FAST TRACK SQL

SQL

▼ Pre-Course Materials

- Server Informations
- **Create Project From Intellij**
- How to get PUBLIC IP from EC2 machine ?
- HR Schema

Pre - Conditions and Materials

1. **We will use EC2 instance(JENKINS SERVER) from AWS that you already created it from AMI**
   a. Jenkins → 8081
   b. Oracle Database → 1521
   c. Spartan App. → 8000
   d. Spartan with Auth. → 7000
   e. HR ORDS → 1000

   NOTE

   —> If you comfortable to create AWS EC2 Instance by following video from LMS , you can create with new email address to use as `free tier user.`

   —> These IPs `will not be available 15 days after SDET FAST TRACK`

   - **IP 1 —>** 54.158.207.205

2. **Create Project From Intellij**

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/a074b626-576e-4e96-a9cb-fb8d943141dd/Project_Creation.mp4

NOTE —> If you don't have ultimate edition we are gonna add `Plugin -> Database Navigator` to use with `Community Version`

- **Database Navigator Plugin**

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/5799fe95-86df-46cd-bfe2-4269fcba06ac/DB_Navigator_01.mp4

- **Adding SQL Sheet to run queries**

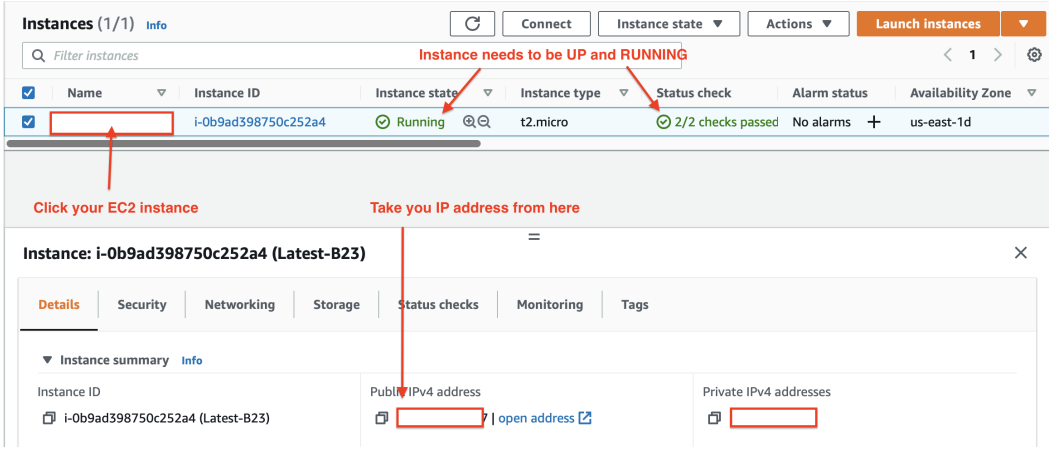
https://s3-us-west-2.amazonaws.com/secure.notion-static.com/17a8a196-202e-4559-a583-f8ced04878f5/CreateSQLFile.mp4

- **DBeaver** —> Watch Installation video from **LMS**
- SQL Developer

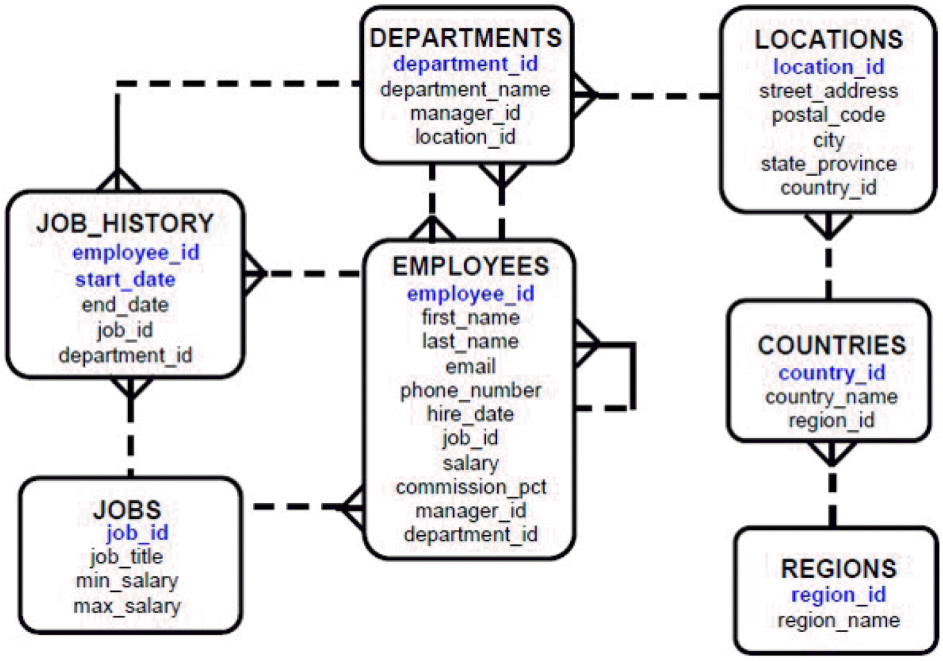
`NOTE -> Just use which ever you comfortable to do practice by yourself`

3. **Getting IP From EC2**

⚠️ You will need `PUBLIC IP` address from `EC2` machine to connect `ORACLE Database`



HR SCHEMA



DAY 01

- SELECT
- ORDER BY
- GROUP BY
- SINGLE ROW FUNC
- ROWNUM
- SELF JOIN
- DISTINCT
- LIKE
- HAVING
- VIEW
- DDL&DML
- SET OPERATORS
- WHERE
- AGGREGATE FUNCTIONS
- AS
- SUBQUERY
- JOINS

Database

1. What is Database ?
    - Database is a systematic collection of data.
    - Databases support storage and manipulation of data.
    - Databases make data management easy
    - In **databases** we store **data** in an organized manner

Relational Database

- **PK Primary Key**

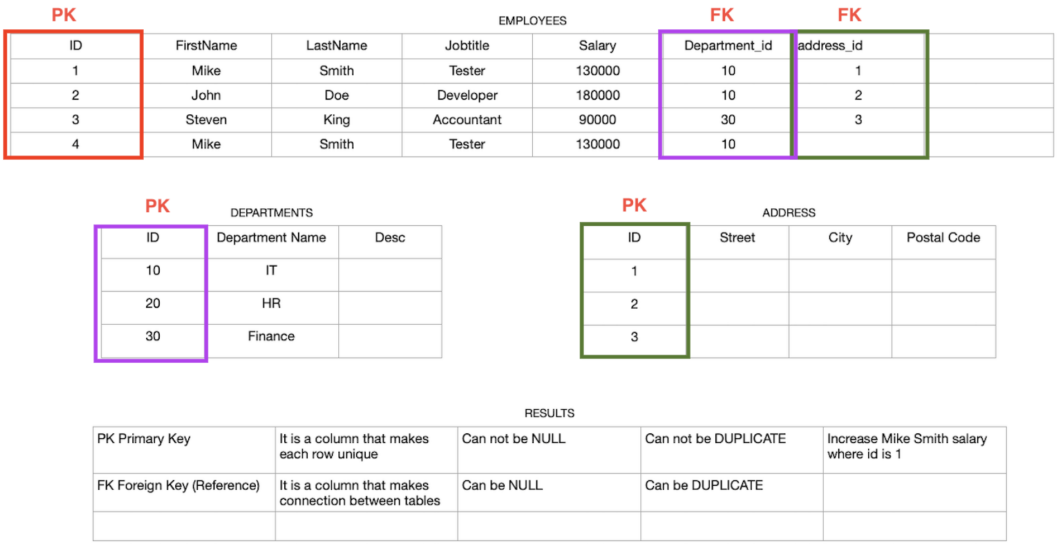
  Each table should have at least a single column which uniquely identify the record (row) which is called **PRIMARY_KEY.**

  - Can not be DUP,
  - Can not be NULL,
  - Primary key can contain one or more than one row (composite key)

- **FK Foreign Key**

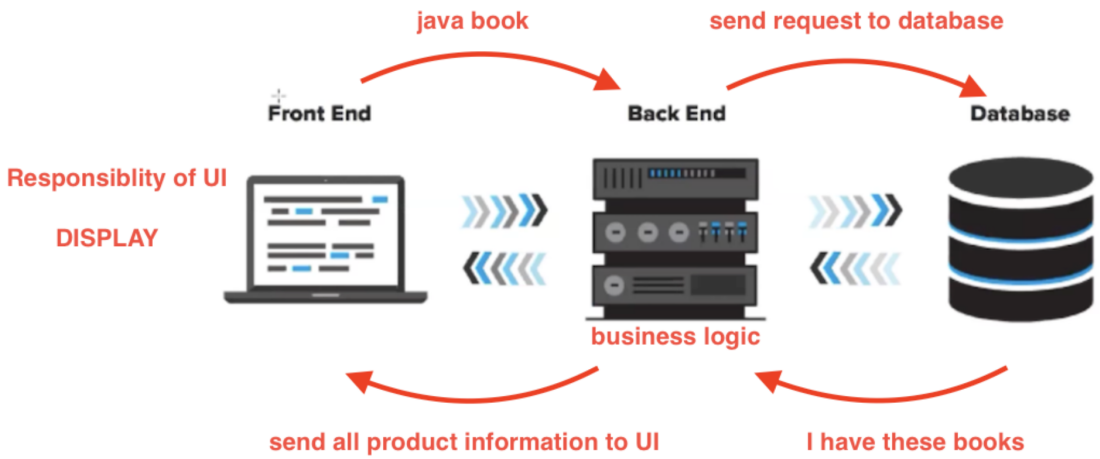
  It is a column that makes connection for two table.If we use one of the PK for another table connection we can call it as FK

  - Can be NULL
  - Can be DUP

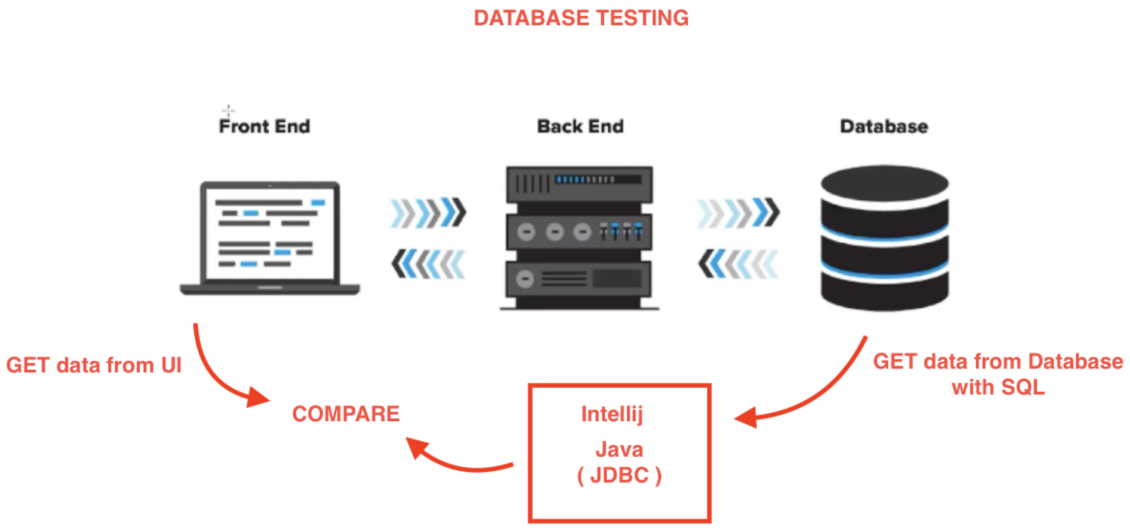


⚠️ What is `PK Primary Key` and `FK Foreign Key ?`

- Assume that we are trying to find Java book in amazon.com.When you type java book in search area and hit the enter or click the search icon we get all data in 1 second(Over 6000 results).
- All necessary data comes from Database with the help of business layer (Backend) for UI



- To do database testing we need to get data from source of data (database), then we need to compare these data vs UI information.Also we can do some database itself test too.This is the flow for database testing



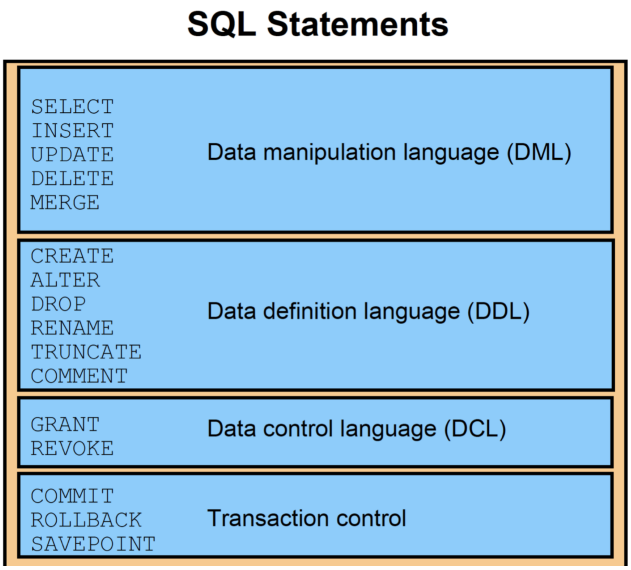
Non Relational Database

- The non-relational database, or NoSQL(`Not only SQL` —> They may use SQL and other Query Languages.For example Cassandra is one of the NOSQL database and it uses CQL (Cassandra Query Language)) database, stores data. However, unlike the relational database, there are no tables, rows, primary keys or foreign keys. Instead, the non-relational database uses a storage model optimized for specific requirements of the type of data being stored.
- Article

SQL —> Structured Query Language

1. What is SQL ?
    - SQL stands for 'Structured Query Language' and is used for communicating with the databases.

- 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.
- Used to manage the database
- SQL Statements



2. What is Query ?
    - A set of instructions
    - Communicating with Database with SQL

```
select * from employess ;

-- it will retrieve all employees information from employees table
```

SELECT

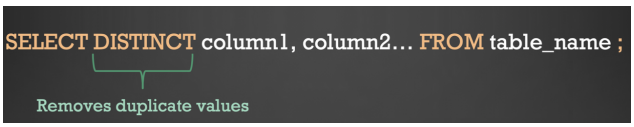
- SQL language is case **INSENSITIVE**
- If we want to query data from all column, we can use an **asterisk (*)** as the shorthand for all columns.
- Each SQL Statements end with **semi-colon**


```
/*
   MULTI LINE COMMENTS
*/

--- SELECT ---
select * from employees;
-- Single line comments
```

```
-- SQL language is NOT CASE SENSITIVE
-- Tables/columns may or not

select * FROM eMpLoYeeS;

-- display all information from countries table
select * from countries;

-- display firstname from employees table
select first_name from employees;

select FIRST_NAME from employees;

-- display firstname,lastname and salary from employees table
select first_name,last_name,salary  from employees;
```

DISTINCT

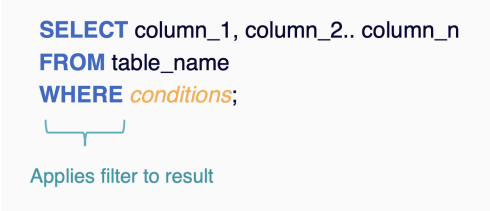
- Remove Duplicates




```
--- DISTINCT ---

/*
 it show different values from query result based provided column
 Original data still stays in database
 */

select FIRST_NAME from EMPLOYEES;
-- it returns  107

-- display all different names from employees
select distinct FIRST_NAME from EMPLOYEES;
-- it return 91 result as a different names

--display different department_ids from employees table
select distinct DEPARTMENT_ID from EMPLOYEES;



-- IQ -->  What is Distinct keyword in sql ?
```

WHERE

- The WHERE clause appears right after the FROM clause of the SELECT statement.
- **The conditions are used to filter the rows** returned from the SELECT statement.



SELECT WHERE Statement

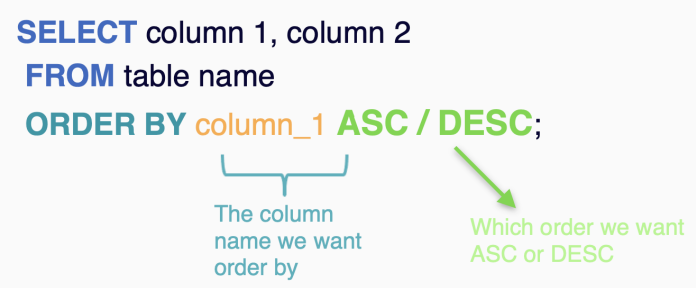
OPERATOR	DESCRIPTION
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<> or !=	Not equal
AND	Logical operator AND
OR	Logical operator OR

```
--- WHERE ---

/*
 -- Which keyword we are using in JAVA to filter ?
    -- if statement


 -- To filter individual results from database while we are running query
     we are gonna use WHERE keyword
 */

select * from EMPLOYEES
where FIRST_NAME='David';

--> '' it is case SENSITIVE

-- display firstname,lastname,salary where firstname is Peter
select first_name,last_name,salary from employees
where first_name='Peter';

-- display firstname,lastname,salary where firstname is David and last_name is Austin
select first_name,last_name,salary from EMPLOYEES
where first_name='David' and last_name='Austin';

-- display all information from employees where salary is bigger than 3000
select * from EMPLOYEES
where SALARY>3000;

-- display all info from employees  who is making more than 3000 and department id is 10 ;
select * from EMPLOYEES
where SALARY>3000 and DEPARTMENT_ID=10;

-- display all info from employees where salary  equals or more than 5000  and salary  equals or less than 8000
select * from EMPLOYEES
where SALARY>=5000 and SALARY<=8000;

-- BETWEEN lower and upper --> boundries are INCLUDED
-- it will give result within this range
select * from EMPLOYEES
where SALARY between 5000 and 8000;

-- display all info from employees where employee_id between 100 and 120
select * from EMPLOYEES
where EMPLOYEE_ID between 100 and 120;  --> range is included

-- OR --
-- display all info from employees who is working as IT_PROG or MK_MAN or SA_REP
-- solution 1
select * from EMPLOYEES
where JOB_ID='IT_PROG' or JOB_ID='MK_MAN' or JOB_ID='SA_REP';
```

```
-- IN CLAUSES --
-- it uses or logic
-- it checks jobIDs from IN paranthesis to see condition is matching
-- solution 2
select * from EMPLOYEES
where JOB_ID IN('IT_PROG','MK_MAN','SA_REP');


-- display all employees name except who is working as IT_PROG or MK_MAN or SA_REP
select FIRST_NAME,JOB_ID from EMPLOYEES
where JOB_ID NOT IN('IT_PROG','MK_MAN','SA_REP');
-- 107 --> 36 of thm working as IT_PROG / MK_MAN / SA_REP
-- 71 employees

-- IS NULL
-- display all information from employees where manager id is null
select * from EMPLOYEES
where MANAGER_ID is null;
--> 1 result it is Steven King

-- IS NOT NULL
-- display all information from employees where manager id is not null
select * from EMPLOYEES
where MANAGER_ID is not null ;
--> 106 employees has manager
```

ORDER BY

- The ORDER BY clause allows you to sort the rows returned from the SELECT statement in ascending or descending order based on criteria specified
- If we do not use ASC or DESC, **it will automatically order by ASC as DEFAULT**.

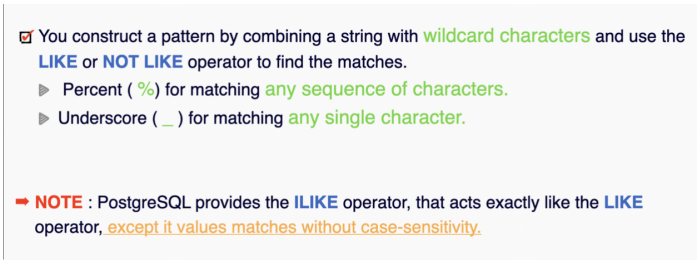



```
/*
 ORDERBY

    - It allows us to sort data based on provided column
    - AS A DEFAULT it will order the result ASCENDING ORDER (A-Z,0-9)
    - If you wanna sort it DESCENDING order we need specify right after column name

 */
-- display all employees based salary in asc order
select * from EMPLOYEES
order by SALARY;

select * from EMPLOYEES
order by SALARY asc;
-- NOTE :BOTH query will give same result

-- display all employees based salary in desc order (9-0 ) ( high to low )
select * from EMPLOYEES
order by salary desc ;

-- display all employees based firstname in desc order (Z-A)
select * from EMPLOYEES
order by FIRST_NAME desc ;
```

```
-- display all information from employees
-- where job ids are IT_PROG and order them based on salary asc order
select * from EMPLOYEES
where JOB_ID='IT_PROG'
order by salary ;

-- display all information from employees and order them based firstname asc
select * from EMPLOYEES
order by FIRST_NAME  ;

--IQ -->  display all information from employees and order them based firstname asc and based on lastname desc
select * from  EMPLOYEES
order by FIRST_NAME,LAST_NAME DESC;

/*
 it will first order result based on firstname
 after getting result if we have same names it will order them based lastname in desc order

 David Austin
 David Berstein
 David Lee

 --After execution
 David Lee
 David Berstein
 David Austin

 */
```

LIKE

- Use it for partial search
- Percent (%) for matching **any sequence of characters. (0 or more)**
- Underscore (_) for matching **any single character.**




```
/*


    - Percent ( % ) for matching any sequence of characters. (0 or more )
    - Underscore ( _ ) for matching any single character.

    - contains
    - startswith
    - endswith

 */

  -- display all employees where first_name STARTSWITH A
select * from EMPLOYEES
where first_name like 'A%';


-- display all employees where first_name STARTSWITH A and length of it 4 letter
select * from EMPLOYEES
where first_name like 'A___';


-- display 5 letter first names from employees ENDSWITH  m
select * from EMPLOYEES
```

```
where FIRST_NAME like '____m';

-- it gives all firstnames ENDSWITH  m
select * from EMPLOYEES
where FIRST_NAME like '%m';

-- display firstnames where second letter is a
select first_name from EMPLOYEES
where FIRST_NAME like '_a%';

-- display all employees where job id CONTAINS IT
select * from EMPLOYEES
where JOB_ID like '%IT%';


--> IQ --> What are the wilcard characters in SQL
    --> % --> it refers any sequence of characters. (0 or more )
    --> _ --> it refers any single char
```

AGGREGATE FUNCTIONS - MULTI ROW FUNCTIONS

- Aggregate functions **ignore null values except count(*)**

COUNT

- The COUNT function returns the number of input rows that match a specific condition of a query.
- it **does not consider NULL values in the column.**

MAX

MIN

SUM

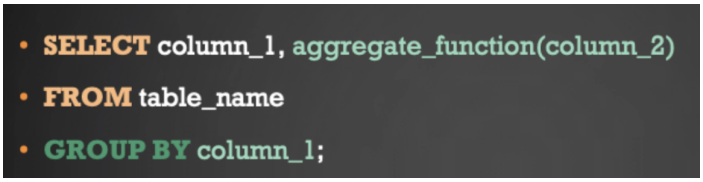
AVG

```
select * from employees;

/*
 AGGREATE FUNCTIONS - MULTI ROWS - GROUP FUNCTIONS

- count  --> it will count rows
- max    --> it will max value
- min    --> it will min value
- sum    --> it will give total value
- avg    --> it will give average values

 Aggreate functions takes multi row and return one result

 NOTE --> All aggreate functions will ignore NULL values
 */

 -- How many departments we have ?
select * from departments;
select count(*) from departments;

-- How many locations we have ?
select * from locations;
select count(*) from LOCATIONS;


-- NULL VALUES
select * from employees;

-- count(*) will count all rows after execution that why we have result as 1
```

```
select count(*) from EMPLOYEES
where DEPARTMENT_ID is null;   --She is Kimberly

-- count(DEPARTMENT_ID) will count only department IDs  after execution we have only one department ID which null
-- Since all aggreagate functions ignore NULL values it will give result as 0
select count(DEPARTMENT_ID) from EMPLOYEES
where DEPARTMENT_ID is null;   --She is Kimberly



-- BOTH EXECUTION will give same result since we dont have any null values for department ID
select count(DEPARTMENT_ID) from EMPLOYEES
where DEPARTMENT_ID is  not null;

select count(*) from EMPLOYEES
where DEPARTMENT_ID is  not null;


-- how many different firstname we have ?
select count(distinct FIRST_NAME) from EMPLOYEES;

-- how many employees working as IT_PROG or SA_REP
select count(*) from EMPLOYEES
where JOB_ID in ('IT_PROG','SA_REP');


-- how many employees getting salary more than 6000
select count(*) from EMPLOYEES
where SALARY>6000;

-- MAX--
select FIRST_NAME,salary from EMPLOYEES;
select max(salary) from EMPLOYEES;


-- MIN--
select min(salary) from EMPLOYEES;

-- AVG--
select avg(salary) from EMPLOYEES;

select round(avg(salary))   from EMPLOYEES;   // 6462
select round(avg(salary),1) from EMPLOYEES;   // 6462.1
select round(avg(salary),2) from EMPLOYEES;   // 6462.36
select round(avg(salary),3) from EMPLOYEES;  //  6462.364


-- SUM --
select sum(salary) from EMPLOYEES;
```

GROUP BY

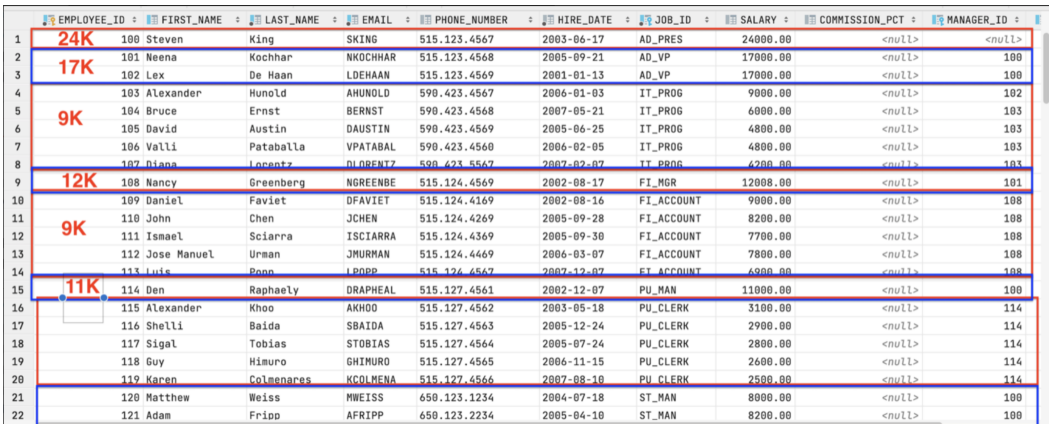
- The group by clause divides the rows retrieved from select statement into group based on the row we provide
- Group by can only be used along with group / aggregate / multi row functions.
- It groups all same data into one row



⚠️ Possible Interview Questions Topic

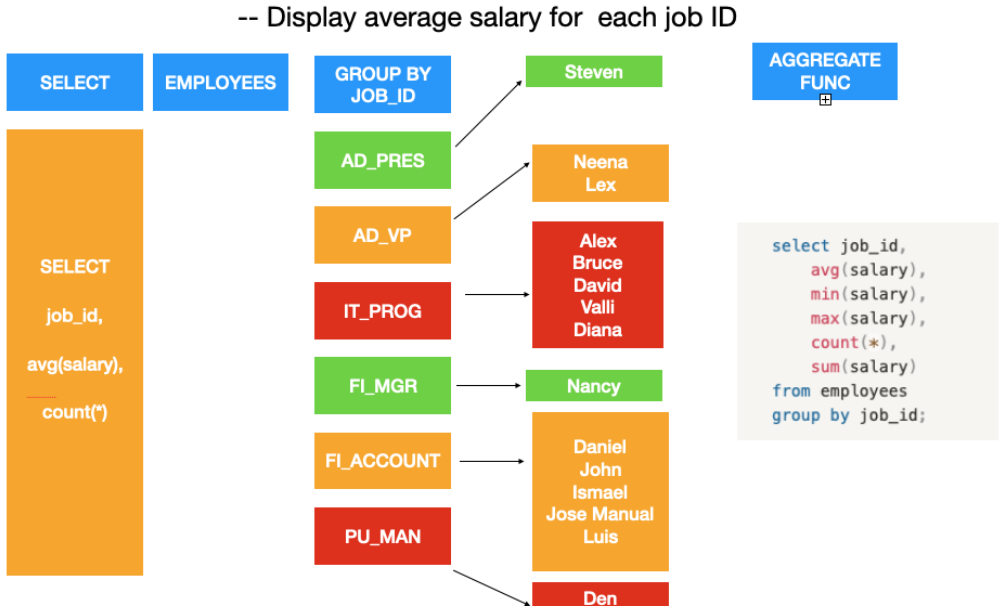
HOW IT WORKS ?

- It divides table into small pieces to apply multi row functions based on provided column . In that example we divide that table based on job id to get avg / max/ min /sum salary and count(*) of employees for **EACH JOB_ID**



	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID
1	100	Steven	King	SKING	515.123.4567	2003-06-17	AD_PRES	24000.00	<null>	<null>
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	2005-09-21	AD_VP	17000.00	<null>	100
3	102	Lex	De Haan	LDEHAAN	515.123.4569	2001-01-13	AD_VP	17000.00	<null>	100
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	2006-01-03	IT_PROG	9000.00	<null>	102
5	104	Bruce	Ernst	BERNST	590.423.4568	2007-05-21	IT_PROG	6000.00	<null>	103
6	105	David	Austin	DAUSTIN	590.423.4569	2005-06-25	IT_PROG	4800.00	<null>	103
7	106	Valli	Pataballa	VPATABAL	590.423.4560	2006-02-05	IT_PROG	4800.00	<null>	103
8	107	Diana	Lorentz	DLORENTZ	590.423.5567	2007-02-07	IT_PROG	4200.00	<null>	103
9	108	Nancy	Greenberg	NGREENBE	515.124.4569	2002-08-17	FI_MGR	12008.00	<null>	101
10	109	Daniel	Faviet	DFAVIET	515.124.4169	2002-08-16	FI_ACCOUNT	9000.00	<null>	108
11	110	John	Chen	JCHEN	515.124.4269	2005-09-28	FI_ACCOUNT	8200.00	<null>	108
12	111	Ismael	Sciarra	ISCIARRA	515.124.4369	2005-09-30	FI_ACCOUNT	7700.00	<null>	108
13	112	Jose Manuel	Urman	JMURMAN	515.124.4469	2006-03-07	FI_ACCOUNT	7800.00	<null>	108
14	113	Luis	Popp	LPOPP	515.124.4567	2007-12-07	FI_ACCOUNT	6900.00	<null>	108
15	114	Den	Raphaely	DRAPHEAL	515.127.4561	2002-12-07	PU_MAN	11000.00	<null>	100
16	115	Alexander	Khoo	AKHOO	515.127.4562	2003-05-18	PU_CLERK	3100.00	<null>	114
17	116	Shelli	Baida	SBAIDA	515.127.4563	2005-12-24	PU_CLERK	2900.00	<null>	114
18	117	Sigal	Tobias	STOBIAS	515.127.4564	2005-07-24	PU_CLERK	2800.00	<null>	114
19	118	Guy	Himuro	GHIMURO	515.127.4565	2006-11-15	PU_CLERK	2600.00	<null>	114
20	119	Karen	Colmenares	KCOLMENA	515.127.4566	2007-08-10	PU_CLERK	2500.00	<null>	114
21	120	Matthew	Weiss	MWEISS	650.123.1234	2004-07-18	ST_MAN	8000.00	<null>	100
22	121	Adam	Fripp	AFRIPP	650.123.2234	2005-04-10	ST_MAN	8200.00	<null>	100

24K
17K
9K
12K
9K
11K



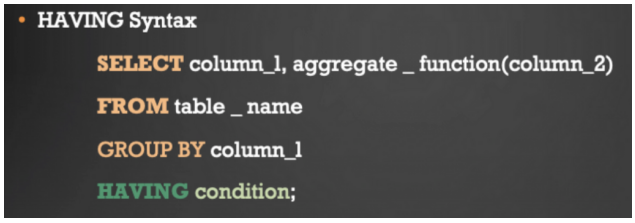

```
--- GROUP BY ---

-- Task1 : display average salary for IT_PROG
select avg(salary) from EMPLOYEES
where JOB_ID='IT_PROG';

-- Task 2  : display average salary for SA_REP
select avg(salary) from EMPLOYEES
where JOB_ID='SA_REP';

-- Task 3  : display average salary for SA_REP
select avg(salary) from EMPLOYEES
where JOB_ID='MK_MAN';
```

```
-- how many different job id we have ?
select count(distinct job_id) from EMPLOYEES;

/*
    INSTEAD OF TYING SAME QUERY FOR 19 DIFFERENT JOB IDs
    We are gonna group them based on JOB ID and get average salary in one shot
 */

select FIRST_NAME,LAST_NAME,JOB_ID,SALARY from EMPLOYEES;

-- get me AVG salary for EACH job_id from Employees table
select JOB_ID,avg(salary) from EMPLOYEES
group by JOB_ID;

select JOB_ID,avg(salary),min(salary),max(SALARY),sum(salary),count(*) from EMPLOYEES
group by JOB_ID;


-- get me total salary for EACH department from Employees table

select FIRST_NAME,LAST_NAME,DEPARTMENT_ID,SALARY from EMPLOYEES;

select DEPARTMENT_ID,sum(salary),count(*),max(salary),min(salary),round(avg(salary)) from EMPLOYEES
where DEPARTMENT_ID is not null
group by DEPARTMENT_ID;

-- order results based on employees count in desc
select DEPARTMENT_ID,sum(salary),count(*),max(salary),min(salary),round(avg(salary)) from EMPLOYEES
where DEPARTMENT_ID is not null   -- to remove null department id from result
group by DEPARTMENT_ID
order by count(*) desc;

-- order results based on department id in asc
select DEPARTMENT_ID,sum(salary),count(*),max(salary),min(salary),round(avg(salary)) from EMPLOYEES
where DEPARTMENT_ID is not null   -- to remove null department id from result
group by DEPARTMENT_ID
order by DEPARTMENT_ID;

-- order results based on avg(salary) in desc
select DEPARTMENT_ID,sum(salary),count(*),max(salary),min(salary),round(avg(salary)) from EMPLOYEES
where DEPARTMENT_ID is not null
group by DEPARTMENT_ID
order by avg(SALARY) desc;
```

HAVING

⚠️ What is difference between `HAVING` and `WHERE` ?

- “having” works same as “where” however it only works with group by.
- **Main difference between** the HAVING and WHERE clauses.
  - The HAVING statement sets the condition for group rows created by the GROUP BY clause after the GROUP BY clause applies while the WHERE clause sets the condition for individual rows before GROUP BY clause applies.



```
--- HAVING ---
-- display department_ids where their avg salary is more than 5K
select DEPARTMENT_ID,sum(salary),count(*),max(salary),min(salary),round(avg(salary)) from EMPLOYEES
where DEPARTMENT_ID is not null   -- to remove null department id from result
group by DEPARTMENT_ID
having avg(SALARY)>5000
order by avg(salary) desc;


-- display department id where employees count is bigger than 5
select DEPARTMENT_ID,count(*) from EMPLOYEES
group by DEPARTMENT_ID
having count(*)>5;

    -- order them based on number of employees in desc
        select DEPARTMENT_ID,count(*) from EMPLOYEES
        group by DEPARTMENT_ID
        having count(*)>5
        order by count(*) desc;

-- IQ --> display duplicated ( more than one) firstnames from employees table
select first_name,count(*) from EMPLOYEES
group by first_name
having count(*)>1
order by count(*) desc;
```

AS

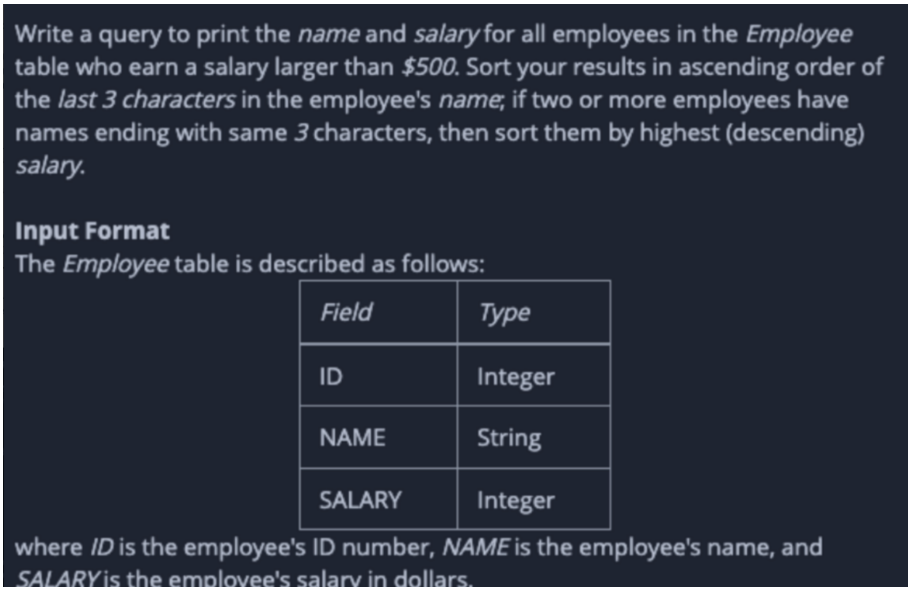
- Allow us to rename **columns** or **table** selections with an alias

SINGLE ROW FUNCTIONS

- They are ready methods / functions in SQL
- For Other functions click this link

```
--- AS ---
/*

 1.Column aliases --> it is temporary name to show in display


 2.Table aliases ---> before JOINS


 NOTE --> We are just displaying and nothing will change in DB

 */

select FIRST_NAME from EMPLOYEES;

-- It makes column name UPPERCASE to show in display.We CANT give any space too
select FIRST_NAME as given_name from EMPLOYEES;

-- it will show column as it is.Also we can give space too
select FIRST_NAME as "given name" from EMPLOYEES;

-- display annual salary for all employees as annual_salary
select FIRST_NAME, SALARY*12  from EMPLOYEES;

select FIRST_NAME, SALARY*12 as annual_salary from EMPLOYEES;

/*
```

```
 STRING FUNCTIONS

    -- CONCAT

    -- Java--> + --->     firstname+" "+lastname  --> Steven King
    -- SQL --> || -->     firstname||' '||lastname --> Steven King


 */

-- Display all employees firstname and lastnames as full name
-- first_name -->  Steven
-- last_name  -->  King
-- full_name  -->  Steven King

select FIRST_NAME||' '||LAST_NAME as full_name from EMPLOYEES;

-- Display email information as full_email with using @gmail.com domain
-- CONCAT (value1 ,value2)
select concat(EMAIL,'@gmail.com') as full_email from EMPLOYEES;

select concat('Cydeo ',concat(EMAIL,'@gmail.com')) as full_email from EMPLOYEES;

--UPPER (varchar)
select UPPER(EMAIL||'@gmail.com') as full_email from EMPLOYEES;
-- Steven KING
select FIRST_NAME||' '|| UPPER(LAST_NAME) as full_name from EMPLOYEES;

--LOWER (varchar)
select LOWER(EMAIL||'@gmail.com') as full_email from EMPLOYEES;

--INITCAP(varchar)
-- It makes first letter uppercase
select email from EMPLOYEES;
select initcap(email) from EMPLOYEES;


-- LENGTH(varchar)
select email,length(email||'@gmail.com') as email_length   from EMPLOYEES
order by email_length desc;


-- SUBSTR(columnName,beginningIndex,numberOfChar)
/*

    - if beginning index 0 , it is treated as 1
    - if the beginning index negative , it will start from backward
    - if we dont specify number of char it will work till the end


 */
    --display initial from firstname and lastname -->      S.K. ---  N.K.
        select * from EMPLOYEES;

        select substr(FIRST_NAME,0,1)||'.'||substr(LAST_NAME,0,1)||'.' as initials from employees;
        --                    S         .             K            .
        select substr(FIRST_NAME,1,1)||'.'||substr(LAST_NAME,1,1)||'.' as initials from employees;
        --                    S         .             K            .
        -- STEVEN --> VE
        -- NEENA  --> EN
        select FIRST_NAME,substr(FIRST_NAME,-3,2) from EMPLOYEES;

        -- STEVEN --> VEN
        -- NEENA  --> ENA
        select FIRST_NAME,substr(FIRST_NAME,-3) from EMPLOYEES;

-- IQ -->
/*
    Write a query to print first_name and salary for all employees in the Employee table who earn a salary larger than 3000
    Sort your results in ascending order of the last 3 characters in the employees first_name
    if two or more employees have first_names ending with same 3 characters, then sort them by highest salary
*/
```



Write a query to print the *name* and *salary* for all employees in the *Employee* table who earn a salary larger than *$500*. Sort your results in ascending order of the *last 3 characters* in the employee's *name*; if two or more employees have names ending with same *3* characters, then sort them by highest (descending) *salary*.

Input Format

The *Employee* table is described as follows:

Field	*Type*
ID	Integer
NAME	String
SALARY	Integer

where *ID* is the employee's ID number, *NAME* is the employee's name, and *SALARY* is the employee's salary in dollars.

VIEW

- Lets combine all the things
- To use same query again and again we can use view as a virtual table
- A VIEW in SQL Server is like a virtual table that contains data from one or multiple tables.
- It does not hold any data and does not exist physically in the database.

```
select * from employees;

create view EmployeesInfo as
select FIRST_NAME||' '||UPPER(LAST_NAME)||' makes '|| salary as employees_Salary,
       salary*12 as annual_salary from employees;
       -- Steven         KING      makes      24000
select * from EMPLOYEESINFO;

-- How many columns we have in view ?
    -- employees_Salary
    -- annual_salary

-- can we call specific column from view

select Employees_Salary from EMPLOYEESINFO;

drop view EMPLOYEESINFO;
```

SUBQUERY

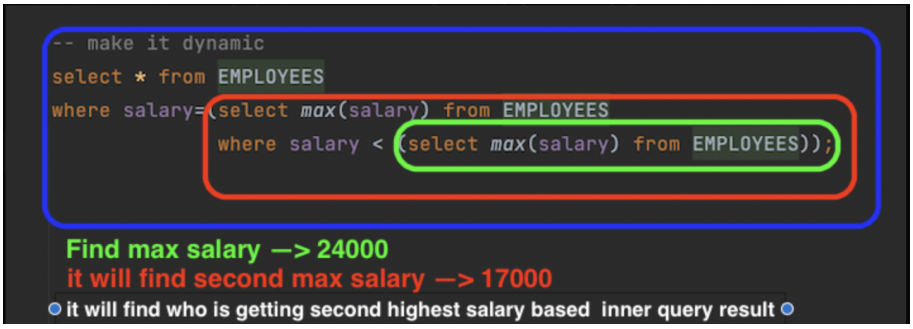
> ⚠️ Possible Interview Questions Topic

- Using one query inside another query inside the another query.
- We can use nested queries in sql
- A subquery is a query nested inside another query

```
SELECT *
FROM employees
WHERE salary = (SELECT MAX(salary) FROM employees);
```

```
    select * from employees;

    -- give me all information  who is getting max  salary

        -- max salary
        select max(salary) from EMPLOYEES;  --24 K

        -- if we know max salary can we find who is getting
        select *  from EMPLOYEES
        where  salary =24000;   --> it is HARCODED.

        -- if the Steven King salary changes we cant find person who is getting max salary

        -- SOLUTION --> make it dynamic
        select *  from EMPLOYEES
        where  salary=(select max(salary) from EMPLOYEES);


   -- give me all information  who is getting min salary

        -- min salary
        select min(salary) from EMPLOYEES;  --2100

        -- if we know min  salary can we find who is getting
        select *  from EMPLOYEES
        where  salary =2100;


        -- SOLUTION --> make it dynamic
        select *  from EMPLOYEES
        where  salary=(select min(salary) from EMPLOYEES);

   -- display all information who is getting second highest salary ?

        -- find max salary
        select max(salary) from EMPLOYEES;  --24 K

        -- second highest
        select max(salary) from EMPLOYEES
        where salary < 24000;  ---> 17 K

        -- find second highest dynamicly
        select max(salary) from EMPLOYEES
        where salary < (select max(salary) from EMPLOYEES); --17 K

        -- find me who is getting 2nd highest salary
        select * from EMPLOYEES
        where salary=17000;   --> still hardcoded

        -- make it dynamic
        select * from EMPLOYEES
        where salary=(select max(salary) from EMPLOYEES
                      where salary < (select max(salary) from EMPLOYEES));



-- display all information who is getting more than average ?
     -- find avg salary
        select avg(salary) from employees;

     -- display who is getting over avg
        select  * from EMPLOYEES
        where salary > 6462;

     -- SOLUTION --> make it dynamic
        select  * from EMPLOYEES
        where salary > (select avg(salary) from employees)
```

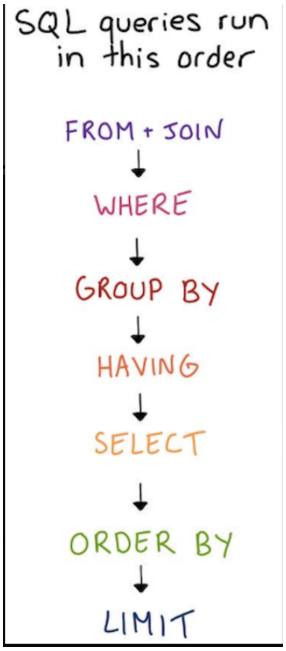
```
-- HOMEWORK --> display all information who is getting second minimum salary ?
```



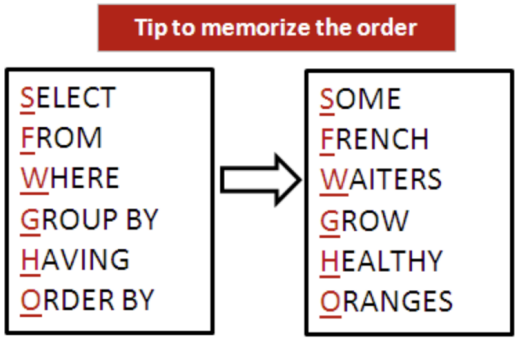
ROWNUM

⚠️ Possible Interview Questions Topic

- 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 (=).**
- Limits the number of result displayed in the query result.
- For other databases like postgresql, mysql the keyword is **LIMIT** and it comes after all clauses.
- if they ask like this question they want to see your subquery skill set



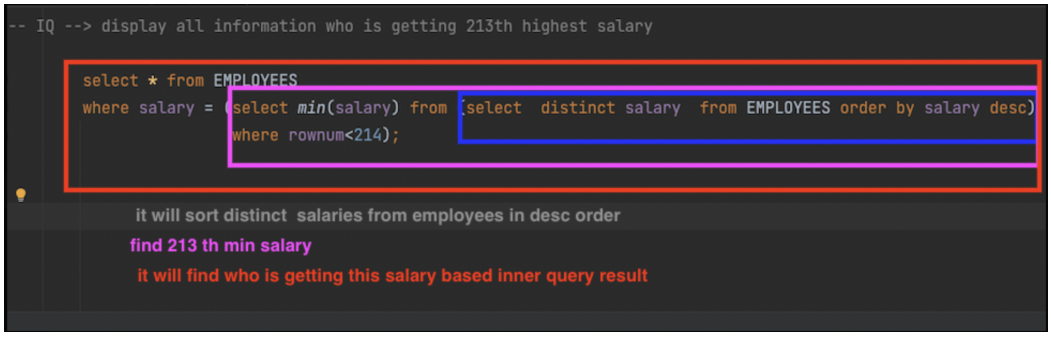
EXECUTION ORDER



ORDER TO WRITE QUERY

```
--- ROWNUM ---
/*
 it limits rows based on provided number
```

```
 */

select * from EMPLOYEES
where rownum<11;


--display all information from employees who is getting first 5 highest salary

-- BAD PRACTICE
-- It is getting data before order them based salary and
-- cut the list from line 6 then it tries to order them between first 5 rows
-- That is why we are getting WRONG LIST
select * from EMPLOYEES
where rownum<6
order by salary desc ;

-- CORRECT ANSWER
select * from (select * from EMPLOYEES order by salary desc )
where rownum<6;


-- display all information who is getting 5th highest salary
    --display all different salaries in desc order
    select  distinct salary  from EMPLOYEES order by salary desc;

    -- display 5th highest salary
        select min(salary) from (select distinct salary  from EMPLOYEES order by salary desc)
        where rownum<6;

    -- who is getting 5th highest salary

        select * from EMPLOYEES
        where salary=13000;      --it is hardcoded

        select * from EMPLOYEES
        where salary = (select min(salary) from (select  distinct salary  from EMPLOYEES order by salary desc)
                        where rownum<6);


-- IQ -->  display all information who is getting 213th highest salary
       --  display all information who is getting 4th highest salary

        select * from EMPLOYEES
        where salary = (select min(salary) from (select  distinct salary  from EMPLOYEES order by salary desc)
                        where rownum<5);

-- HOMEWORK -->  display all information who is getting 3th lowest salary
```



DDL(Data Definiton Language) & DML (Data Manipulation Language)

- **DDL is used to define data structures**

  CREATE - DROP- TRUNCATE-ALTER

- **DML is used to manipulate data itself.**

SELECT-INSERT-UPDATE-DELETE

RDBMS (Relational Database Management System)

- Relational Database is consists of tables that related to each other with Primary Key and Foreign Key

Table Constraints

One of the benefit of using database instead of excel or flat files , we can add rules for the table columns known as constraints ,

There are 5 of them :

PK (Primary Key): A primary key is used to uniquely identify all table records.

- non-duplicate Values
- CAN NOT be NULL
- Ex : employee_id in employees table —- department_id in departments table

FK (Foreign Key): A foreign key is an attribute or a set of attributes that **reference the primary key of some other table**. So, basically, it is used to link together two tables.

- it can be DUPLICATE or NULL for another table
- Ex : department_id in employee table

Unique Constraint :

- A column that can only have unique value,
- Unlike Primary key column , it can have null value
- You can have many `UNIQUE` constraints per table, but only one `PRIMARY KEY` constraint per table.

Not Null Constraint :

- This ensures that every column does not have a null value, such as an empty cell.

Check Constraint :

- This confirms that each entry in a column or row satisfies a precise condition and that every column holds unique data.
    - Assume that there is an employees salary and it cant be smaller than 60000
        - Create check constraint for salary
            - `salary Integer CHECK ( salary>60000 )`
        - Try to run following SQL Query

```
INSERT INTO team (employeeId, firstname, jobtitle, age, salary)
VALUES (1,'Mike','Tester',25,30000);  --> salary is smaller than
```

        - It ill give

```
[23000][2290] ORA-02290: check constraint (HR.SYS_C007788) violated
Position: 0
```

CREATE

- To create a new table SQL, you use the CREATE TABLE statement.
- First, you specify the name of the new table after the CREATE TABLE clause

  CONSTRAINT IS NOT MANDATORY TO GIVE
- Next, you list the column name, its data type, and column constraint. You can have multiple columns in a table, each column is separated by a comma (,). The column constraint defines the rules for the column e.g., NOT NULL.

```
CREATE TABLE table_name
(column name TYPE constraint);
```

INSERT

- When you create a new table, it does not have any data.
- The first thing you often do is to insert new rows into the table.
- SQL provides the **INSERT** statement that allows you to insert one or more rows into a table at a time.

```
INSERT INTO tableName (column1, column2,…)

VALUES (value1, value2 … );
```

- The value list must be in the same order as the columns list
- After all the insertion is done, you have to commit them.
- If you don't commit, it will not be enter in the database
  - Syntax: commit; or commit work;
  - We just need to do it one time at the end

UPDATE

- First, specify the table name where you want to update the data after UPDATE clause.
- Second, list the columns whose values you want to change in the SET clause.
- Third, determine which rows you want to update in the condition of the WHERE clause.

```
UPDATE table_name
SET column1 = value1,
    column2 = value2 , ..
WHERE condition;
```

DELETE

- To delete rows in a table, you use DELETE statement as follows:
- Second, specify **which row to delete** by using the condition in the WHERE clause.

```
DELETE FROM table_name
WHERE condition;
```

- The DELETE statement returns the number of rows deleted.
- If no rows are deleted, the DELETE statement returns zero.

ALTER

- To change existing table structure, you use ALTER TABLE statement.

```
ALTER TABLE table_name action;
```

☑ The popular actions that we can do with alter keyword:

- **ADD COLUMN** : adds column to the table
- **DROP COLUMN** : drops the column from the table
- **RENAME COLUMN** : renames the column name
- **RENAME TO** : renames the table name

TRUNCATE

- Cleans all the data from table BUT it keeps the columns / features and structure of table

```
TRUNCATE table_name;
```

DROP

- Deletes all table including with its columns

```
DROP TABLE table_name;
```

```
select * from employees;

/*
create table syntax:
    create table TableName(
        colName1 DataType Constraints,
        colName2 DataType Constraints(optional)
        colName3 DataType Constraints,
        ...
    );
*/
--- CREATE TABLE ----
create table scrumteam(

    empid Integer PRIMARY KEY,
    firstname varchar(30) not null,
    jobtitle varchar(20) not null,
    age integer,
    salary integer

);

select * from scrumteam;



 /*
INSERT INTO tableName (column1, column2,…)
VALUES (value1, value2 … );
*/
select * from scrumteam;

insert into scrumteam(empid, firstname, jobtitle, age, salary)
values (1,'Mike','Tester',34,120000);

insert into scrumteam(empid, firstname, jobtitle, age, salary)
values (2,'John','Tester',32,130000);

insert into scrumteam(empid, firstname, jobtitle, age, salary)
values (3,'John','Developer',37,160000);

insert into scrumteam(empid, firstname, jobtitle, age, salary)
values (4,'Dembe','Developer',42,200000);

-- to push changes into database we need to use commit or commit work
commit;


/*
UPDATE table_name
SET column1 = value1,
column2 = value2 , …
WHERE condition;
*/
select * from scrumteam;

UPDATE scrumteam
SET salary=salary+5000;

UPDATE scrumteam
SET salary=salary+20000
WHERE JOBTITLE='Tester';

UPDATE scrumteam
SET salary=salary+10000
WHERE age>40;

commit;

/*
```

```
DELETE FROM table_name
WHERE condition;
*/

DELETE from scrumteam
where empid=1;

select * from scrumteam;

DELETE from scrumteam
where jobtitle='Developer' and age>40;

commit;

-- ALTER

-- ADD NEW COLUMN
ALTER TABLE scrumteam add gender varchar(10);

select * from scrumteam;
-- update empID=2 gender as Male
update scrumteam
set gender='Male'
where empid=2;
-- check (gender in ('MALE', 'FEMALE'))

-- RENAME THE COLUMN
ALTER TABLE scrumteam rename column salary to annual_salary;

-- DROP COLUMN
ALTER TABLE scrumteam drop column gender;
select * from scrumteam;

-- RENAME TO ---> CHANGE TABLE NAME
ALTER TABLE scrumteam rename to agileteam;
select * from agileteam;
commit;

-- TRUNCATE
truncate table agileteam;
select * from agileteam;

-- DROP
drop table agileteam;
commit;

/*

 IO ---> What is difference between TRUNCATE and DROP

    TRUNC --> it will delete table content
    DROP  --> it will delete all table including contents too

 */
```

JOINS

- With the help of PK and FK we can communicate between two or more table to get the information

```
CREATE TABLE address(
address_id Integer PRIMARY KEY,
address VARCHAR(50) NOT NULL,
phone Integer NOT NULL
);

INSERT INTO address (address_id, address, phone) VALUES (5,
'1913 Hanoi Way',  28303384);
INSERT INTO address (address_id, address, phone) VALUES (7,
'692 Joliet Street',  44847719);
INSERT INTO address (address_id, address, phone) VALUES (8,
'1566 Inegl Manor',  70581400);
INSERT INTO address (address_id, address, phone) VALUES (10,
'1795 Santiago',  86045262);
INSERT INTO address (address_id, address, phone) VALUES (11,
'900 Santiago',  16571220);
```

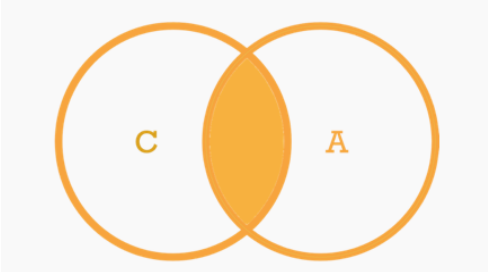
```
CREATE TABLE customer(
customer_id Integer PRIMARY KEY,
first_name VARCHAR(50) NOT NULL,
last_name VARCHAR(50)NOT NULL,
address_id Integer REFERENCES address(address_id)
);

INSERT INTO customer (customer_id, first_name, last_name,
address_id) VALUES (1, 'Mary' ,  'Smith',  5);
INSERT INTO customer (customer_id, first_name, last_name,
address_id) VALUES (2,  'Patricia' ,  'Johnson' ,  NULl);
INSERT INTO customer (customer_id, first_name, last_name,
address_id) VALUES (3,  'Linda' ,  'Williams' ,  7);
INSERT INTO customer (customer_id, first_name, last_name,
address_id) VALUES (4, 'Barbara' ,  'Jones' , 8);
INSERT INTO customer (customer_id, first_name, last_name,
address_id) VALUES (5,  'Elizabeth' ,  'Brown' ,  NULL);

commit work;
```

INNER JOIN

- To get data from both tables, we use the INNER JOIN clause in the SELECT statement as follows:
- **If it finds a match**, it combines columns of both rows into one row and add the combined row to the returned result set.
- Only matching portion of the tables. Inner join eliminates if there is no match

  IF YOU TYPE ONLY JOIN AS A DEFAULT IT IS INNER JOIN .

  GENERALLY YOU WILL USE INNER JOIN

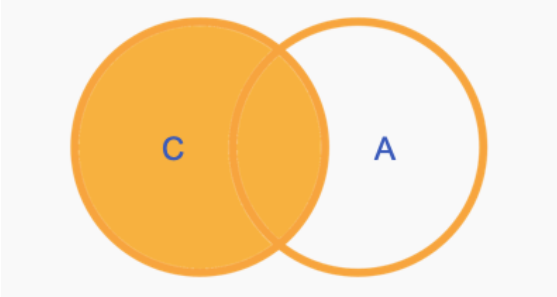


Inner Join Table

c_id	first_name	last_name	address	phone
1	Mary	Smith	1913 Hanoi Way	28303384
3	Linda	Williams	692 Joliet Street	44847719
4	Barbara	Jones	1566 Inegl Manor	70581400

LEFT OUTER JOIN

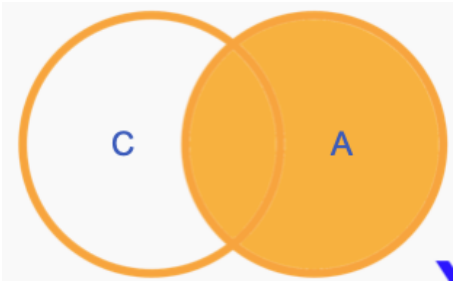
- A matching part from both tables and an unmatching part from the left table.
- Left outer join produces a complete set of records from Table customer, with the matching records (where available) in Table Address. If there is no match, the right side will contain null.



Customer				Address		
c_id	first_name	last_name	address_id	address_id	address	phone
1	Mary	Smith	5	5	1913 Hanoi Way	28303384
2	Patricia	Johnson	NULL	NULL	NULL	NULL
3	Linda	Williams	7	7	692 Joliet Street	44847719
4	Barbara	Jones	8	8	1566 Inegl Manor	70581400
5	Elizabeth	Brown	NULL	NULL	NULL	NULL

RIGHT OUTER JOIN

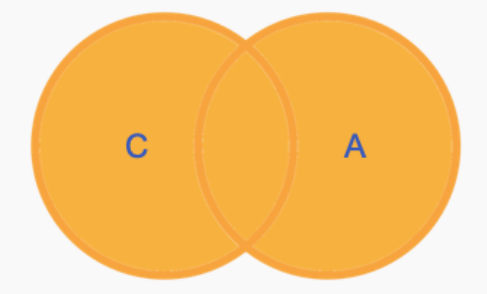
- Matching part from both table and unmatching part from right table.
- Right outer join produces a complete set of records from Table Address, with the matching records (where available) in Table Customer. If there is no match, the left side will contain null.



Customer				Address		
c_id	first_name	last_name	address_id	address_id	address	phone
1	Mary	Smith	5	5	1913 Hanoi Way	28303384
3	Linda	Williams	7	7	692 Joliet Street	44847719
4	Barbara	Jones	8	8	1566 Inegl Manor	70581400
NULL	NULL	NULL	NULL	10	1795 Santiago	86045262
NULL	NULL	NULL	NULL	11	900 Santiago	16571220

FULL OUTER JOIN

- All part of the tables both matching portions and unmatching portions from both tables.
- Full outer join produces the set of all records in Table Customer and Table Address with matching records from both sides where available. If there is no match, the missing side will contain null.



Customer				Address		
c_id	first_name	last_name	address_id	address_id	address	phone
1	Mary	Smith	5	5	1913 Hanoi Way	28303384
2	Patricia	Johnson	NULL	NULL	NULL	NULL
3	Linda	Williams	7	7	692 Joliet Street	44847719
4	Barbara	Jones	8	8	1566 Inegl Manor	70581400
5	Elizabeth	Brown	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	10	1795 Santiago	86045262
NULL	NULL	NULL	NULL	11	900 Santiago	16571220

```
select * from employees;

/*

 INNER JOIN

 - it gives only matching portion of tables
 - the order tables does not matter

 */
select * from CUSTOMER;
select * from ADDRESS;


select FIRST_NAME,LAST_NAME,ADDRESS,PHONE
    from ADDRESS inner join CUSTOMER
        on ADDRESS.ADDRESS_ID = CUSTOMER.ADDRESS_ID;

-- Another Q --> What if I want to customer.address_id information too
select FIRST_NAME,LAST_NAME,CUSTOMER.ADDRESS_ID,ADDRESS.ADDRESS_ID,ADDRESS,PHONE
from ADDRESS inner join CUSTOMER
                        on ADDRESS.ADDRESS_ID = CUSTOMER.ADDRESS_ID;


-- Another Q --> Do I need to type table name always
-- ALIASES
    -- Table Aliases
        -- Customer  --> C
        -- Employees --> E
        -- Address   --> A

select FIRST_NAME,LAST_NAME,C.ADDRESS_ID,A.ADDRESS_ID,ADDRESS,PHONE
from ADDRESS A  inner join CUSTOMER C
                        on A.ADDRESS_ID = C.ADDRESS_ID;

/*
 LEFT OUTER JOIN / LEFT JOIN

 -- it will give  matching part + unique for LEFT table
 -- The order of is important

 */
select FIRST_NAME,LAST_NAME,ADDRESS,PHONE
    from CUSTOMER C left outer join ADDRESS A
        on C.ADDRESS_ID = A.ADDRESS_ID;
```

```
select FIRST_NAME,LAST_NAME,ADDRESS,PHONE
    from ADDRESS A left outer join CUSTOMER C
    on A.ADDRESS_ID = C.ADDRESS_ID;

/*
 RIGHT OUTER JOIN / RIGHT JOIN

 -- it will give  matching part + unique for RIGHT table
 -- The order of is important

 */
select FIRST_NAME,LAST_NAME,ADDRESS,PHONE
    from customer c right outer join address a
    on c.ADDRESS_ID = a.ADDRESS_ID;

select FIRST_NAME,LAST_NAME,ADDRESS,PHONE
    from address a right join CUSTOMER c
    on a.ADDRESS_ID = c.ADDRESS_ID;



/*
 FULL OUTER JOIN

 -- It will all matching + unique for LEFT + unique for  RIGHT
 -- The order of tables does not matter

 */
select FIRST_NAME,LAST_NAME,ADDRESS,PHONE
from address a full join CUSTOMER c
                        on a.ADDRESS_ID = c.ADDRESS_ID;

/*
```

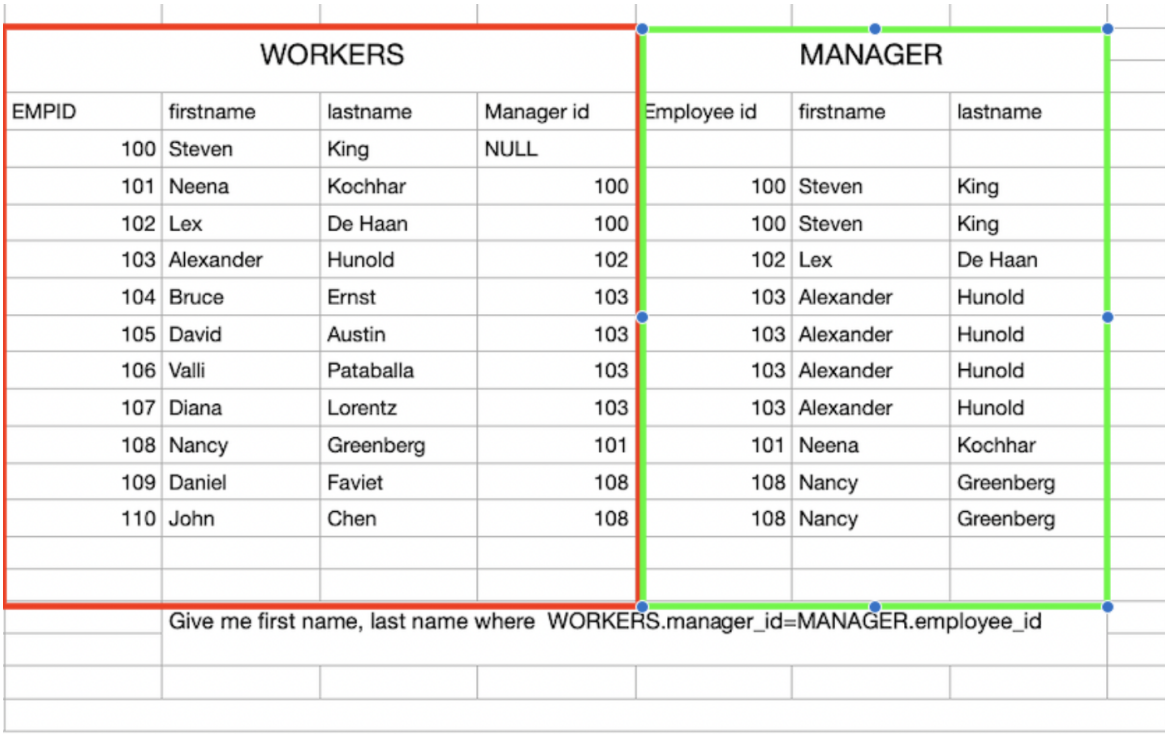
JOIN EXERCISES

```
select * from employees;

--1.Display all first_name and  related department_name

    --- Steven    Executive
    --- David        IT
    select * from employees;
    select * from departments;

    select FIRST_NAME, DEPARTMENT_NAME
        from EMPLOYEES E inner join DEPARTMENTS D
            on E.DEPARTMENT_ID = D.DEPARTMENT_ID;


    -- order list based firstname
        select FIRST_NAME, DEPARTMENT_NAME
        from EMPLOYEES E inner join DEPARTMENTS D
                                    on E.DEPARTMENT_ID = D.DEPARTMENT_ID
        order by FIRST_NAME ;

--2.Display all first_name and department_name including the  department without employee
    select * from employees;
    select * from departments;
    -- RIGHT JOIN
    select FIRST_NAME,DEPARTMENT_NAME from employees e right join departments d
        on e.DEPARTMENT_ID = d.DEPARTMENT_ID;

    -- LEFT  JOIN
    select FIRST_NAME,DEPARTMENT_NAME from departments d left join employees e
        on d.DEPARTMENT_ID = e.DEPARTMENT_ID;

    -- how many employee we have for each department name
    select DEPARTMENT_NAME,count(*) from departments d left join employees e
           on d.DEPARTMENT_ID = e.DEPARTMENT_ID
    group by DEPARTMENT_NAME;


--3.Display all first_name and department_name including the  employee without department
```

```
select * from employees;
select * from departments;

    -- Matching portion+unique Employee  --> LEFT JOIN
        select FIRST_NAME,DEPARTMENT_NAME from employees e left outer join departments d
            on e.DEPARTMENT_ID = d.DEPARTMENT_ID;

    -- RIGHT JOIN
        select FIRST_NAME,DEPARTMENT_NAME from departments d right join employees e
            on d.DEPARTMENT_ID = e.DEPARTMENT_ID;


--4.Display all first_name and department_name
--including the  department without employee  and  employees  without departments
    select FIRST_NAME,DEPARTMENT_NAME from departments d full join employees e
               on d.DEPARTMENT_ID = e.DEPARTMENT_ID;
```

SELF JOIN

- A self-join is a join in which a table is joined with itself, especially when the table has a **FOREIGN KEY** that references its own **PRIMARY KEY**
- The self join can be viewed as a join of two copies of the same table



WORKERS				MANAGER		
EMPID	firstname	lastname	Manager id	Employee id	firstname	lastname
100	Steven	King	NULL			
101	Neena	Kochhar	100	100	Steven	King
102	Lex	De Haan	100	100	Steven	King
103	Alexander	Hunold	102	102	Lex	De Haan
104	Bruce	Ernst	103	103	Alexander	Hunold
105	David	Austin	103	103	Alexander	Hunold
106	Valli	Pataballa	103	103	Alexander	Hunold
107	Diana	Lorentz	103	103	Alexander	Hunold
108	Nancy	Greenberg	101	101	Neena	Kochhar
109	Daniel	Faviet	108	108	Nancy	Greenberg
110	John	Chen	108	108	Nancy	Greenberg

Give me first name, last name where WORKERS.manager_id=MANAGER.employee_id

```
-- Display manager name of 'Neena'

-- manager_id for Neena
select MANAGER_ID from employees
where FIRST_NAME='Neena';

-- manager firstname and lastname for Neena
select FIRST_NAME,LAST_NAME from EMPLOYEES
where EMPLOYEE_ID=(select MANAGER_ID from employees
```

```
                    where FIRST_NAME='Neena');

-- all employees with firstname,lastname,manager_id
select EMPLOYEE_ID,FIRST_NAME,LAST_NAME,MANAGER_ID from EMPLOYEES;

-- display all employees and their managers
select WORKERS.FIRST_NAME,WORKERS.LAST_NAME,MANAGERS.FIRST_NAME,MANAGERS.LAST_NAME
from EMPLOYEES WORKERS  inner join EMPLOYEES MANAGERS
    on WORKERS.MANAGER_ID=MANAGERS.EMPLOYEE_ID;

-- Where is STEVEN ?

-- to see Steven
select WORKERS.FIRST_NAME,WORKERS.LAST_NAME,MANAGERS.FIRST_NAME,MANAGERS.LAST_NAME
from EMPLOYEES WORKERS  left join EMPLOYEES MANAGERS
                                  on WORKERS.MANAGER_ID=MANAGERS.EMPLOYEE_ID
order by WORKERS.FIRST_NAME;


-- TASK --> Given the Employee table, write a SQL query that finds out employees who earn more than their managers.
```

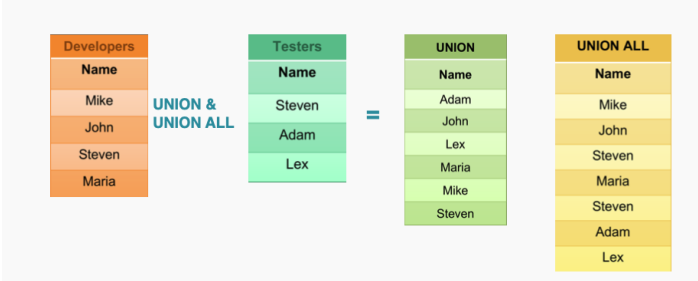
SET OPERATORS

```
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 into developers values (1, 'Mike', 155000);
insert into developers values (2, 'John', 142000);
insert into developers values (3, 'Steven', 850000);
insert into developers values (4, 'Maria', 120000);
insert into testers values (1, 'Steven', 110000);
insert into testers values(2, 'Adam', 105000);
insert into testers values (3, 'Lex', 100000);

commit work;
```

- For SET operators to work :
  - You need 2 independent queries
  - Same number of columns in Select statement
  - Same data type in same order

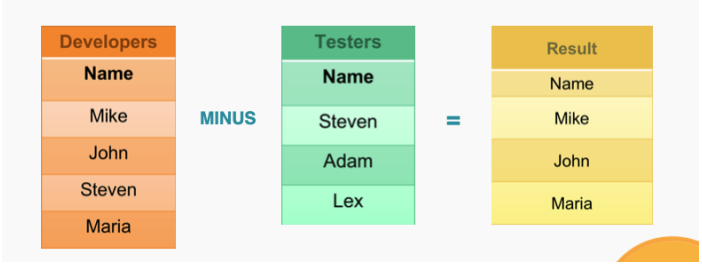
UNION VS UNION ALL

- The UNION operator combines result sets of two or more SELECT statements into a single result set.
- The following are rules applied to the queries:
  - Both queries must return the same number of columns.
  - The corresponding columns in the queries must have compatible data types.
- **The UNION operator removes all duplicate rows unless the UNION ALL is used.**
- We often use the UNION operator to combine data from similar tables that are not perfectly normalized.
- As a default it will order table according to ID_NUMBER for UNION



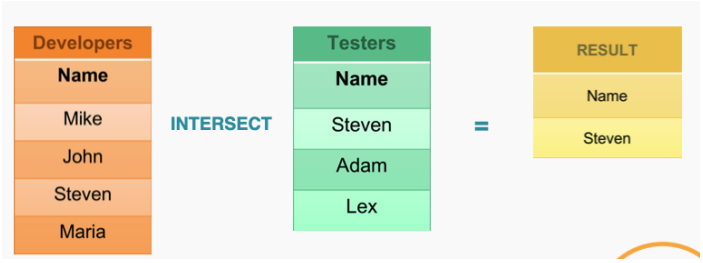
MINUS

- MINUS set operator returns records from first query that is not present in second query.
- It will only return values (from 1st query) that are not common in 2 queries
- Let's say we have two query result.



INTERSECT

- INTERSECT set operators returns records that are present/common/appear in both query results.
- It will sort and remove duplicates.
- Let's say we have two query result.



SUMMARY OF SET OPERATORS

- **UNION** -> combines, removes duplicates, sorts
- **UNION ALL.** -> combines, does not remove duplicates, does not sort
- **MINUS** -> show records from query1 that are `not` present in query2
- **INTERSECT** -> show `common records` from 2 queries

```
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
);
select * from Testers;

insert into developers values (1, 'Mike', 155000);
insert into developers values (2, 'John', 142000);
insert into developers values (3, 'Steven', 850000);
insert into developers values (4, 'Maria', 120000);
insert into testers values (1, 'Steven', 110000);
insert into testers values(2, 'Adam', 105000);
insert into testers values (3, 'Lex', 100000);

commit work;

select * from testers;
select * from developers;

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
);
select * from Testers;

insert into developers values (1, 'Mike', 155000);
insert into developers values (2, 'John', 142000);
insert into developers values (3, 'Steven', 850000);
insert into developers values (4, 'Maria', 120000);
insert into testers values (1, 'Steven', 110000);
insert into testers values(2, 'Adam', 105000);
insert into testers values (3, 'Lex', 100000);

commit work;

select * from testers;
select * from developers;

/*
 UNION

 - to merge more then one table vertically
 - Remove DUPs
 - Make an order as default asc based on ID


 */

select * from developers
union
select * from testers;


select names from developers
union
select names from testers;



/*
```

```
 UNION ALL

 - Dont Remove DUPS
 - Dont sort the results
 */

select * from developers
union all
select * from testers;


select names,salary from developers
union all
select names,salary from testers;


/*
MINUS
 It removes same data from first table and gives result from first table

 */


select * from developers
minus
select * from testers;


select names from developers
minus
select names from testers;

select * from employees;
/*
 INTERSECT

 */
select * from developers
intersect
select * from testers;


select names from developers
intersect
select names from testers;
```

QUESTIONS

```
1-Do you have experience in creating SQL queries?

2-What types of Joins do you know?

3-Do you know about joins in sql? Let's say we have an employee_id in two tables,
how do you use joins to extract data from the two tables?

4-How do you know relational databases tables are connected each other?

5-What are wildcards in SQL, give an example?

6-How to get the second max value in a table?

7-SQL Command Language, what are DML commands? What does SQL,DML,DDL stand for?

8-On a scale of 1-10 (10 being the highest) how would you rate yourself in SQL?

9-Whats primary key and foreign key?

10-How do you get unique values with SQL?

11-Tell me about joins? how many of them?

12-How do you join 3 table
```

```
13-Third highest salary

14-DESC and ASC in sql

15-What is Distinct keyword in sql

16-what is difference between union and union All
```