARY>17000` will be applied to employees who belong to either `DEPARTMENT_ID=60` OR `DEPARTMENT_ID=20`

NOTE: Always use the parenthesis when you are using multiple logical operators to ensure that the conditions are evaluated correctly

iii. Arithmetic operator

1. () Parenthesis
2. ** Exponent
3. / division
4. * multiplication
5. + addition
6. - subtraction

→ Pre-defined functions

a. These predefined functions are divided into categories:

i. Aggregate Functions

1. avg():

a. This function is used to calculate the average of the column
b. Syntax for avg() is

```
SELECT avg(column_name) FROM table_name;
```

c. Example:

```
SELECT avg(SALARY) FROM employees;
```

2. sum():

a. This function is used to calculate the summation of all the records in the column
b. Syntax for sum() is

```
SELECT sum(column_name) FROM table_name;
```

c. Example :

```
SELECT sum(SALARY) FROM employees;
```

3. count():

a. This function is used to count all the records in the column
b. Syntax :

```
SELECT count(*/column_name) FROM table_name;
```

c. Example:

```
SELECT count(*) FROM employees;
```

4. max():

a. This function is used to find the maximum value from the column
b. Syntax:

```
SELECT max(column_name) FROM table_name;
```

c. Example:

```
SELECT max(SALARY) FROM employees;
```

5. min():

a. This function is used to find the maximum value from the column
b. Syntax:

```
SELECT min(column_name) FROM table_name;
```

c. Example:

```
SELECT min(SALARY) FROM employees;
```

ii. Comparison Function

1. The comparison function compares different values

2. isnull():

a. This check for the value whether it is present in the list or not
b. Example:

```
SELECT FIRST_NAME FROM employees WHERE isnull(SALARY);
```

3. least(column_name, value):

a. This will return the least values from the column which are less than the specified value
b. Syntax:

```
least(column1, column2, column3)
least(num1, num2, num3)
least(str1, str2, str3)
least(date1, date2, date3)
```

c. This works for any data type and returns the least among the values
d. This can u=be used in scenarios where we have to give a discount of a certain percentage, then you'll use this function
e. We can pass up to 255 values as arguments to it

4. **greatest(column_name, value):**
    a. This will return the greatest value from the column which are greater than the specified value

NOTE: Whenever you do a comparison with NULL, it'll always return NULL. These are also referred to as pessimistic queries, which means that we are searching for NULL values

```
SELECT SALARY FROM employees WHERE SALARY=NULL;
```

5. **IN** operator:
    a. It operator allows you to specify multiple values in where clause
    b. `IN` operator is a shorthand for `OR` operator
    c. Example:

```
SELECT * FROM employees WHERE DEPARTMENT_ID IN (10, 20,30);
SELECT * FROM employees WHERE SALARY NOT IN (24000, 17000);
```

6. **BETWEEN** operator:
    a. The `BETWEEN` operator selects values within a given range. The values can be numbers, text, or dates.
    b. The `BETWEEN` operator is inclusive: begin and end values are included.
    c. Example of `BETWEEN` operator:

```
SELECT * FROM employees WHERE SALARY BETWEEN 10000 AND 12000;
```

7. **LIKE** operator:
    a. This operator is used to search a specific pattern in the column
    b. It supports 2 wildcard characters
    c. '%' represents 0 or more characters
    d. '_' represents single characters
    e. Example:

Names starting with 's' are printed

```
SELECT FIRST_NAME FROM employees WHERE FIRST_NAME LIKE 's%';
```

Names ending with 'a' are printed

```
SELECT FIRST_NAME FROM employees WHERE FIRST_NAME LIKE '%a';
```

Names containing 'a' in between are printed

```
SELECT FIRST_NAME FROM employees WHERE FIRST_NAME LIKE '%a%';
```

Names with 'a' as second character are printed

```
SELECT FIRST_NAME FROM employees WHERE FIRST_NAME LIKE '_a%';
```

iii. String Function

1. **concat():**
    a. It is used to concatenate two columns

b. Example:

```
SELECT concat(FIRST_NAME, " ", LAST_NAME) AS EmployeeName FROM employees;
```

2. instr():

a. When we want to find the first occurance of any substring, then we use this function

b. Example:

```
SELECT instr(FIRST_NAME, 'ar') AS pos_of_ar FROM employees;
```

3. length()

a. If you want to find the length of any column in bytes and characters, we use length() function

b. Returns the number of bytes given in a string

c. Example:

```
SELECT length(FIRST_NAME) as len_bytes FROM employees;
```

4. char_length():

a. Returns the number of characters in a string

b. Example:

```
SELECT char_length(FIRST_NAME) as length_chars FROM employees;
```

c. Difference between length() and char_length() is that the number of bytes may not be same as the number of characters, but you may get different results while using the character set UTF-8

5. left():

a. If you want to get first ‘n’ characters of any column, we will use the `left()` function

b. First parameter is the column name, second parameter is number of first characters you want

c. Example:

```
SELECT left(FIRST_NAME, 3) FROM employees;
```

6. right():

a. If you want to get last ‘n’ characters of any column, we will use the right() function

b. First parameter is the column name, second parameter is number of first characters you want

c. Example:

```
SELECT right(FIRST_NAME, 3) FROM employees;
```

7. substring()

a. When you want to extract substring with their position value and with specified length then we use this function.

b. Syntax:

c.

```
SELECT substring(column_name, start_position, number_of_characters) FROM
table_name;
```

d. Example:

```
SELECT substring(FIRST_NAME, 3, 2) FROM employees;
SELECT substring(FIRST_NAME, 3) FROM employees;
```

e. Realtime example of polymorphism can be `substring()` function

8. substring_index():

a. This function is used to extract substring from a string, based on delimiter that separates the substring based on the string
b. It takes the arguments as
   i. the original string,
   ii. Delimiter
   iii. counter parameters that specify which occurrence of delimiter to use for substring extraction
c. Example:

```
SELECT substring_index(STREET_ADDRESS, ' ', -1) AS street FROM locations;
```

9. ltrim():

a. It'll remove the unwanted space from the left side
b. Syntax:

```
SELECT ltrim(column_name) from table_name;
```

10. rtrim():

a. It'll remove the unwanted space from the right side
b. Syntax:

```
SELECT ltrim(column_name) from table_name;
```

11. trim():

a. It'll remove the unwanted spaces/tab spaces at start/end
b. Syntax:

```
SELECT ltrim(column_name) from table_name;
```

12. lower():

a. This will convert the string to lower case
b. Syntax:

```
SELECT lower(column_name) FROM table_name;
```

c. Example:

```
SELECT lower(FIRST_NAME) AS fname FROM employees;
```

13. upper():

a. This will convert the string to upper case
b. Syntax:

```
SELECT upper(column_name) FROM table_name;
```

c. Example:

```
SELECT upper(FIRST_NAME) AS fname FROM employees;
```

14. **lpad():**
    a. This is used for right justification
    b. Syntax:

```
SELECT lpad(column_name, number_of_characters, delimiter) FROM table_name
```

    Example:

```
SELECT lpad(FIRST_NAME, 10, '*') AS fname, FIRST_NAME FROM employees;
```

15. **rpad():**
    a. This is used for left justification
    b. Syntax:

```
SELECT rpad(column_name, number_of_characters, delimiter) FROM table_name
```

    Example:

```
SELECT rpad(FIRST_NAME, 10, '*') AS fname, FIRST_NAME FROM employees;
```

16. **intcap():**
    a. It is used to make initial character in capital
    b. Syntax:

```
SELECT intcap(column_name) FROM table_name;
```

17. **replace():**
    a. This function is used to replace all the occurrence of substring within a string with new substring
    b. Syntax:

```
SELECT replace(column_name, character_to_be_replaced, replacement_character)
```

    c. Example:

```
SELECT replace(FIRST_NAME, 'a', 'Y') FROM employees;
```

18. **reverse():**
    a. This function is used to reverse the string
    b. Syntax:

```
SELECT reverse(column_name) FROM table_name;
```

    c. Example:

```
SELECT reverse(FIRST_NAME) FROM employees;
```

19. **locate():**
    a. This function will find the position of substring within a string
    b. This will return the position (not index) at which substring is found
    c. Syntax:

```
SELECT locate(substring_to_locate, column_name) FROM table;
```

    d. Example:

```
SELECT locate('ell', FIRST_NAME) FROM employees;
```

20.find_in set()

a. This function within a list of strings
b. Syntax:

```
SELECT find_in_set(string_to_be_searched, comma_separated_string_list) from
table_name;
```

c. Example:

```
SELECT DEPARTMENT_ID FROM departments WHERE find_in_set('IT', DEPARTMENT_NAME);
```

iv. Math Functions

1. In math functions, there are multiple functions like
   a. **abs()**:
      i. return absolute value of a number

   b. **floor()**:
      i. return the largest integer value that is greater than the argument passed

   c. **ceil()**:
      i. return the largest integer value that is greater than or equal to the argument passed

   d. **round()**:
      i. this will round off the decimal

   e. truncate():
      i. this returns the specified number of decimals

   f. mod():
      i. returns the mod

   g. pow():
      i. this function will return the power

v. Control Flow Functions

1. CASE

a. return the corresponding result in `THEN` branch if the condition in the `WHEN` branch is satisfied, otherwise, return the result in the `ELSE` branch.
b. `CASE` statement checks the condition and returns the value of the condition that matches, so if one condition is `TRUE`, it'll stop reading and return the result
c. If there is no else or none of the conditions is `TRUE`, it'll return `NULL`

d. It accepts two parameters:
    i. Condition
    ii. Result
e. It works by evaluating a series of conditions and it returns a result based on first condition that maybe TRUE
f. Syntax:

```
CASE Value
     WHEN condition1, THEN result1
     WHEN condition2, THEN result2
     WHEN condition3, THEN result3
     WHEN condition4, THEN result4
END
```

g. Example:

```
SELECT FIRST_NAME, LAST_NAME, SALARY,
CASE
     WHEN SALARY<=6000 THEN SALARY*.10
    WHEN SALARY<=10000 THEN SALARY*.20
    WHEN SALARY<=20000 THEN SALARY*.30
    ELSE SALARY*.5
END AS bonus
FROM employees;
```

2. IF

a. return a value based on a given condition.

3. IFNULL

a. return the first argument if it is `NOT NULL` , otherwise returns the second argument.

4. NULLIF

a. return `NULL` if the first argument is equal to the second argument, otherwise, returns the first argument.

vi. Date Functions
vii. Windows Functions

→ `ANY` operator

a. Analogous with `OR` operator in case of sub-queries
b. It'll return boolean values as a result
c. It returns TRUE if any of the value meets the condition
d. It is mostly used in the sub-queries to specify that the value (result should match any value returned by this subquery)
e. Example:

```
SELECT DEPARTMENT_ID, FIRST_NAME FROM employees WHERE DEPARTMENT_ID = ANY
(SELECT DEPARTMENT_ID FROM departments);
```

→ `IN` can be used to return a smaller list of values, `ANY` operator will be used when you have large list or when a subquery generates a result

→ `ALL` operator

a. Analogous with `AND` operator in case of subqueries
b. It is used in subquery to specify that result should match all the values returned by subquery
c. In `ALL` operator, we need to specify a condition using `WHERE` clause
d. `ALL` operator can be replaced with combination of `NOT` and `ANY` operator (because `AND=NOT(OR)`)

Problem / challenge:
Write a query, To show the bonus of employees who are working in a particular department.

a. If the employee is working in the Marketing department and earns less than 6000 then add a bonus of 5 %.
b. If the employee is working in the HR department and has a salary < 5000 then add a bonus of 6%.
c. If the the employee is working in IT dept. and earns salary > 10000 then add bonus of 10%

Soln:

```
SELECT FIRST_NAME, LAST_NAME, DEPARTMENT_ID, SALARY,
CASE
     WHEN SALARY<=6000 AND DEPARTMENT_ID=20 THEN SALARY*0.05 #
marketing dept
     WHEN SALARY<=5000 AND DEPARTMENT_ID=40 THEN SALARY*0.06 # HR
dept
     WHEN SALARY>=10000 AND DEPARTMENT_ID=60 THEN SALARY*0.5 # IT
dept
END AS bonus
FROM employees WHERE DEPARTMENT_ID IN (20, 40, 60);
```

NOTE:

a. Sequence in which SQL retrieves the data is how the developer or DBA writes an SQL query
b. Example:

```
SELECT FIRST_NAME, SALARY+1000 increment FROM employees ORDER BY
increment;
```

c. Below sequence is followed while writing or executing an SQL query

i. SELECT
ii. FROM
iii. WHERE
iv. GROUP BY
v. HAVING
vi. ORDER BY

→ LIMIT clause, OFFSET clause

a. LIMIT clause is used to restrict the number of rows returned by the SELECT statement
b. Syntax:

```
# limits the number of rows to return
SELECT column_name FROM table_name LIMIT 10 ;
# the number of rows to return(5) after the offset value(10) rows
SELECT column_name FROM table_name LIMIT 5 OFFSET 10 ;
```

c. Example:

```
SELECT * FROM employees LIMIT 10;
SELECT * FROM employees LIMIT 2 OFFSET 10;
```

d. Row offset: This specify the number of rows to skip before starting to return the row

→ distinct() function

a. DISTINCT keyword is used to return distinct(remove duplicate rows) based on specific columns
b. Example:

```
SELECT distinct(FIRST_NAME) FROM employees;
```

→ ORDER BY clause

d. ORDER BY clause is used to sort the result of the query in ascending or descending order
e. Syntax:

```
SELECT column_name FROM table_name ORDER  BY column_name (ASC/DESC)
```

Example:

```
SELECT SALARY FROM employees ORDER BY SALARY;
SELECT SALARY FROM employees ORDER BY SALARY DESC;
```

f. ASC stands for ascending order and DESC stands for descending order
g. By default, it'll sort in ascending order
h. In ORDER BY clause, sorting will be performed at server RAM, ORDER BY does sorting at RAM only so that sorting is performed before sending response to the client
i. ORDER BY clause will be the last statement of any SQL query

j. Note (while using `ORDER BY` clause):

```
SELECT SALARY, EMPLOYEE_ID FROM employees ORDER BY SALARY ASC,
EMPLOYEE_ID DESC;
```

In the above command, you've selected `EMPLOYEE_ID` and `SALARY`. It'll retrieve the data first by `SALARY` in ascending order, and then the `EMPLOYEE_ID` in descending order. This will give the result in ascending order of the `SALARY` column