- Select: Column name.
- From: Table Name
- Where: Logical Condition
- Group By: columnName
- Having: Can use only apply when group by is present. Logical Condition
- Order by : Column Name

1. **The USE Statement**
    - USE; : It mention =

TABLE NAME CITY

2. **Selecting specific Columns**:
   Ex:
   Select Id, name, CountryCode from city;

3. **Select Distinct:**
   Distinct will just return the unique values on those columns you mention.
   Ex,
   Select distinct CountryCode from city;

4. **Where Clause:**
   Can filter records using (=, <>, <, >, >=, <=, BETWEEN, LIKE, IN()). Any of those logical operators.

   Ex,
   (=) : select * from city where CountryCode = 'USA' limit 5;
   (>): select * from city where ID > '5' limit 5;
   (<=): SELECT * FROM city where ID <= 10;

5. **Between & And**

Using And is to return records which satisfy all criteria.

Ex, SELECT * FROM city where ID between 1 and 10;
Which will give only 1 to 10 value

6. **Using Where clause & And at the same time.**

Ex,

- select ID, Name, CountryCode, Population from city where ID > 10 and Population <= '90000' limit 15;

- select ID, Name, CountryCode, Population from city where ID > 10 and CountryCode = 'USA' limit 15;

- select ID, Name, CountryCode, Population from city where Name = 'New York' and CountryCode = 'USA' limit 15;

7. Multiple AND at the same Time In Where Clause:
   Ex,
   select ID, Name, CountryCode, Population from city where Name = 'New York' and CountryCode = 'USA' and Population < 82000000 ;

8. **Where & OR Clause;**
   - OR - You can add multiple logical conditions in your WHERE clause using an OR statement.
   - Use OR to return records which satisfy any criteria.

- It show you the number of Row in one criteria have ex,ID 5 have (11 rows), and ID 11 have (20 rows).

Ex,

```
 select ID, Name, CountryCode, Population from city where
Population <= 8200 or Population <=1000 or Population <= 7000;
```

```
select ID, Name, CountryCode, Population from city where ID = 12 or
ID= 11 or ID = 13;
```

```
 select ID, Name, District, Population from city where Population <
1000 or Population = 400 or Population = 800;
```

9. **WHERE & IN**
   In condition do exact same thing like OR condition do.
   Ex,
   ```
    select ID, Name, District, Population from city where ID <
   10 or ID in (12,13,14);
   ```

10. **Like**
    Allows you to use pattern matching in your logical operators. Instead of exact matching.
    Ex,
    - select Name, CountryCode from city where CountryCode like '%USA%';
    - select Name from city where Name like '%New York%';

- select Name, CountryCode from city where CountryCode like '%AFG%';

11. **Group By Clause**

Group By contains columnName and OtherColumnName. Group By is great for comparing different segments of your data. Such as like,

- Group By Category
- Group By Store Location
- Group by category, store location

GroupBy is optional and its come after any WHERE Clause and before any HAVING or ORDER BY clauses in your query.

12. **Group Aggregation**

Combine GROUP BY with aggregate functions like COUNT, or SUM to specify how values are summarized for each each group.

Ex Of Count:

- select Name CountryCode, count(Name) from city group by CountryCode;
- select Population, count(Name) from city group by Population;

13. **Comments & Aliases**

- Comments can be added to your code using “--” or “/*”, which tells SQL to ignore those lines.

  Comments can apply to entire lines, portions of a lines, Or multiple lines.

- Aliases allow you to assign a custom name to a field in your result set, using an AS statement.

  Ex of AS:

  - select Population, count(Name) as Ahad from city group by Population; (This query shows that when you rename your column name using AS).

  - select population, count(name), count(Name) as total_name from city group by Population; (This query describes you more clearly what you doing on your table using AS).

Multiple Group By: This allows you to create groups and sub-groups in your result set, just like specifying multiple row or column labels in a Table.

Ex:

- select ID, Name, District, count(ID) as NUMBEROF_ID from city group by ID, Name, District; ~ (This shows using multiple group by show you the result of counting each ID have how many total columns.

14. Aggregate Functions

- **Count()** ~ Skip NULL, except COUNT(*)
- **Count distinct()** ~ Skip NULL
- **Min()** ~ Finds the smallest value
- **Max() ~** Find the largest Value
- **Avg() ~** Average of All Values
- **Sum() ~** Sum of All Values (Treat Null as Zero)

Ex,

- select ID, count(Population) as Ahad, min(Name) as short_name, max(Name) as long_name, avg(Name) as avarage_name from city group by ID limit 10;

- select CountryCode,count(ID) as total_ID, min(Name) as total_name, max(Name) as longer_name, avg(Name) as avg_name from city group by CountryCode limit 10;

15. The Having Clause

Using Having where you specify the filtering logic that you want applied to your group-level aggregated metrics.

Having can only be used with GROUP BY, if you are trying to filter your results, but aren't grouping with GROUP BY, then you should use a WHERE clause instead.

Such as,

- Having count(*) > 1
- Having sum(payment)>10
- Having min(rental-date)< '01-05-2021'

EX,

- select CountryCode, count(*) as total_Country from city group by CountryCode having count(*) >= 10;
- select CountryCode, count(*) as total_Country from city group by CountryCode having count(*) <= 10;
- select CountryCode, count(*) as total_Country from city group by CountryCode having count(*) = 10;

16. Order By

SQL lets you sort your results by including ORDER BY after your FROM clause and after WHERE, GROUP BY, and HAVING clauses, if applicable.

ORDER BY defaults to ascending order(low to high), but can be modified to sort in descending order by adding DESC after the column reference.

Ex,

- DESC: select ID, Name, CountryCode from city order by CountryCode desc limit 10;
- select ID, Name, CountryCode from city order by CountryCode, Name desc Limit 10;
- select CountryCode, sum(Name) as total_Name from city group by CountryCode order by sum(Name) desc limit 10;
- select ID, sum(Population) as total_country from city group by ID order by sum(Population) desc limit 10;

17. The CASE Statement

Case statements execute in the order they appear, if a record satisfies more than one logical condition, the record will be assigned by the first then statement.

Case statements allow you to use conditional logic to specify how your results should be calculated for various cases.

One of the most common application for CASE is to “Bucket” value, as show

EX,

Case

When category in (“horror’, ‘suspense’) then ‘too scary’ when length >90 then ‘too long’ else ‘we should see it’

End;

- select distinct ID,
  ```
  -> case
  -> when ID < 10 then 'they are good'
  -> when ID between 11 and 10 then 'they are fine'
   -> when ID > 21 then 'they are bad'
   -> else 'they are sucks'
  -> end as id_bucket
   -> from city limit 30;
  ```

- select distinct ID, case when ID < 5 then 'they are okay' when ID between 6 and 8 then 'they are good' when ID > 10 then 'they are bad' else 'they are sucks' end as id_bucket from city limit 20;

- select Name, case when ID = 1 and CountryCode = District then 'name one is active' when ID = 1 and CountryCode = Population then 'store one is inactive' when ID = 2 and CountryCode = District then 'store two is active' when ID = 2 and CountryCode = Population then 'store two is inactive' else 'try again later' end as ID_bucket from city limit 15;

- select ID, case when ID = 1 then 'student is pumped' when CountryCode = District then 'student is learning' when

CountryCode = District then 'student got smarter' else 'UH.OH.CHECH LOGIC' end as student_status from city;

❖ CASE & COUNT & GROUP BY & ORDER BY

- select ID, Name, count(case when CountryCode = District then ID else null end) as count_id_1, count(case when CountryCode = District then ID else null end) as count_id_2 from city group by ID, Name order by ID;

- select ID, count(case when CountryCode = Population then ID else null end) as ID_1, count(case when CountryCode = Population then ID else null end) as ID_2 from city group by ID;

18. JOIN

➢ Inner join ~ from leftTableName INNER JOIN rightTableName. (Return record that exist in both tables, and exclude unmatched records from either table).

➢ Left Join ~ from leftTableName LEFT JOIN rightTableName. (Returns all records from the LEFT tables, and any matching records from RIGHT tables).

➢ Right Join ~ from leftTableName RIGHT JOIN rightTableName. (Returns all records from the RIGHT table, and any matching records from the left tables).

- Full Outer Join ~ From leftTableName FULL JOIN rightTable.Name (Returns all records from Both tables, including non-Matching records).

- Union ~ Select From firstName UNION select from secondName. (Returns all data from one table, with all data from another table appended to the end).