SQL COMMANDS

DML: Data Manipulation Language.
Insert, Select, Update, Delete (CRUD) which we will execute
DDL: Data Definition Language.
Create, Alter, Drop which is mostly done by developers

Some of The Most Important SQL Commands

- **SELECT** - extracts data from a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database
- **INSERT INTO** - inserts new data into a database
- **CREATE DATABASE** - creates a new database
- **ALTER DATABASE** - modifies a database
- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

NOTE: Column and Row index numbers start from '1' not '0' in SQL...

DATABASE CONNECTION WITH JDBC LIBRARY

```
Connection connection = DriverManager.getConnection(URL,user,password);
Statement statement = connection.createStatement();
ResultSet resultset = statement.executeQuery("query");

resultset.close();
statement.close();
connection.close();

DatabaseMetaData db = connection.getMetaData();
ResultSetMetaData rs = resultSet.getMetaData();
```

Oracle RDBMS
Oracle developed by Oracle Corporation is the most popular relational database system (RDBMS). Not only Oracle is a RDBMS, but also provides functionality for Cloud, Document Store, Graph DBMS, Key-value storage, BLOG, and PDF Storages. Recently, Oracle just announced autonomous feature that allows database to be intelligent and self-managed.

The current version of Oracle Database is 18c.

Oracle database is a relational database (RDBMS). Relational databases store data in a tabular form of rows and columns. Column of a database table represents the attributes of an entity and rows of a table stores records. An RDBMS that implements object-oriented features such as user-defined types, inheritance, and polymorphism is called an object-relational database management system (ORDBMS). Oracle Database has extended the relational model to an object-relational model, making it possible to store complex business models in a relational database.

One characteristic of an RDBMS is the independence of physical data storage from logical data structures.

In Oracle Database, a database schema is a collection of logical data structures, or schema objects. A database user owns a database schema, which has the same name as the user name.

Schema objects are user-created structures that directly refer to the data in the database. The database supports many types of schema objects, the most important of which are tables and indexes.

A schema object is one type of database object. Some database objects, such as profiles and roles, do not reside in schemas.

SELECT/DESCRIBE

- **select * from countries**; ==> retrieves all columns from countries table.
- **describe employees**; ==> Retrieves all specifications/features of the table.
- **select country_name, country_id from countries**; ==> retrieves only two specific columns from countries table.
- **select count(*) employees**; ==> retrieves row number of the table.
- **select** jobs.job_id , job_history.employee_id **from jobs, job_history**; selects different columns from different tables through the same command

WHERE / AND / OR / IN / NOT / IS NULL / IS NOT NULL

- select * from employees **where** employee_id>50 **and** salary>7000;
- select * from employees **where** employee_id>50 **or** salary>7000;
- select * from employees **where** employee_id>50 **and** (salary>7000 **or** last_name='King');
- select * from employees **where** employee_id>50 **and not** last_name='King';
- select * from employees where **first_name='Janette'**; ==> retrieves only Janette. **(USE SINGLE QUOTE)**
- select * from employees **where** first_name='John' **and** age=40; ==> retrieves only named John of which has the age 40 at the same time.
- select * from employees **where** department_id **is null**;
- select * from employees **where** department_id **is not null**;
- select first_name, last_name from employees **where** salary>3000 **and** salary<10000; it is the same as below.
- select first_name, last_name from employees **where** salary **between** 3000 **and** 10000; it is the same as above.
- select job_title from jobs **where** job_title='President' **or** job_title='Sales Manager' **or** job_title='Finance Manager';it is the same as below.
- select job_title from jobs **where** job_title **IN** ('President', 'Sales Manager', 'Finance Manager'); it is the same as above.
- select country_name from countries **where** country_id **not in** ('US', 'CA'); retrieves all country names except USA and Canada.

SELECTING COLUMNS FROM MULTIPLE TABLE

- select c.country_name, l.city from countries c, locations l;
- select e.first_name, d.department_name from employees e, departments d;
- select c.country_name, d.department_name from countries c, departments d;
- select e.first_name, l.city from employees e, locations l;

DISTINCT

- select **distinct** * from employees; ==> retrieves any row if it has at least a single unique column.

- select **distinct** first_name from employees; ==> retrieves unique names from table. (removes duplicates)

- select **distinct count(*)** from employees; retrieve number of unique rows if any row has at least a single unique data.

- select **count(distinct** first_name**)** from employees; retrieve number of unique **first_name**s.

- select **distinct count(distinct Employee_id and last_name)** from employees; retrieve each data if Employee_id and last_name are unique.

CONCETANATION
- select first_name **||** ' ' **||** last_name, salary from employees; ==> Concatenates first_name and last_name columns and merges into one.

- select first_name **||** ' makes $' **||** salary **||** ' a month' as Salary List from employees; gives something like below:
Steven makes $24000 a month
Alexander makes $9000 a month
- select **concat**(e.first_name, last_name) as "Full Name" from employees;
- select **concat**(**concat**(e.first_name, ' '), last_name) as "Full Name" from employees;

AS
- select first_name **||** ' ' **||** last_name **as** "FULLNAME", salary from employees; ==> modifies the title of the column and gives NAME instead of FIRST_NAME. (as keyword in this case is optional)

ORDER BY / ASC / DESC
- select * from Employees **order by** employee_id; ==> retrieves in ascending order
- select * from Employees **order by** employee_id **ASC**; ==> retrieves in ascending order
- select * from Employees **order by** employee_id **DESC**; ==> retrieves in descending order
- select * from Employees **order by** employee_id, age; ==> retrieves in ascending **order by** employee_id. If any two employee_id is equal, it orders by age among those same employee_id datas.

- select * from Employees **order by** first_name **ASC**, last_name **DESC**; ==> retrieves in ascending order by first_name. If any two first_name is same, it orders by last_name among those same first_name data in descending order.

*** **If we do not use ASC or DESC, it will automatically order by ASC.**
- select * from employee **order by 1 asc**; ==> orders all table according to 1st column by ascending order.
- select * from employee **order by 3 desc**; ==> orders all table according to 3rd column by descending order.

ROWNUM

It shows number of rows of the result. Only works with less than (<) and less then and equals (<=), does not work with greater than (>) and equals (=).

- Select * from employees where **rownum < 6;** It will display first 5 rows
- Select first_name, salary from employees where **rownum <=5;** It will display first 5 row
- Select first_name, **rownum** from employees; It will give each row number of the first names as a separate column.

*** **select** first_name from (Select first_name, **rownum** from employees) where **rownum** = 6; in this case we can use rownum with '=' because we have already created rownum as a separate column and we are using that column for condition. But we cannot use rownum=6 directly without creating a subquery.

select min(salary) from (select **distinct** salary from employees **order by** salary **desc**) where **rownum**<6; Gives 5th highest salary…

LIKE / % / _

- select * from employees where first_name **like 'T%';** ==> retrieves rows which starts with 'T'.
- select * from employees where first_name **not like 'T%';** ==> retrieves rows which DOES NOT start with 'T'.
- select * from employees where first_name **like 'T__%';** ==> retrieves rows which starts with 'T' and contains at least 2 characters after 'T' (there are two under_score just after 'T' in command).
- select * from employees where first_name **like '%s';** ==> retrieves rows which ends with 's'.
- select * from employees where first_name **like '%om%';** ==> retrieves rows which contains 'om' in itself.
- select * from employees where first_name **like '_o%';** ==> retrieves rows of which has 'o' as second character in itself.
- select * from employees where first_name **like '%a%b%';** ==> retrieves rows of which contains 'a' and 'b' after 'a'.
- select * from employees where first_name **is null;** ==> retrieves rows of which has null value in first_name column.
- select * from employees where first_name **is not null;** ==> retrieves rows of which has null as name.

MAX / MIN

- select **max(**salary**)** from employees; ==> retrieves a single raw which has highest salary in table.
- select **min(**salary**)** from employees; ==> retrieves a single raw which has lowest age in table.

MATH OPERATORS

- select job_title, min_salary, **min_salary+2000** from jobs; ==> creates new column and adds 2000 to the salary column.
- select min_salary, **min_salary+2000** as "Raised Salary", **min_salary-1000** as "Reduced Salary", **min_salary*12** as "Annual Salary", **(min_salary+max_salary)/2** as "Average Salary" from jobs;

SINGLE ROW FUNCTIONS:

*** It will go to each and every row and execute the function.

UPPER/LOWER/INITCAP/LENGTH/TRIM/DUAL TABLE/ROUND/TRUNCATE/MOD are the subtitles of the SINGLE ROW FUNCTION...

UPPER / LOWER / INITCAP
- Select **upper(**first_name**)**, **lower**(last_name), **initcap**(email) from employees;
==> it converts all first names into uppercase letters in brand new column,
==> it converts all last names into lowercase letters in brand new column,
==> it converts all emails' first letters into uppercase letters and the other letters lowercase in brand new column.
- select first_name from employees where **lower(**first_name**)**='ellen'; ==> for searching without case sensitivity. It works like "ellen".toLowerCase.

LENGTH
- select first_name, **length(**first_name**)** from employees; ==> retrieves all first names and length of those names in another column.
- select first_name from employees where **length(**first_name**) between** 4 **and** 7;
==> gets all names which contains 4, 5, 6 or 7 letters in itself.
- select first_name from employees where **length(**first_name**) in** (4,6,7); ==> gets all names which contains 4 or 6 or 7 letters in itself. It will not retrieve 5 letters long names.

SUBSTR **substr**(ColumnName, Beginning Index, Ending Index)
- select **substr**(first_name, 0, 5) from employees;
- select concat(**substr**(first_name, 0, 1), substr(last_name, 0, 1)) as "Initials" from employees;
- select **instr**(first_name, 'e') from employees where first_name = 'Ellen';
it works like indexOf() method and Returns the first index of 'e' from 'Ellen'

REPLACE
Select **replace**(column_name, old_value, new_value) from TableName;
select **replace**(first_name, 'E', 'e') from employees;
select **replace**(first_name, 'Ellen', 'Ela') from employees;

TRIM
Trim(value)
- select **trim**(first_name) from employees;
- select **rtrim**(first_name) from employees; right trim
- select **ltrim**(first_name) from employees; left trim

DUAL TABLE (This is a kind of DUMMY Table which is provided by Oracle)
- Select 'Hello world' as Hello **from dual**; it will print 'Hello world' on console without using any real table from our database.
- Select 2+3 **from dual**; ==> it will print 5 on console without using any real table from our database.
==> **Dual** is a special table which is a dummy table that we can quickly test out our expressions...
- Select **sysdate** from **dual**; returns system date (DD-MM-YY)

ROUND / TRUNCATE / MOD
- Select **round** (8/3) result from **dual**; Result will be 3. it will round to the closest integer and uses dual table
(Exp for round: **round** (7/3)=>2 and **round** (8/3)=>3)

- Select **trunc** (8/3) result from **dual**; Result will be 2. it will round down to integer and uses dual table
- Select **trunc** (7/3) result from **dual**; Result will be 2 again. it will round down to integer
- Select **mod** (7,3) result from **dual**; Result will be 1. it will get the remainder(%) of parenthesis and uses dual table
- Select * from employees where **mod** (employee_id, 2)=1; Returns rows if employee_id remainder 2 is 1; (101,103,105, ……)

MULTI ROW FUNCTIONS:

*** It combines the result of multiple row and generates only a single result...

COUNT/AVG/MIN/MAX/SUM/AGGREGATE are the subtitles of the MULTI ROW FUNCTION...

COUNT

* Counting all row numbers in a table==> select **count(*)** from employees;
* Counting unique first name row numbers in a table: ==>select **count(distinct first_name)** from employees;

- select **count(**first_name**)**, last_name from employees;==> **gives compile error**, because we send a query for only one result(one row) and multiple results(multiple row) concurrently. Database cannot send different row number results at the same time...

- select **count(*)** as "DEPARTMENT COUNT" from employees where department_id=90; ==> Gives the count of how many people have department_id as 90 in the table.

AVERAGE

select **avg(**salary**)** as "Average Salary" from employees; ==> it returns average of the salaries with decimal number (8914.764758657657869)

select **round(avg(**salary**))** as "Average Salary" from employees; ==> rounds to the closer integer. (8915)
select **round(avg(**salary**), 2)** as "Average Salary" from employees; ==> it returns the average with only two digits right after decimal.(8914.23)
select **trunc(avg(**salary)) from employees; rounds to the below integer(8914)

MIN

select **min(**salary**)** as "Min Salary" from employees;

MAX

select **max(**salary**)** as "Max Salary" from employees;

SUM

select **sum(**salary**)** as "Total Salary" from employees;

GROUP BY/ HAVING

Group By groups all same data into one row. "**having**" works same as "where" however it only works with group by. We **cannot use** group by and where at the same command (There is an exception for using them together.

- select job_id, count(*) from employees **group by** job_id; Returns each different job_id in seperate row and how many employees with same job_id.

- select job_id, max(salary) from employees **group by** job_id; Returns each different job_id in one row and max salary for each job_id.

- select job_id, min(salary) from employees **group by** job_id; Returns each different job_id in one row and min salary for for each job_id

- select department_id, count(*)from employees **group by** department_id; Returns the number of employees in each department:

Returns all different department ids in one row for each id and total amount of employee for each department.

- select salary, count(*) from employees **group by** salary order by salary desc; It returns two separate columns. First shows the salary itself, second shows how many employees get this salary.

```
- select department_id as "Department ID",
             count(*) as "Employee Number",
             min(salary),
             max(salary),
             round(avg(salary))
 from employees group by department_id;
```

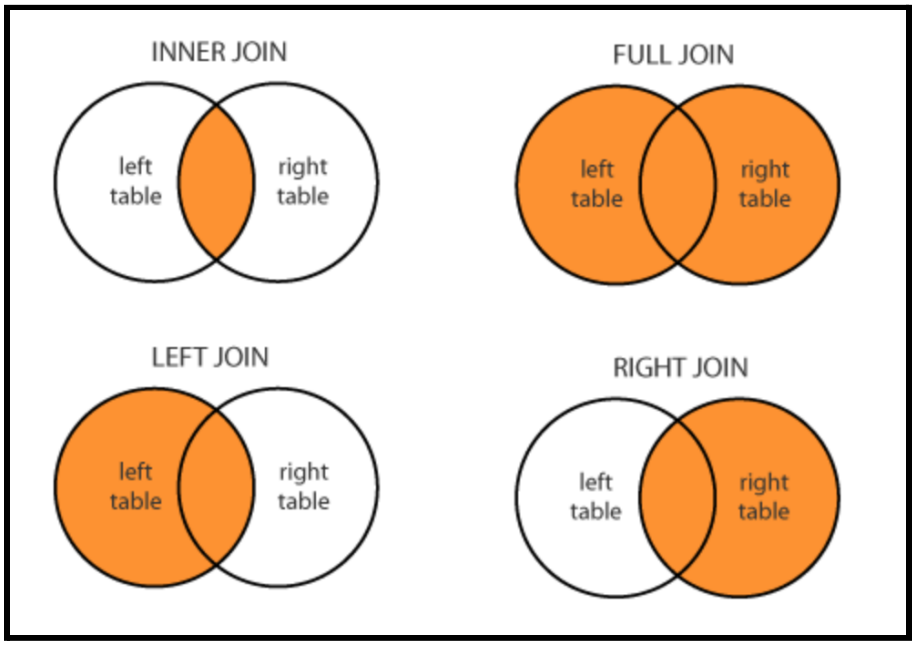
- select department_id, **count(*)** as "Employee Number" from employees **group by** department_id **having** count(*) > 5; It returns departments only if the employee number is higher then 5...

```
- select job_id as "Job ID",
        count(*) as "Employee Number",
        round(avg(salary)),
        min(salary),
        max(salary)
from employees group by job_id having min(salary)>4000 and max(salary)>10000
order by 4;
```

==> it will retrieve all the Job IDs from the table which has at least 4000 salary in order by column 4 (min(salary) in this case).

SQL JOINS

INNER JOIN: Only matching portion of the tables. Inner join eliminates if there is no match.
FULL JOIN: All part of the tables both matching portions and unmatching portions from both tables.
LEFT JOIN: Matching part from both table and unmatching part from left table.
RIGHT JOIN: Matching part from both table and unmatching part from right table.



Select * from **LeftTable l** join **RightTable r** on l.id=r.id

JOIN / ON
"**Join**" keyword enables us to combine two table and get data from different tables concurrently.
Select c.country_name, r.region_name from countries c **join** regions r **on** r.region_id=c.region_id; It means that; there is no region_name in countries table however region_id is foreign key(FK) in countries tables where it is primary key(PK) in regions table. So RELATIONAL DATABASE comes into picture here and it enables us to link two different table by using those PK and FK. We tell database that region_id columns in countries and regions table are identical and I linked them together, so when I ask any regions data from countries table, please provide me all information from there.

- select e.first_name, j.end_date from job_history j **join** employees e **on** j.employee_id=e.employee_id; We can swap the tables position in query as you see. Above and below codes returns the same data.
- select e.first_name, j.end_date from employees e **join** job_history j **on** j.employee_id=e.employee_id;

select l.city, d.department_name, c.country_name, c.region_id, r.region_name from locations l
join departments d **on** l.location_id=d.location_id
join countries c **on** l.country_id = c.country_id
join regions r **on** c.region_id = r.region_id; See below:
In this case we linked locations table to the departments table first, then locations table to the countries table, then linked countries table to the regions table. So, there is no limitation. If we cannot find a direct link between two table, we can link one by one and access any table we want. The only thing we have to do is just find the relations between two table (common columns) and then do the same for another one. That is the reason why Relational DataBase is the most popular type.

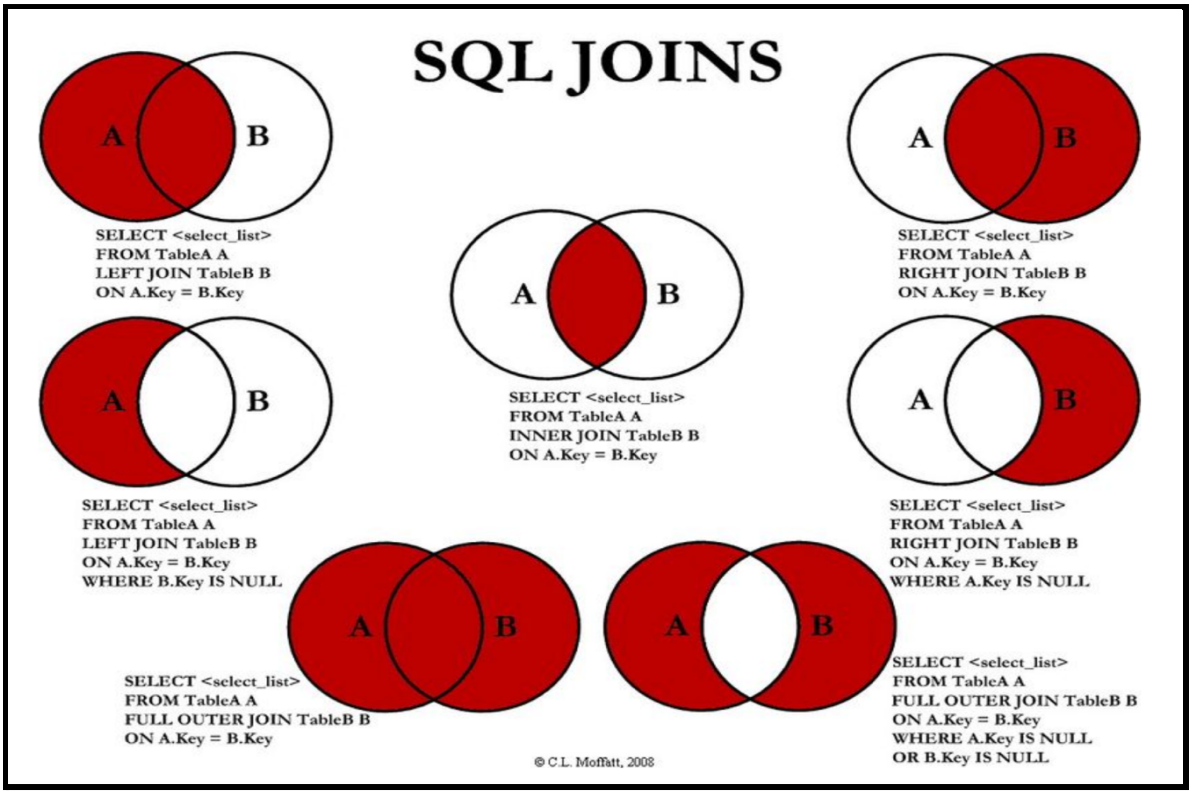
- select e.first_name, d.department_name, j.job_title from employees e
join departments d **on** e.department_id=d.department_id
join jobs j **on** e.job_id=j.job_id;

JOIN ON / GROUP BY / HAVING

- select **count(*)**, d.department_name from employees e
join departments d **on** e.department_id=d.department_id
group by d.department_name **having** count(*)>5 **order by** department_name;
Combines employees and department tables by using "join/on" keyword, then it return all department names and how many employee belong to that department if amount of employee is greater than 5 and orders in ascending.

select l.city as Location, count(d.department_name) from departments d
join locations l **on** d.location_id=l.location_id
group by l.city **having** city **like** 'S%' **and** count(d.department_name)>1;
it combines departments and locations then returns city and how many departments in this city if city starts with 'S' and has more than one department.

LEFT/RIGHT JOIN
Left Outer Join = Left Join
Right Outer Join = Right Join
Full Outer Join = Full Join



- Select c.country_name, r.region_name from countries c **left join** regions r **on** r.region_id=c.region_id; In this case countries is left table and

regions is right table. It will return all matching portion from countries and regions. Also, it will return unmatching part of the countries table.

- select d.department_id, d.manager_id, e.first_name **from** departments d **right join** employees e **on** d.manager_id=e.manager_id **where** d.manager_id **is null**;

select * from customer c **left join** address a **on** c.address_id = a.address_id **where** c.address_id **is null**;

select * from customer c **right join** address a **on** c.address_id = a.address_id **where** c.address_id **is null**;

FULL JOIN
select * from customer c **full join** address a **on** c.address_id = a.address_id;

select * from customer c **full join** address a **on** c.address_id = a.address_id where c.address_id **is null or** a.address_id **is null**;

SELF JOIN
Self join is used to make a comparison within the same table. Using aliases is mandatory.
• We use self join when we want to combine rows with other rows in the same table.
• To perform the self join operation, we must use a table alias to help SQL distinguish the left table from the right table of the same table.
• Let's say we want to print out employee full name with their manager's name together.
• SELECT e1.first_name, e1.last_name, e1.manager_id, e2.last_name FROM employees e1 **JOIN** employees e2 **ON e1.manager_id = e2.employee_id**;
- select emp.first_name as "Employee Name", mng.first_name as "Manager Name" from employees emp
join employees mng **on** emp.manager_id=mng.employee_id;

SUB (INNER) QUERY
* Sub query is a query inside another query…

- select first_name from employees
where salary =(select max(salary)from employees);
- select first_name, salary from employees where salary > (select avg(salary)from employees) order by salary desc; same as below
- select * from (select first_name, salary from employees
where salary > (select avg(salary)from employees)); same as above
- select first_name, department_id from employees where employee_id in (select manager_id from departments where manager_id is not null);
- select max(salary) from employees where salary<(select max(salary) from employees); ==> retrieves a single raw which has second highest salary in table.
- select max(salary) from employees where salary<(select max(salary) from employees where salary<(select max(salary) from employees)); ==> retrieves a single raw which has third highest salary in table.
- select salary from employees **order by** salary **desc limit 3**; ==> retrieves 3 highest salary rows from table. (limit only works with MySQL)

- select salary from employees order by salary desc limit 2-1,1; ==> retrieves second highest salary raw from table. (limit works with MySQL)
- select salary from employees order by salary desc limit 5-1,3; ==> retrieves 5th, 6th and 7th highest salary rows from table. (limit works with MySQL)

==> limit N-1,X --> start from Nth highest salary and retrieve totally X rows which is ordered by descending. (limit works with MySQL)

SET OPERATORS (UNION / UNION ALL / MINUS / INTERSECT)

Only applies to the multiple independent queries.
Queries must select same column names with same data type.

Union: it is used to combine two query results with same column name, same data type and return as a single combined result. It removes duplicates and sorts the table.
Union All: it is used to combine two tables with same column name, same data type and return as a single combined result without removing duplicates.
Minus: Compares two tables and removes the common data from first table.
Intersect: It returns the common part of two tables and removes the rest.
***** Note:** We can compare two tables whether those are identical if we use 'minus' or 'intersect' key words.

5,6,7,8,9,10 **union** 6,7,8	5,6,7,8,9,10
5,6,7,8,9,10 **union all** 6,7,8	5,6,7,8,9,10,6,7,8
5,6,7,8,9,10 **minus** 6,7,8	5,9,10
5,6,7,8,9,10 **intersect** 6,7,8	6,7,8

Examples:

Exp-1:

```
select first_name, salary from employees where salary >12000 Query-1
union all
select first_name, salary from employees where salary >9000; Query-2
```

For example; 18 rows come from query-1 and 25 come from query-2. Union all combines them and returns totally 18+25=43 rows.

Exp-2:

```
select first_name, salary from employees where salary >12000 Query-1
union
select first_name, salary from employees where salary >9000; Query-2
```

However; 18 rows come from table-1 and 25 come from Table-2. Union all combines them and returns totally 25 rows because Table-2 is totally contains Table-1 rows. So, it removes duplicates.

Exp-3:

```
select first_name, salary from employees where salary >9000 Query-2
minus
select first_name, salary from employees where salary >12000; Query-1
```

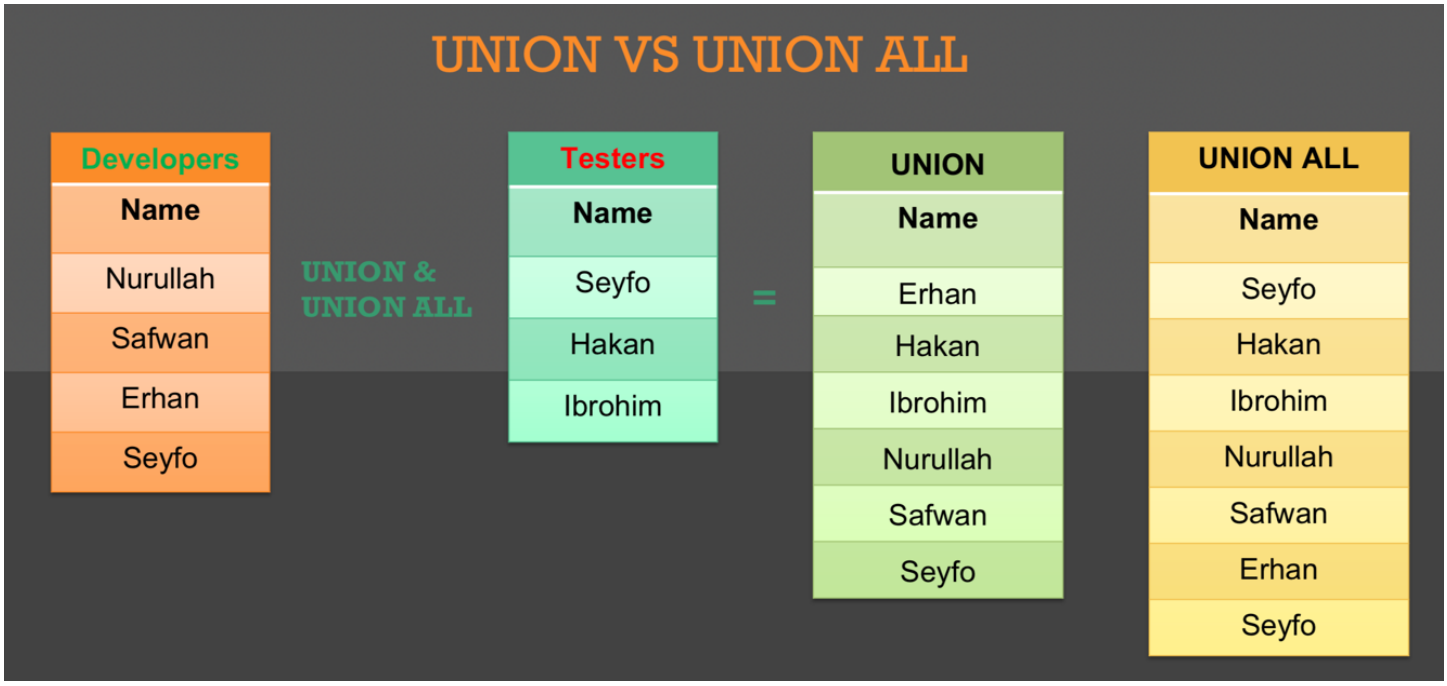
It will subtract common rows. In this case table-1 is completely contained in table-2. So result is 23-8=15 rows

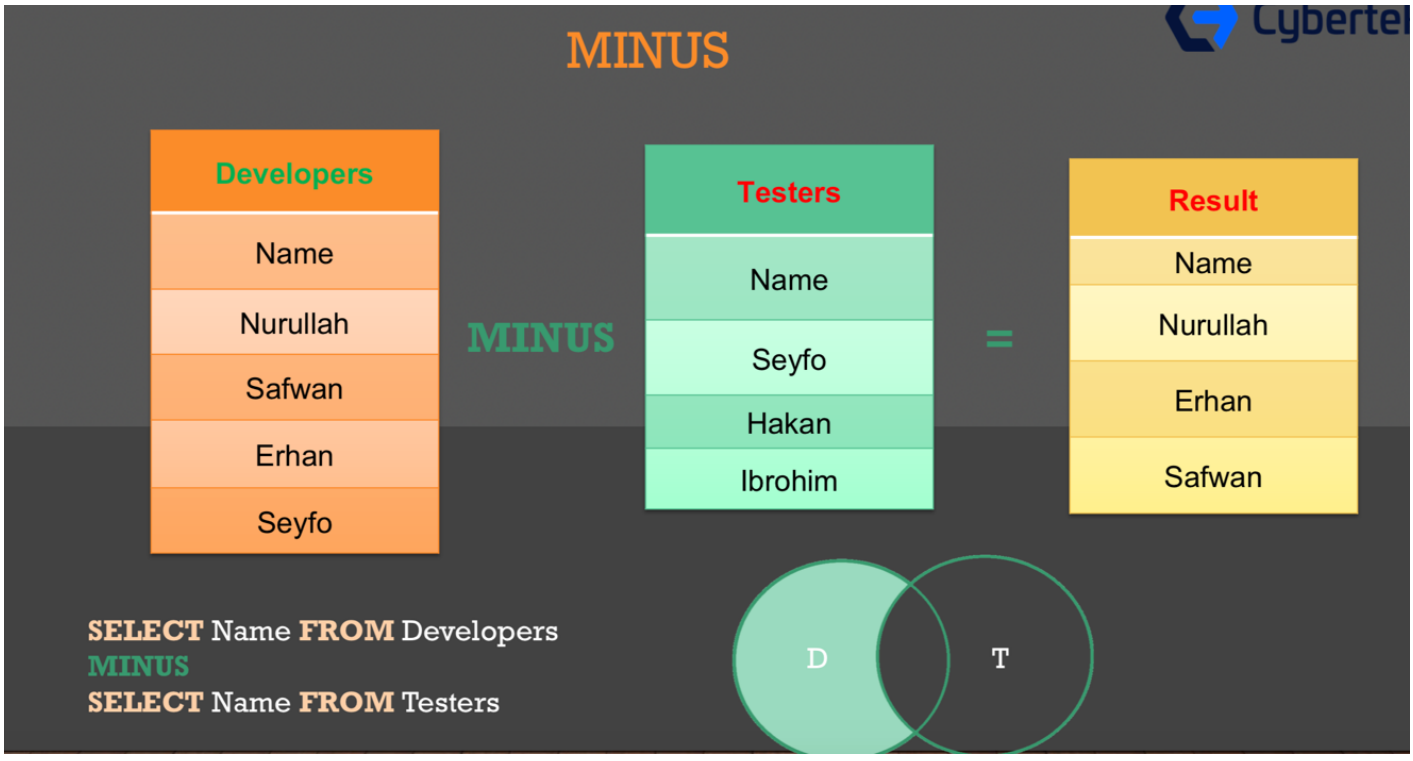
Exp-4:

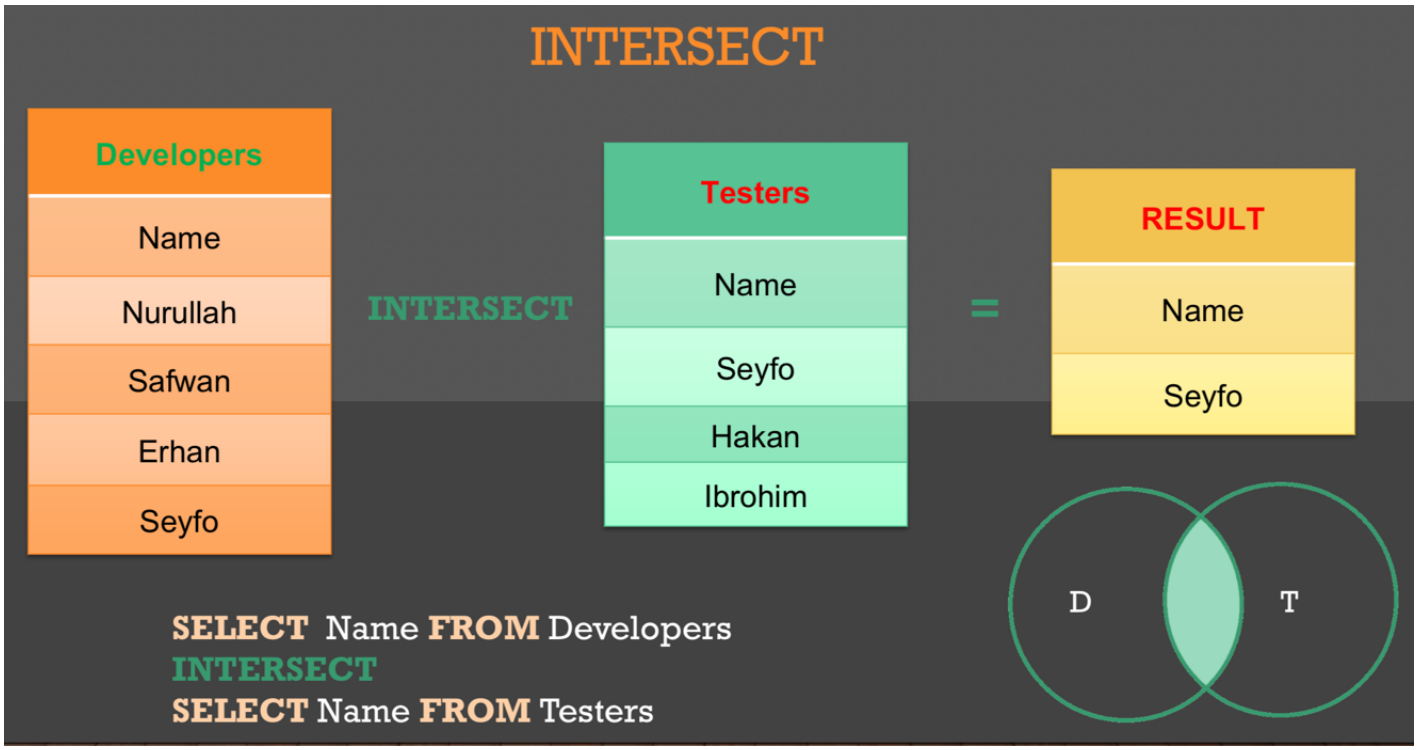
```
select first_name, salary from employees where salary >9000 Query-2
intersect
select first_name, salary from employees where salary >12000 Query-1;
```

It will return only common rows. In this case table-1 is completely common for both tables. So; result is 8 rows (Query-1)

NOTE: There is big difference between join and union. We don't have to have same column names in join, only one relational primary key is enough to link them together. However, we have to have exactly the same column names and data type for union and union all.







VIEWS

Views are created in order to store your queries as virtual table. If there are a lot of columns and we don't want to use all columns and make the table simpler, we can create a view and reuse it repeatedly. Actually, view does not store any data however, it contains the retrieve statements to provide reusability. We can create view if we use some queries mostly.

```
create view View_Name as select column_names from table_name where condition;

create view EmployeeInfo as select first_name || last_name as "Full Name"
from employees;
select * from EmployeeInfo;

create view CountryNames as select country_name from countries;
select * from CountryNames;

create view IT_Programmers as select * from employees where job_id =
'IT_PROG';

SELECT * from IT_Programmers;

drop view view_name;    deletes the view
drop view IT_Programmers;
```

create or replace view ViewName as … It tries to update the view, if there is no view as mentioned, it creates new in this case.
create or replace view EmployeeInfo **as** select first_name || ' ' || last_name as "Full Name", lower(email||'@gmail.com') as "Email" from employees;
create or replace view SDETs as select first_name||' '||last_name as FullName, lower(email||'@gmail') Email, '$'||salary Salary from employees where job_id='IT_PROG' order by salary desc;

CREATING DATA TABLE

SQL CONSTRAINS:

Not Null Constrain: The NOT NULL constraint enforces a column NOT to accept NULL values. This means that you cannot insert a new record, or update a record without adding a value to this field.

Primary Key Constrain: Each table should have at least a single column which uniquely identify the record (row) which is called PRIMARY_KEY, it is combination of **unique constrain** and **not null constrain**. If there is primary key in the table that specific column:
- must contain unique values,
- cannot contain NULL values,
- a table cannot have more than one Primary Key within the table, however primary key can contain either single or multiple fields.

Foreign Key: We can declare reference constrain like "OtherTableName(PK_Column)". We can identify any column as FK if it is declared as PK in another table.

Unique Constrain: Data must be unique throughout all the table.

Exp:
```
create table ScrumTeam(
 EmployeeID Integer primary key,
 FirstName varchar(30),
 LastName varchar(30) ,
 JobTitle varchar(20)
);
insert into ScrumTeam values(1, 'Tom', 'AAA', 'PO');
insert into ScrumTeam values(2, 'Jack', 'BBB', 'BA');
insert into ScrumTeam values(3, 'Betty', 'CCC', 'Scrum Master');
insert into ScrumTeam values(4, 'John', 'DDD', 'Developer');
insert into ScrumTeam values(5, 'Peter','EEE', 'SDET');
select * from scrumteam;
```

Exp:
```
create table Student
(
    ID int not null,
    FirstName varchar(255) not null,
    LastName varchar(255) not null,
    Age int,
    Primary Key (ID ,last_name));
```

```
insert into Student values(1, "Tom", "AAA", 15);
insert into Student values(1, "Jack", "BBB", 15);
insert into Student values(null, "Betty", "CCC", 16);
insert into Student values(4, "John", null, 17);
insert into Student values(5, "Peter", "EEE", 18);
select * from Student;
1|Tom|AAA|15
1|Jack|BBB|15
5|Peter|EEE|18
Error: near line 12: NOT NULL constraint failed: Student.ID
Error: near line 13: NOT NULL constraint failed: Student.LastName
```

Example for table creation:

```
create table Employee(

    EmpID varchar(255),
    EmpName varchar(255),
    Age int,
    PhoneNumber int,
    EmailID varchar(255),
    Address varchar(255),
    Salary int
);
insert into Employee values(1, "Tom", 20, 9876543, "aa@gmail.com", "Tysons,
USA", 70000);
insert into Employee values(2, "Tommy", 25, 123456, "bb@gmail.com",
"Oxford,UK",75000);
insert into Employee values(3, "Tomas", 30, 514315, "cc@gmail.com",
"Frankfurt,Germany",80000);
insert into Employee values(4, "Steve", 25, 856784, "dd@gmail.com",
"Rio,Brasil", 85000);
insert into Employee values(5, "Peter", 35, 456673, "ee@gmail.com",
"Milano,Italy", 90000);
insert into Employee values(6, "John", 25, 524355, "ff@gmail.com", "Sydney,
AUS", 95000);
insert into Employee values(7, "Cris", 40, 917363, "gg@gmail.com", "Berlin,
DEU", 100000);
insert into Employee values(8, "Yusuf", 25, 917363, "hh@gmail.com",
"Tysons,USA", 90000);
```

INSERTING ALL VALUES AT THE SAME TIME INTO MULTIPLE TABLE

```
create table Developers(
ID_Number integer primary key,
Names varchar(30),
Salary integer);

create table Testers(
ID_Number integer primary key,
Names varchar(30),
Salary integer);
insert all
    into Developers values(1, 'Nurullah', 155000)
```

```
    into Developers values(2, 'Safwan', 142000)
    into Developers values(3, 'Erhan', 85000)
    into Developers values(4, 'Seyfo', 120000)
    into Testers values(1, 'Seyfo', 110000)
    into Testers values(2, 'Hakan', 105000)
    into Testers values(3, 'Ibrohim', 100000);
```

UPDATING ROW

```
Update TableName set ColumnName = value where condition;
update scrumteam set firstname ='Muhtar' where EmployeeID='1';
update scrumteam set lastname ='Muhtar' where firstname='Tom';
```

DELETING ROW

```
delete from TableName where condition;
delete from scrumteam where firstname='Jack';
delete from scrumteam where JobTitle='SDET';
```

ALTER

It is used for modifying table structure.
Alter Commands:

- **add column** adds new column
- **drop column** deletes column
- **rename column** renames column name
- **rename to** renames table name

```
alter table Table_Name AlterCommand;
alter table scrumteam add salary integer;
alter table scrumteam rename column salary to Income;
alter table scrumteam drop column Income;
alter table scrumteam rename to AgileTeam;
```

- If you would like to **rename any PK** in table, you have to rename first same column which exists as FK in another table, then rename the column where this column exists as PK.
- If you would like to **drop**(delete) **any table** which has a **PK**, you cannot delete if this PK column exists as FK in another table.

DROP / TRUNCATE

- drop table table_name deletes all table including its columns.
- truncate table table_name cleans all the data from table but keeps the columns, features and table structure as it was.
- If you would like to **truncate any table** which has a **PK**, you cannot truncate if this PK column exists as FK in another table. You have to truncate first FK table and then you can truncate the PK table.

COMMIT

- Approves all the changes that have been made.

commit; Same as below
commit work; Same as above

TOP SQL INTERVIEW QUESTIONS

1. State the differences between HAVING and WHERE clauses.

Basis for Comparison	WHERE	HAVING
Implemented in	Row operations	Column operations
Applied to	A single row	The summarized row or groups
Used for	Fetching specific data from specific rows according to the given condition	Fetching the entire data and separating according to the given condition
Aggregate functions	Cannot have them	Can have them
Statements	Can be used with SELECT, UPDATE, and DELETE	Cannot be used without a SELECT statement
GROUP BY clause	Comes after the WHERE clause	Comes before the HAVING clause

2. **What is SQL?**
SQL stands for 'Structured Query Language' and is used for communicating with the databases. SQL is the standard query language for Relational Database Management Systems (RDBMS) that is used for maintaining them and also for performing different operations of data manipulation on different types of data. Basically, it is a database language that is used for the creation and deletion of databases, and it can be used to fetch and modify the rows of a table and also for multiple other things.

3. Explain the different types of SQL commands.

- **Data Definition Language**: DDL is that part of SQL which defines the data structure of the database in the initial stage when the database is about to be created. It is mainly used to create and restructure database objects. Commands in DDL are:
  - Create table
  - Alter table
  - Drop table

- **Data Manipulation Language:** DML is used to manipulate the already existing data in the database. It helps users to retrieve and manipulate the data. It is used to perform operations like inserting data into the database through the **insert** command, updating the data with the **update** command, and deleting the data from the database through the **delete** command.
- **Data Control Language:** DCL is used to control access to the data in the database. DCL commands are normally used to create objects related to user access and also to control the distribution of privileges among users. The commands that are used in DCL are **Grant** and **Revoke.**
- **Transaction Control Language:** It is used to control the changes made by DML commands. It also authorizes the statements to assemble in conjunction into logical transactions. The commands that are used in TCL are **Commit**, **Rollback**, **Savepoint**, **Begin**, and **Transaction.**

4. What is a default constraint?

Constraints are used to specify some sort of rules for processing data and limiting the type of data that can go into a table. Now, let's understand the default constraint.

Default constraint: It is used to define a default value for a column so that the default value will be added to all the new records if no other value is specified. For example, if we assign a default constraint for the emp_salary column in the below table and set the default value as 85000, then all the entries of this column will have a default value of 85000 unless no other value has been assigned during the insertion.

5. What is a unique constraint?

Unique constraints ensure that all the values in a column are different. For example, if we assign a unique constraint to the emp_name column in the below table, then every entry in this column should have a unique value.

6. **How would you find the second highest salary from the below table?**

```
select max(salary) from employee where salary not in (select max(salary)
from employee);
```

7. What is a Primary Key?

A primary key is used to uniquely identify all table records. It cannot have NULL values, and it must contain unique values. A table can have only one primary key that consists of single or multiple fields.

Now, we will write a query for demonstrating the use of a primary key for the Employee table:

```
CREATE TABLE Employee (
Employee_Id int NOT NULL,
Employee_name varchar(255) NOT NULL,
Employee_designation varchar(255),
Employee_Age int,
PRIMARY KEY (Employee_Id)
);
```

8. What is a Foreign Key?

A foreign key is an attribute or a set of attributes that references to the primary key of some other table. So, basically, it is used to link together two tables.
Let's create a foreign key for the below table:

```
CREATE TABLE Orders (
OrderID int NOT NULL,
OrderNumber int NOT NULL,
PersonID int,
PRIMARY KEY (OrderID),
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
)
```

9. What do you understand by a Character Manipulation function?
Character manipulation functions are used for the manipulation of character data types.
Some of the character manipulation functions are:

- **UPPER:** It returns the string in uppercase.
  - **Example:**

```
SELECT UPPER('demo string') from String;
```

  - **Output:**

```
DEMO STRING
```

- **LOWER:** It returns the string in lowercase.
  - **Example:**

```
SELECT LOWER ('DEMO STRING') from String
```

  - **Output:**

```
demo string
```

- **INITCAP:** It converts the first letter of the string to uppercase and retains others in lowercase.
  - **Example:**

```
SELECT Initcap('dATASET') from String
```

  - **Output:**

```
Dataset
```

- **CONCAT:** It is used to concatenate two strings.
  - **Example:**

```
SELECT CONCAT('Data','Science') from String
```

  - **Output:**

```
Data Science
```

- **LENGTH:** It is used to get the length of a string.
  - **Example:**

- SELECT LENGTH('Hello World') from String
    - **Output:**
- 11

10. What is AUTO_INCREMENT?

AUTO_INCREMENT is used in SQL to automatically generate a unique number whenever a new record is inserted into a table. Since the primary key is unique for each record, we add this primary field as the AUTO_INCREMENT field so that it is incremented when a new record is inserted. The AUTO-INCREMENT value is by default starts from 1 and incremented by 1 whenever a new record is inserted.

Syntax:

```
CREATE TABLE Employee(
Employee_id int NOT NULL AUTO-INCREMENT,
Employee_name varchar(255) NOT NULL,
Employee_designation varchar(255)
Age int,
PRIMARY KEY (Employee_id)
)
```

11. What is the difference between DELETE and TRUNCATE commands?

- **DELETE:** This query is used to delete or remove one or more existing row.
- **TRUNCATE:** This statement deletes all the data from inside a table.

The difference between DELETE and TRUNCATE commands are as follows:

- TRUNCATE is a DDL command, and DELETE is a DML command.
- With TRUNCATE, we can't really execute and trigger, while with DELETE we can accomplish a trigger.
- If a table is referenced by foreign key constraints, then TRUNCATE won't work. So, if we have a foreign key, then we have to use the DELETE command.

12. What is wrong with the below-given SQL query?

```
SELECT gender, AVG(age) FROM employee WHERE AVG(age)>30 GROUP BY gender
```

Aggregation may not appear in the WHERE clause unless it is in a subquery contained in a HAVING clause or a select list, the column being aggregated is an outer reference.
This basically means that whenever we are working with aggregate functions and we are using GROUP BY, we can't use the WHERE clause. Therefore, instead of the WHERE clause, we should use the HAVING clause.
Also, when we are using the HAVING clause, GROUP BY should come first and HAVING should come next.
Right query should be as below:

```
select gender, avg(age) from employee group by gender having avg(age)>30
```

13. What are Views? Give an example.
Views are virtual tables used to limit the tables that we want to display, and these are nothing but the result of a SQL statement that has a name associated with it. Since views are not physically present, they take less space to store.

Let's consider an example. In the below employee table, say, we want to perform multiple operations on the records with gender 'Female'. We can create a view-only table for the female employees from the entire employee table.

Now, let's implement it on the SQL server.

Below is our employee table:

```
select * from employee;
```

	e_id	e_name	e_salary	e_age	e_gender	e_dept
1	1	Sam	95000	45	Male	Operations
2	2	Bob	80000	21	Male	Support
3	3	Anne	125000	25	Female	Analytics
4	4	Julia	112000	30	Female	Analytics
5	5	Matt	159000	33	Male	Sales
6	6	Jeff	112000	27	Male	Operations

Now, we will write the syntax for view.

Syntax:

```
create view female_employee as select * from employee where
e_gender='Female';
select * from female_employee;
```

14. What do you know about Joins? Define different types of Joins.
The Join clause is used to combine rows from two or more tables based on a related column between them. There are various types of Joins that can be used to retrieve data, and it depends upon the relationship between tables.

There are four types of Joins:

- **Inner Join:** Inner Join basically returns records that have matching values in both tables.
- **Left Join:** Left Join returns rows that are common between the tables and all the rows of the left-hand-side table, i.e., it returns all the rows from the left-hand-side table even if there are no matches available in the right-hand-side table.
- **Right Join:** Right Join returns rows that are common between the tables and all the rows of the right-hand-side table, i.e., it returns all the rows from the right-hand-side table even if there are no matches available in the left-hand-side table.
- **Full Join:** Full Join returns all the rows from the left-hand-side table and all the rows from the right-hand-side table.

15. What are differences between Views and Tables.

Views	Tables
It is a virtual table that is extracted from a database.	A table is structured with a set number of columns and a boundless number of rows.
Views do not hold data themselves.	Table contains data and stores the data in databases.
A view is also utilized to query certain information contained in a few distinct tables.	A table holds fundamental client information and the cases of a characterized object.
In a view, we will get frequently queried information.	In a table, changing the information in the database changes the information that appears in the view

16. What do you understand by Self Join?
Self Join is used for joining a table with itself. Here, depending upon some conditions, each row of the table is joined with itself and with other rows of the table.

17. What is the difference between Union and Union All operators?
The **Union** operator is used to combine the result set of two or more select statements. For example, the first select statement returns the fish shown in Image A and the second returns the fish shown in Image B. Then, the Union operator will return the result of the two select statements as shown in Image A U B. Also, if there is a record present in both tables, then we will get only one of them in the final result.

18. What is the use of the Intersect operator?
The **Intersect** operator helps combine two select statements and returns only those records that are common to both the select statements. So, after we get Table A and Table B over here and if we apply the Intersect operator on these two tables, then we will get only those records that are common to the result of the select statements of these two.

19. How can you copy data from one table into another?
Here, we have our employee table. We have to copy this data into another table. For this purpose, we can use the INSERT INTO SELECT operator. Before we go ahead and do that, we would have to create another table that would have the same structure as the above-given table.
First create the second table with the same table structure with copied one. Then use the syntax below:

employee_duplicate New table employee First table

```
insert into employee_duplicate select * from employees
```

20. How many Aggregate functions are available in SQL?

SQL Aggregate functions determine and calculate values from multiple columns in a table and return a single value.

There are 7 aggregate functions in SQL:

- **COUNT():** Returns number of table rows.
- **MAX():** Returns the largest value among the records.
- **MIN():** Returns smallest value among the records.
- **AVG():** Returns the average value from specified columns.
- **SUM():** Returns the sum of specified column values.
- **FIRST():** Returns the first value.
- **LAST():** Returns last value.

21. What do you mean by Subquery?

Query within another query is called as Subquery. A subquery is called inner query which returns output that is to be used by another query.

22. How many row comparison operators are used while working with a subquery?

There are 3-row comparison operators that are used in subqueries which are IN, ANY and ALL.

23. What is the difference between DELETE and TRUNCATE?

The differences are:

- DELETE command is DML command and the TRUNCATE command is DDL.
- DELETE command is used to delete a specific row from the table whereas the TRUNCATE command is used to remove all rows from the table.
- We can use the DELETE command with WHERE clause but cannot use with TRUNCATE command.

24. What is the difference between DROP and TRUNCATE?

TRUNCATE removes all data from the table which cannot be retrieved back but it keeps the structure of the table, DROP removes the entire table from the database and it also cannot be retrieved back.

25. What is a Relationship? How many types of Relationships are there?

The relationship can be defined as the connection between more than one table in the database.

There are 4 types of relationships:

- One to One Relationship
- Many to One Relationship
- Many to Many Relationship
- One to Many Relationship

26. What are some properties of Relational databases?

Properties are as follows:

- In relational databases, each column should have a unique name.
- The sequence of rows and columns in relational databases is insignificant.
- All values are atomic and each row is unique.

27. What do we need to check in Database Testing?

In Database testing, the following thing is required to be tested:

- Database connectivity
- Constraint check
- Required application field and its size

- Data Retrieval and processing with DML operations
- Stored Procedures
- Functional flow

28. What do you mean by Stored Procedures? How do we use it?
A stored procedure is a collection of SQL statements that can be used as a function to access the database. We can create these stored procedures earlier before using it and can execute them wherever required by applying some conditional logic to it. Stored procedures are also used to reduce network traffic and improve performance.

29. What is Database White Box Testing?
Database White Box testing involves:
- Database Consistency and ACID properties
- Database triggers and logical views
- Decision Coverage, Condition Coverage, and Statement Coverage
- Database Tables, Data Model, and Database Schema
- Referential integrity rules

30. What is Database Black Box Testing?
Database Black Box testing involves:
- Data Mapping
- Data stored and retrieved
- Use of Black Box testing techniques such as Equivalence Partitioning and Boundary Value Analysis (BVA)

31. How do you add a column to a table?
To add another column in the table, use the following command:
```
ALTER TABLE table_name ADD column_name varchar(50);
```

32. What is COMMIT?
COMMIT saves all changes made by DML statements. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

33. What is the Primary key?
A Primary key is a column whose values uniquely identify each row in a table. Primary key values must be unique and cannot be null.

34. What are Foreign keys?
When a table's primary key field is added to related tables in order to create the common field which relates the two tables, it called a foreign key in other tables. Foreign key constraints enforce referential integrity.

35. What is CHECK Constraint?
A CHECK constraint is used to specify a condition that each row in the table must satisfy. To satisfy the constraint, each row in the table must make the condition either TRUE or unknown (due to a null). They are used to enforce domain integrity.

36. How to select random rows from a table?
Using a SAMPLE clause, we can select random rows.
For Example,
```
SELECT * FROM table_name SAMPLE(10);
```

37. Give the order of SQL SELECT?

Order of SQL SELECT clauses is:

- SELECT
- FROM
- WHERE
- GROUP BY
- HAVING
- ORDER BY
- Only the SELECT and FROM clauses are mandatory

38. How to write SQL comments?

SQL comments can be inserted by adding two consecutive hyphens (--).

39. Difference between TRUNCATE, DELETE and DROP commands?

- **DELETE** removes some or all rows from a table based on the condition. It can be rolled back.
- **TRUNCATE** removes ALL rows from a table by de-allocating the memory pages. The operation cannot be rolled back
- **DROP** command removes a table from the database completely.

40. What is UNION, UNION ALL, MINUS, INTERSECT?

- **UNION** – returns all distinct rows selected by either query
- **UNION ALL** – returns all rows selected by either query, including all duplicates.
- **MINUS** – returns all distinct rows selected by the first query but not by the second.
- **INTERSECT** – returns all distinct rows selected by both queries.

41. What is the difference between UNIQUE and PRIMARY KEY constraints?

The differences are as follows:

- A table can have only one PRIMARY KEY whereas there can be any number of UNIQUE constrain.
- The primary key cannot contain null values whereas the Unique constrain can contain null values.

42. What is a composite primary key?

The primary key created on more than one column is called composite primary key.

43. What is an Index?

An Index is a special structure associated with a table to speed up the performance of queries. The index can be created on one or more columns of a table.

44. How can we avoid duplicating records in a query?

By using the DISTINCT keyword, duplication of records in a query can be avoided.

45. Explain the difference between Rename and Alias?

Rename is a permanent name given to a table or column whereas Alias is a temporary name given to a table or column.

```
alter table table_name rename column old_name to new_name;
alter table table_name rename to new_name;
```

- If you would like to **rename any PK** in table, you have to rename first same column which exists as FK in another table, then rename the column where this column exists as PK.

46. What are the advantages of Views?
Advantages of Views are:
- Views restrict access to the data because the view can display selective columns from the table.
- Views can be used to make simple queries to retrieve the results of complicated queries. For Example, views can be used to query information from multiple tables without the user knowing.

47. Does View contain Data?
No, Views are virtual structures.

48. Can a View be created based on another View?
Yes, A View can be created based on another View.

49. What is the difference between the HAVING clause and WHERE clause?
Both specify a search condition but Having clause is used only with the SELECT statement and typically used with GROUP BY clause.
If GROUP BY clause is not used then HAVING behaved like WHERE clause only.

50. What is Oracle and what are its different editions?
Oracle is one of the popular databases provided by Oracle Corporation, which works on relational management concepts and hence it is referred to as Oracle RDBMS as well. It is widely used for online transaction processing, data warehousing, and enterprise grid computing.

51. How will you differentiate between VARCHAR & VARCHAR2?
Both VARCHAR & VARCHAR2 are Oracle data types that are used to store character strings of variable length. Their differences are:
- VARCHAR can store characters up to 2000 bytes while VARCHAR2 can store up to 4000 bytes.
- VARCHAR will hold the space for characters defined during declaration even if all of them are not used whereas VARCHAR2 will release the unused space.

52. How can we find out the duplicate values in an Oracle table?
We can use the below example query to fetch the duplicate records.
```
SELECT FIRST_NAME FROM EMP GROUP BY FIRST_NAME HAVING COUNT(EMP_NAME)>1;
```

53. How can we find out the current date and time in Oracle?
```
SELECT CURRENT_DATE from dual;
```

54. When do we use the GROUP BY clause in SQL Query?
GROUP BY clause is used to identify and group the data by one or more columns in the query results. This clause is often used with aggregate functions like COUNT, MAX, MIN, SUM, AVG, etc.

55. What do you understand by a database object? Can you list a few of them?
Object used to store the data or references of the data in a database is known as a database object. The database consists of various types of DB

objects such as tables, views, indexes, constraints, stored procedures, triggers, etc.

56. Can we save images in a database and if yes, how?
BLOB stands for Binary Large Object, which is a data type that is generally used to hold images, audio & video files or some binary executables. This datatype has the capacity of holding data up to 4 GB.

57. What do you understand by database schema and what does it hold?
Schema is a collection of database objects owned by a database user who can create or manipulate new objects within this schema. The schema can contain any DB objects like table, view, indexes, clusters, stored procs, functions, etc.

58. What is a View and how is it different from a table?
View is a user-defined database object that is used to store the results of an SQL query, which can be referenced later. Views do not store this data physically but as a virtual table, hence it can be referred to as a logical table.
View is different from the table as:

- A table can hold data but not SQL query results whereas View can save the query results, which can be used in another SQL query as a whole.
- The table can be updated or deleted while Views cannot be done so.

59. What is meant by a deadlock situation?
Deadlock is a situation when two or more users are simultaneously waiting for the data, which is locked by each other. Hence it results in all blocked user sessions.

60. What is meant by an index?
An index is a schema object, which is created to search the data efficiently within the table. Indexes are usually created on certain columns of the table, which are accessed the most. Indexes can be clustered or non-clustered.

61. Find Max Salary from each department

```
select department_id, max(salary) from employees
group by department_id;
```

62. Find how many months has each employee been working since hired?
months_between function is used first parameter is sysdate it will generate the months between system date and time of hire.

```
select first_name, last_name, hire_date,
round(months_between(sysdate, hire_date)) as months from employees;
```

63. How to print duplicate row in a table?

```
select first_name,manager_id from employees
group by first_name, manager_id having count (*)>1;
```

63. What does 'NUMBER(8,2)' mean when we are creating table?
It means that you can use maximum 6 digits before decimal and 2 digits after decimal. For example:
NUMBER(8,2) 123456,78
NUMBER(6,0) 654321