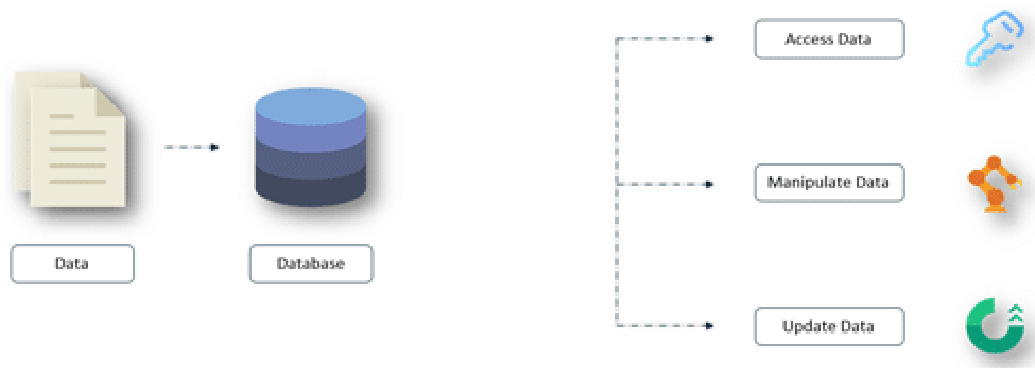
❖ What Is a Database?

- A database is a well-organized collection of data that is stored in an electronic format. To be more specific, a SQL database is an electronic system that allows to easily access, manipulate, and update the data.
- **Ex.** An online telephone directory uses a database to store data of people, phone numbers, and other contact details.



❖ Why we need Database?

- Manage large amounts of data
- Difficult to manage data in spreadsheets.
- Manual validation of data in spreadsheet is difficult
- Flexibility to update database in Database.
- Multiple people can edit Data at same time.

❖ SQL- Structured Query Language

- SQL is a standard language for storing, manipulating and retrieving data in databases
- SQL keywords are **NOT case sensitive** Ex. SELECT as select
- Semicolon is the standard way to separate each SQL statement in database systems. ex. Select * from employee**;**
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987
- Standard language for dealing with Relational Database which can be used to **Create**, **Read**, **Update** and **Delete** database records(**CRUD** Operations)

❖ What SQL can do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

❖ **Few Popular Databases Management Studio**

ORACLE DATABASE

MySQL

mongoDB

Microsoft SQL Server

MariaDB

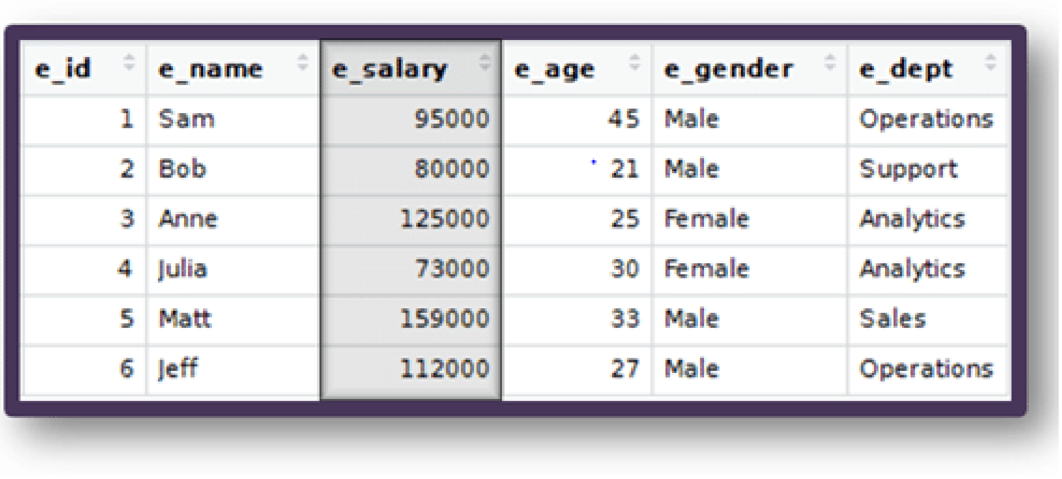
PostgreSQL

Microsoft® Access

IBM DB2

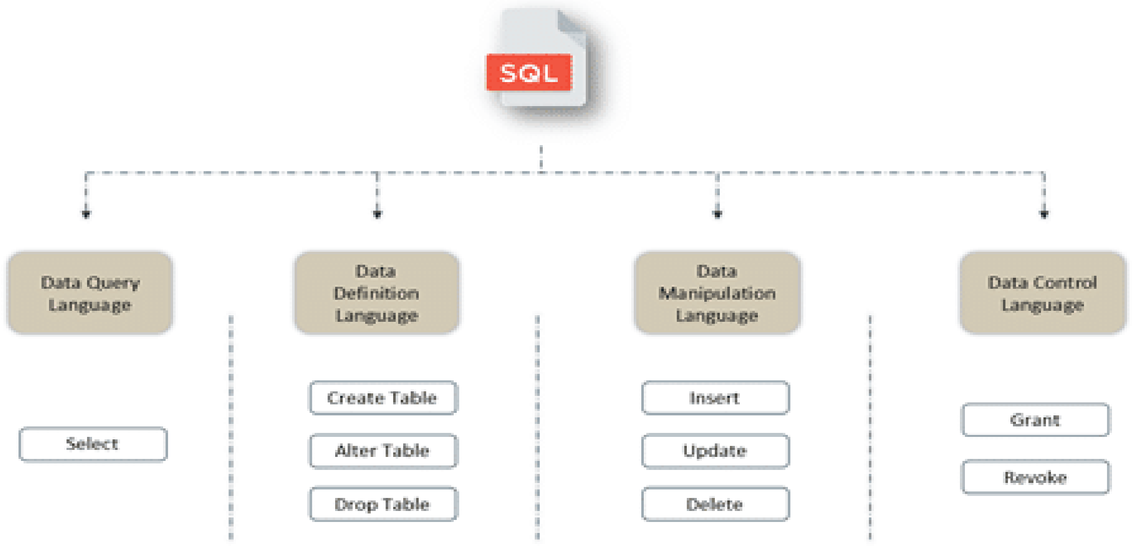
❖ **Tables in SQL: Records and Fields**

- Tables contain rows and columns, where the rows are known as records and the columns are known as fields
- Tables are **database objects that contain all the data in a database**.
- Table is a collection of data, organized in terms of rows and columns.



e_id	e_name	e_salary	e_age	e_gender	e_dept
1	Sam	95000	45	Male	Operations
2	Bob	80000	21	Male	Support
3	Anne	125000	25	Female	Analytics
4	Julia	73000	30	Female	Analytics
5	Matt	159000	33	Male	Sales
6	Jeff	112000	27	Male	Operations

- **SQL Commands:**
  Divided into four categories:
  - Data Query Language (DQL Commands in SQL)
  - Data Definition Language (DDL Commands in SQL)
  - Data Manipulation Language (DML Commands in SQL)
  - Data Control Language (DCL Commands in SQL)



- **Data Query Language (DQL Commands in SQL)**
  - DQL is used to fetch the data from the database.
  - Data Query Language comprises only one command '**select**.' This command can be accompanied by many other clauses to compose queries.

Command	**What it does?**
Select	It retrieves the data/information from the database/table

- **Data Definition Language (DDL Commands in SQL)**
  - The basic DDL commands in SQL are Create Tables, Alter Tables, Drop Tables and Truncate Tables.
  - DDL is used to perform the Create Tables/database, Alter Tables/Database, Drop Table/database and Truncate Table/database.
  - All DDL command are auto-committed that means it permanently save all the changes in the database.

Command	**What it does?**
CREATE TABLE	It creates new table
DROP TABLE	It deletes the ENTIRE table
ALTER TABLE	Modifies the existing table
TRUNCATE TABLE	Deletes the data inside a table, but not the table itself. (**column/structure will remain same**)

Data Manipulation Language (DML Commands in SQL)

- To deal with the data itself directly.
- SQL Commands is used to perform the operations: INSERT, UPDATE, and DELETE.
- DML commands are used to modify the database.

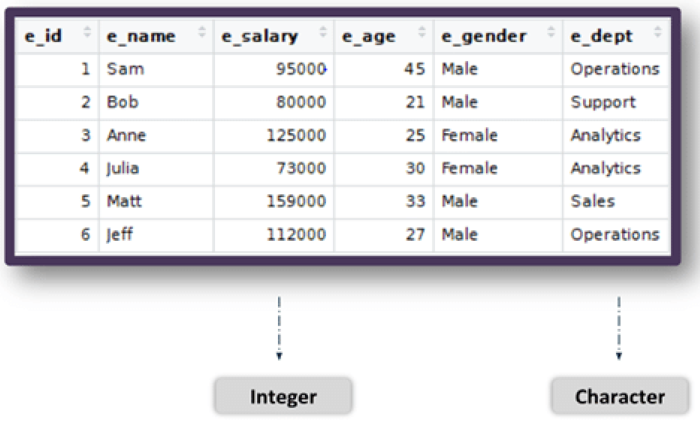
Command	**What it does?**
INSERT	Add new information to the database/table
UPDATE	Modifies the information currently stored in the database/table.
DELETE	Delete information from the database/table

Data control language (optional to learn for tester)

- DCL Commands are used to grant and take back authority from any database user.
    - **Grant**: It is used to give user access privileges to a database.
      Syntax:
      ```
      GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;
      ```
    - **Revoke:** It is used to take back permission from the user.
      Syntax:
      ```
      REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;
      ```

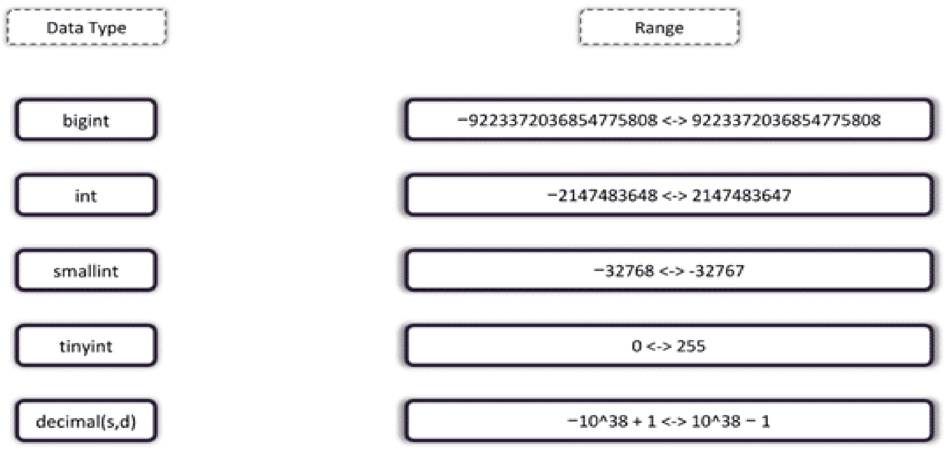
What Is SQL Data Types?

- Data type in SQL basically defines the kind of data that will go into a particular column. All entries of one particular column will be of the same data type.



e_id	e_name	e_salary	e_age	e_gender	e_dept
1	Sam	95000	45	Male	Operations
2	Bob	80000	21	Male	Support
3	Anne	125000	25	Female	Analytics
4	Julia	73000	30	Female	Analytics
5	Matt	159000	33	Male	Sales
6	Jeff	112000	27	Male	Operations

- **Numeric Data types in SQL?**
    - Numeric data types stores all numeric values or integer values.

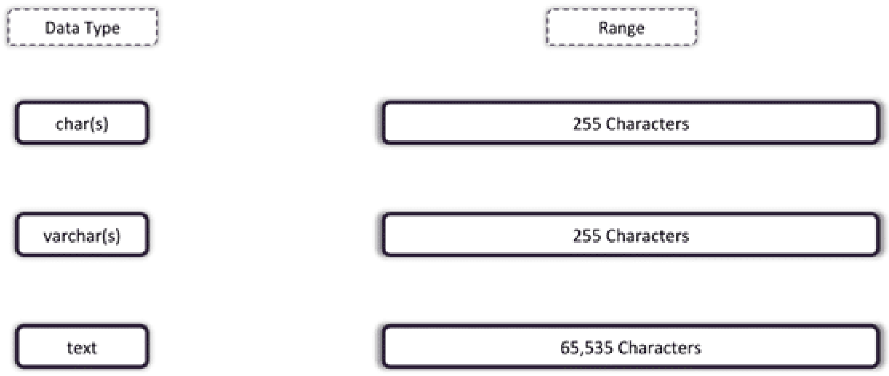


Data Type	Range
bigint	−9223372036854775808 <-> 9223372036854775808
int	−2147483648 <-> 2147483647
smallint	−32768 <-> -32767
tinyint	0 <-> 255
decimal(s,d)	−10^38 + 1 <-> 10^38 − 1

Note: In Decimal(s, d) = s for size and d for decimal

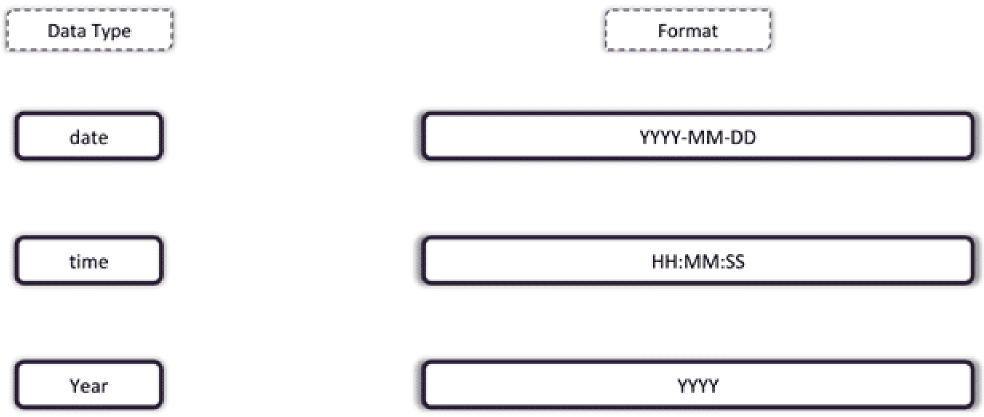
***Decimal* (3, 2) means the value can have 3 digits overall and 2 digits to the right of the decimal.**

- **Character Data Types in SQL**
    - Character data types store all alphabetic values and special characters.



Data Type	Range
char(s)	255 Characters
varchar(s)	255 Characters
text	65,535 Characters

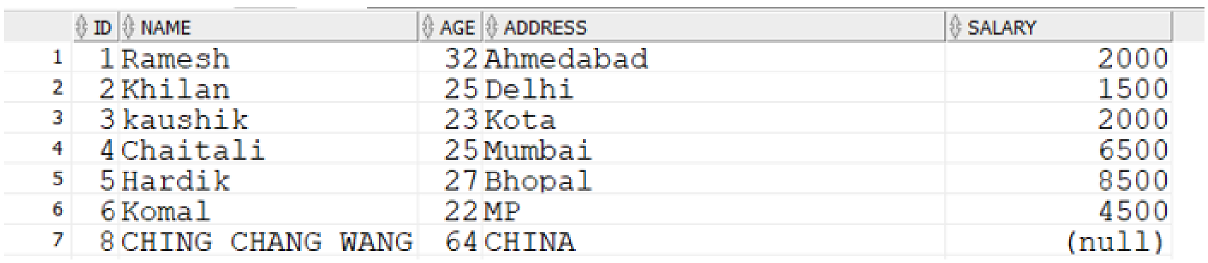
- **Date And Times Data types in SQL**
    - Date and Time data types store a date or a date/time value.



Data Type	Format
date	YYYY-MM-DD
time	HH:MM:SS
Year	YYYY

❖ **Tables in SQL**

- Tables are database objects that contain all data in database. In tables, data is logically organized in a row and column format similar to spreadsheets.



	ID	NAME	AGE	ADDRESS	SALARY
1	1	Ramesh	32	Ahmedabad	2000
2	2	Khilan	25	Delhi	1500
3	3	kaushik	23	Kota	2000
4	4	Chaitali	25	Mumbai	6500
5	5	Hardik	27	Bhopal	8500
6	6	Komal	22	MP	4500
7	8	CHING CHANG WANG	64	CHINA	(null)

❖ **SQL DDL (data definition language) Command**

- DDL changes the structure of the table like creating a table, deleting a table, altering a table etc.
- All the commands of DDL are auto committed that means it permanently save all the changes in database.
- Commands are:
    - **CREATE:** It is used to create a new table in the database.
      Syntax:
      CREATE TABLE table_name (
      Column1 datatype,
      column2 datatype,
      column3 datatype……);

Example:

```
CREATE TABLE CUSTOMERS (
  ID   INT,
  NAME VARCHAR (20),
  AGE INT,
  ADDRESS CHAR (25),
  SALARY   INT
);
```

Note : we also create Database using Following Syntax

CREATE DATABASE *databasename*;
Ex. CREATE DATABASE velocity;

- **ALTER:** The Alter Table statement is used to add, delete or modify columns.
  - If you want to add columns in SQL table, use following **Syntax**:

    ```
    Alter Table Table_name
    ADD column_name column-defination;
    // column definition is i.e. datatypes;
    ```

    Ex:

    ```
    ALTER TABLE CUSTOMERS
    ADD CITY VARCHAR(255);
    ```

  - If you want to modify an existing column in SQL use following **Syntax**:

    ```
    ALTER TABLE Table_name
    MODIFY column_name Column_type;
    ```

    EX:

    ```
    ALTER TABLE CUSTOMERS

    MODIFY CITY CHAR(25);
    ```

    EX: Multiple Column Modify

    ```
    ALTER TABLE CUSTOMERS

    MODIFY ( CITY CHAR(25),

           NAME CHAR(50)   );
    ```

- If we want to delete column by alter command in SQL is

  Syntax:

  ALTER TABLE Table_name

  DROP COLUMN Column_name**;**

  Ex:
  ALTER TABLE CUSTOMERS
  DROP COLUMN CITY**;**

▪ **DROP:**

- The drop Database, Drop table statement is used to drop an existing SQL database or existing table in Database.
- It will remove complete structure from Database.
- This is very important to know that once a table is deleted all the information available in the table is lost forever, so we have to be very careful when using DROP table or Drop Database command.
  Syntax:
  DROP DATABASE database_namE**;**
  EX:
  DROP DATABSE Velocity**;**
  Syntax for drop Table
  DROP TABLE Table_Name**;**
  Ex:
  DROP TABLE Customer**;**

▪ **TRUNCATE**

- The SQL truncate table command is used to delete complete data from an existing table
- You can also use DROP table command to delete table but drop command will remove complete table structure from the database. Where truncate command will delete only all information and structure will remain same.
  Syntax:

  TRUNCATE TABLE Table_name**;**

  TRUNCATE TABLE Customer**;**
- It is similar to the DELETE statement with **no** WHERE clause. I.e. DELETE FROM XYZ**;**

❖ **INSERT**

- Insert statement is used to insert a single record or multiple records into a table in SQL server.
- There are two ways :

  Syntax 1: Specify both the column names and the values to be inserted.
  By using this query we can insert data in specified columns.

  ```
  INSERT INTO Table_Name (column1 , column2,….)
  VALUES (value1, value2,……….);
  ```

  Ex:

  ```
  INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
  VALUES (1, ‘Swapnil’, 27, ‘Pusad’, 62000);
  ```

  Syntax 2: If you are adding for all the columns of the table, you do not need to specify the column names in the SQL query, however, make sure the order of the values is in the same order as the columns in the table.

  ```
  INSERT INTO Table_Name
  Values (value1, value2, -----);
  ```

  Ex:

  ```
  INSERT INTO CUSTOMERS
  VALUES (9, 'SWAPNIL', 25, 'PUNE', 120000);
  ```

❖ **SELECT**

- Select statement is used to fetch the data from database table.

Syntax:

1) It fetch all the columns from the table.
   ```
   SELET * FROM Table_Name;
   ```
   Ex:
   ```
   SELECT * FROM CSTOMERS;
   ```

2) It fetch specify column from the table.
   ```
   SELECT column_names FROM Table_name;
   ```
   Ex:
   ```
   SELECT First_Name  , Last_Name  FROM CUSTOMERS;
   ```

❖ **DISTINCT**

- It is used in conjunction with the select statement to eliminate all the duplicate records and fetching only unique records.
- This statement is used to return only distinct (different) values.

  Syntax1:
  ```
  SELECT DISTINCT * FROM Table_Name;
  ```
  Ex.
  ```
  SELECT DISTINCT * FROM Customer;
  ```

Syntax2:
SELECT DISTINCT column1, column2 , …
FROM Table_Name**;**
Ex.
SELECT DISTINCT CITY FROM CUSTOMERS.

Ex.
SELECT DISTINCT CITY, NAME FROM CUSTOMERS.

❖ SELECT TOP

- Select TOP query is used to fetch a TOP n number or X percent records from a table.
- It is useful on large tables with thousands of records. Returning a large number of records can impact performance.
- Note: Not all database systems support select TOP clause. MYSQL supports LIMIT clause to select a limited number of records, while oracle uses
  FETCH FIRST n ROWS ONLY
  And ROWNUM.

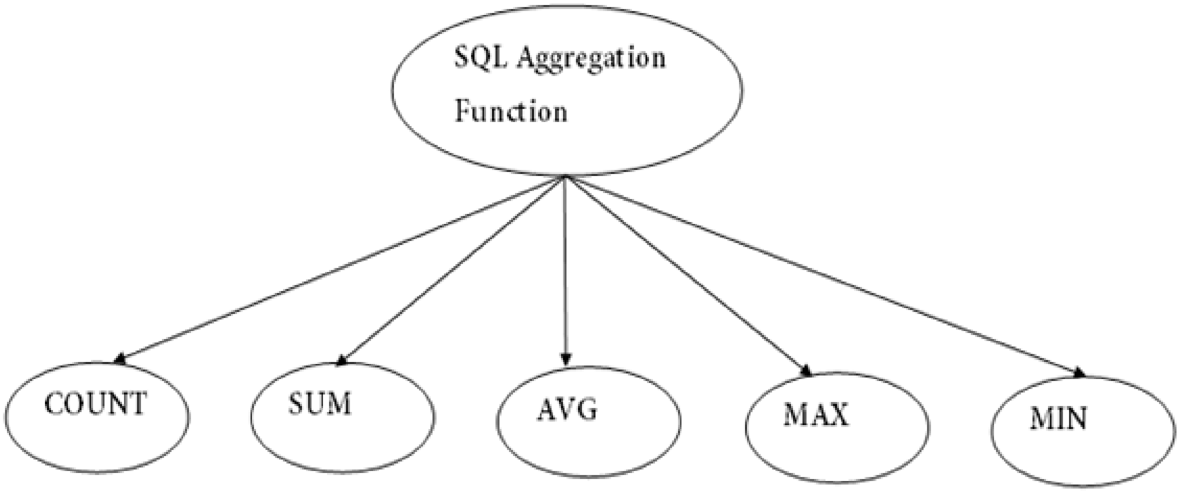
Syntax:
SELECT TOP NO/PERCENT *
FROM Table_Name**;**

Ex:
SEELCT TOP 3 *
FROM Customer**;**

SELECT TOP NO/PERCENT ColumnName
FROM Table_Name**;**
Ex.
SELECT TOP 3 CITY
FROM Customer**;**

❖ **Aggregate functions**

- SQL aggregate functions is used to perform the calculations on multiple rows of a single column of a table.
- It returns a single value.
- It has following types



1) **COUNT()**

- This function returns the number of rows that matches a specified criteria.
- Ex: If you have a record of the voters in selected area and want to count the number of voters then it is very difficult to do it manually but you can do it easily by using the SQL SELECT COUNT query.

  Syntax:
  SELECT COUNT (column name) FROM T.N.**;**
  Ex.
  SELECT COUNT (VoterID) FROM Voters**;**

  SELECT COUNT (*) FROM T.N.**;**

  SELECT COUNT * FROM Customer**;**

2) **Average Function // AVG( )**

- This function returns the average value of numeric value.
  Syntax:
  SELECT AVG (C.N) FROM T.N**;**
  Ex:
  SELECT AVG (MockResult) From VCTC;

3) **SUM function // SUM( )**
   - This function return the total sum of a numeric column.
     Syntax:
     SELECT SUM (C.N) FROM T.N.;
     Ex:
     SELECT SUM (MockResult) FROM VCTC;

4) **Maximum Function // MAX( )**
   - This functions returns the largest value of the selected column
     Syntax:
     SELECT MAX (C.N) FROM T.N.;
     Ex:
     SELECT MAX (MockResult) FROM VCTC;

5) **Minimum Function // Min( )**
   - This functions returns the smallest value of the selected column
     Syntax:
     SELECT MIN (C.N) FROM T.N.;
     Ex:
     SELECT MIN (MockResult) FROM VCTC;

❖ **Where Clause**

- It is used to specify a condition while fetching the data from a single table.
- It is used to filter records.
- It is used to extract only those records that fulfill a specified condition.
- The WHERE clause is not only used in select statement, it is also used in UPDATE, DELETE, etc.
  Syntax:
  SELECT CN1, CN2, CN3, FROM T.N
  WHERE Condition**;**
  // OR
  SELECT * FROM T.N
  WHERE Condition**;**

  Ex:
  SELECT * FROM T.N
  WHERE Name = ‘Swapnil’**;**

NOTE: SQL requires single quotes around text values (most database system also allow double quotes)
However, numeric fields should NOT be enclosed in quotes.

Ex: SELECT * FROM Customers
WHERE ID = 1**;**

❖ AND, OR, NOT

- This operators are used to combine multiple conditions to narrow data in an SQL statement.
- The where clause can be combined with AND, OR and NOT operators.
- The AND & OR operators are used to filter records based on more than one condition.
- **The AND** operator displays a record if all the conditions separated by AND are TRUE.
- **The OR** operator displays a record if any one condition separated by OR are TRUE.
- **The NOT** operator displays a record if the conditions is NOT TRUE.
- **Syntax**
  - SELECT CN1, CN2, ….. FROM TN
    WHERE Condition1 **AND** Condition2**;**
  - SELECT CN1, CN2, ….. FROM TN
    WHERE Condition1 **OR** Condition2**;**
  - SELECT CN1, CN2, ….. FROM TN
    WHERE **NOT** Condition**;**
- **Examples:**
  - SEELCT * FROM CUSTOMERS
    WHERE FNAME = 'SWAPNIL' **AND** LNAME = 'ROKADE'**;**
  - SEELCT * FROM CUSTOMERS
    WHERE FNAME = 'SWAPNIL' **OR** LNAME = 'ROKADE'**;**
  - SEELCT * FROM CUSTOMERS
    WHERE **NOT** FNAME = 'SWAPNIL'**;**

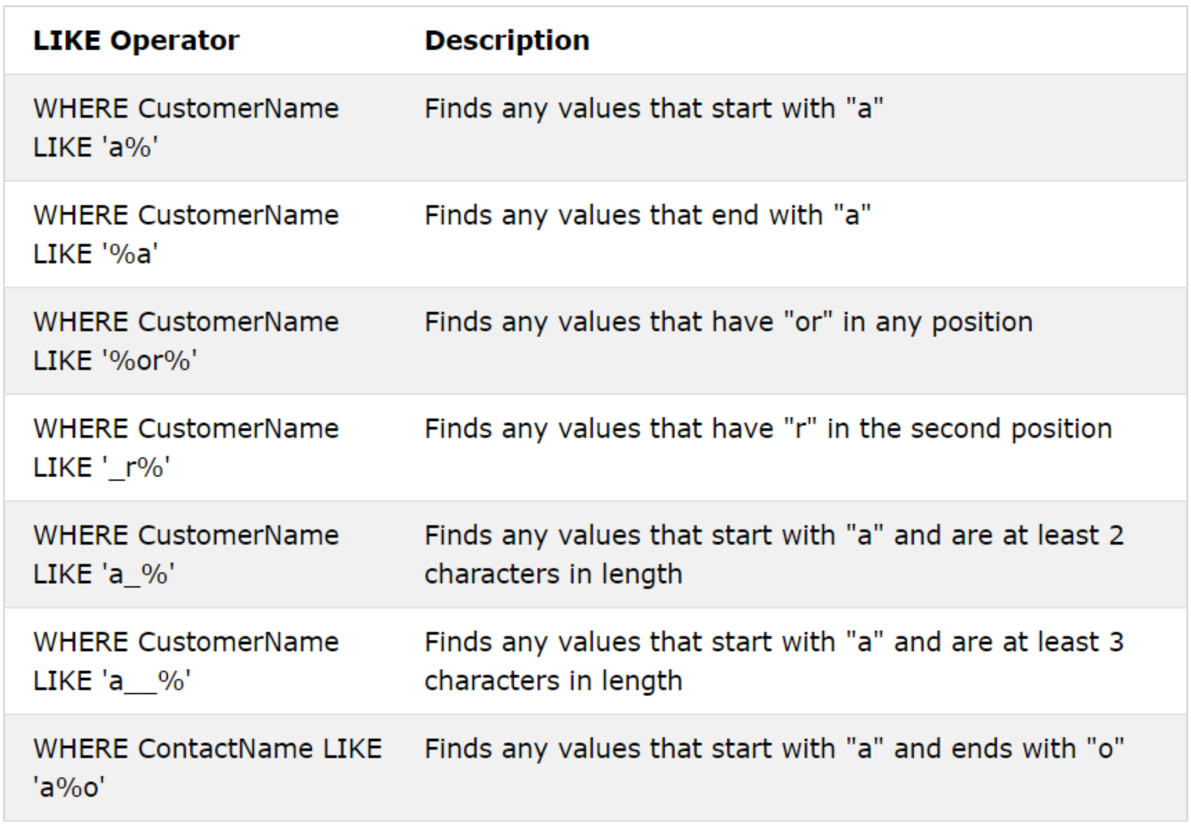
❖ LIKE

- Like clause is used to compare a value to similar values using wildcard operators.
- Like clause is used in a where clause to search for a specified pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator
  - The percent sign **%** represents zero, one or multiple characters
  - The underscore sign '_' represents one, single character.
  - Note: MS Access database uses asterisk * instead of the percent sign '%' and a question mark '?' Instead of underscore '_'.
- The percent sign and underscore can also use in combinations
- **Syntax:**
  - SELECT CN1, CN2, ….. FROM TN
    WHERE CN LIKE 'pattern'**;**

- **Examples**
    - SELECT * FROM Customer
      WHERE Fname LIKE ‘Adi%’**;**
    - SELECT * FROM Customer
      WHERE Fname LIKE ‘Adi_y’**;**

NOTE: A Wildcard character is used to substitute one or more characters in string. Which is used with LIKE cause.

- **Some Examples on Like Clause and Wildcards**



LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

❖ **BETWEEN**

- The between operator selects values value within a given range. The values can be numbers, text or dates.
- The Between operator is inclusive. i.e., begin and end values are included
- Syntax:
    - SELECT CN1,CN2, --- FROM TN
      WHERE CN BETWEEN value1 AND value2**;**
- Example
    - SELECT * FROM Employee
      WHERE Salary BETWEEN 10000 AND 20000;

- **IN**
  - The IN operator allows you to specify multiple values in a WHERE clause.
  - The IN operator is a shorthand for multiple OR conditions.
  - Syntax
    - SELECT * FROM TN
      WHERE CN IN (value1, value2,----ValueN)**;**
  - **Example**
    - SELECT * FROM Employee
      WHERE Salary **IN** (4000, 12000, 10000)**;**
  - **Example of NOT IN: which exactly opposite of IN operator.**
    SELECT * FROM Employee
    WHERE SALARY **NOT IN** (4500, 5000, 4000)**;**

- **IS NULL and IS NOT NULL**
- **What is Null Values?**
  - A field with a **NULL** value is a field with a **no value**.
  - If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then field will be saved a NULL value.
  - A NULL value is different from a zero value or a field that contains spaces.
  - A field with a NULL value is one that has been left blank during record creation.
  - We can test NULL values by using **IS NULL and IS NOT NULL** operators.
    - **IS NULL** operator is used to test for empty values (NULL values).
    - **IS NOT NULL** operator is used to test for non-empty values (NOT null values)
  - **Syntax:**
    - SELECT CN1, CN2 FROM TN
      WHERE CN **IS NULL;**
    - SELECT CN1 , CN2 FROM TN
      WHERE CN **IS NOT NULL;**
  - **Example:**
    - SELECT * FROM Employee
      WHERE Salary **IS NULL;**
    - SELECT * FROM Employee
      WHERE Salary **IS NOT NULL;**

- **ORDER BY CLAUSE**
  - The ORDER BY keyword is used to sort the result set in ascending or descending order
  - The ORDER BY keyword sorts the records in ascending order by defaults. To sort the records in descending order, use the **DESC** keyword.
  - And for ascending use keyword ASC.
  - Syntax:
    - SELECT CN1, CN2, …… FROM TN
      ORDER BY CN1, CN2, **ASC or DESC;**

- Example
  - SELECT * FROM Employee
    ORDER BY Salary **DESC;**
  - SELECT * FROM Employee
    ORDER BY Salary **ASC;**
  - **If you use following syntax then order is by ascending by default.**
    SELECT * FROM Employee
    ORDER BY Salary**;**
  - **In following example if you give two column order then first its order by first column salary and if suppose there are same salary then SQL server check Name column by descending order.**
    SELECT * FROM Employee
    ORDER BY Salary, Name **DESC;**

❖ **Aliases clause**

- SQL aliases are used to give a table, or column in a table, a temporary name.
- Aliases are often used to make column names more readable.
- An alias only exists for duration of that query.
- An alias is created with the **AS** keyword.
- **Syntax**:
  - SELECT CN AS Alias_name
    FROM TN**;**
  - SELECT CN'S
    FROM TN AS Alias_name
- **Example:**
  - SELECT SALARY AS SAL
    FROM EMPLOYEE**;**
  - SELECT SALARY AS SAL, ENAME AS NAME
    FROM EMPLOYEE**;**
  - SELECT SALARY
    FROM EMPLOYEE AS EMP;

❖ **UPDATE**

- The update statement is used to modify/update the existing records in a table.
- Be careful when you updating records. If you omit or forgot to write WHERE clause, All records will be update.
- **Syntax:**

  UPDATE TN
  SET Column1 = Value1, Column2 = Value2, …
  Where Condition**;**
- **Example:**

  UPDATE Customer

  SET Name = 'Swapnil', AGE = 27

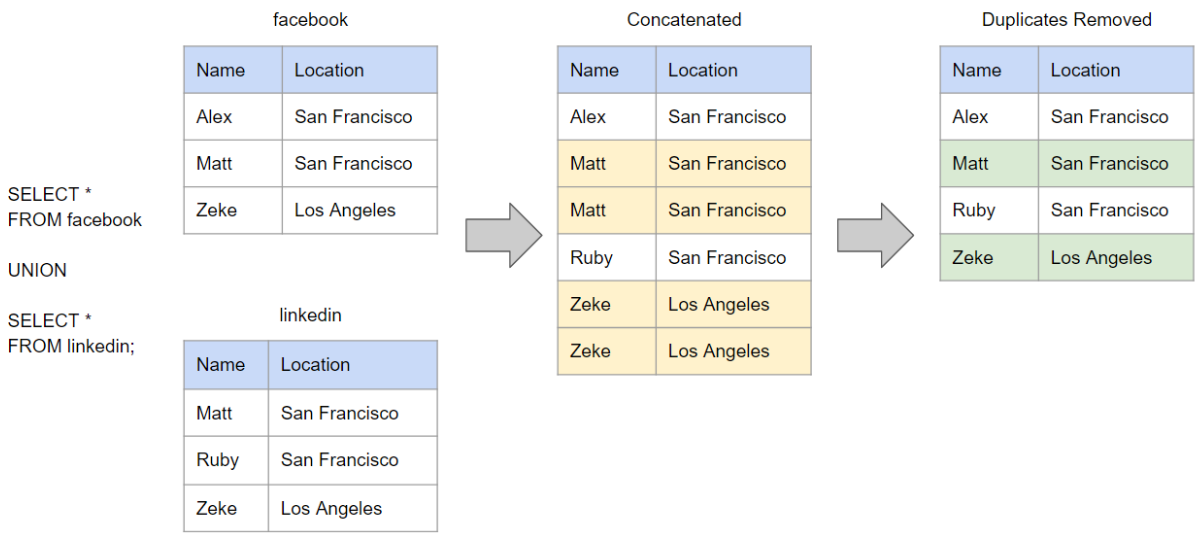
  WHERE Name = 'Mayur'**;**

- **DELETE**
  - The delete statement is used to delete existing records in a table.
  - Be careful whenever we are deleting records from table, if we are forget to write WHERE clause, then all records will be deleted.
  - Syntax

    DELETE FROM TN
    WHERE Condition**;**
  - Example

    DELETE FROM Customer

    Where Name = 'Mayur';
- **UNION**
  - The UNION operator is used to combine the result-set of two or more SELECT statements.
    - Every SELECT statement within UNION must have the same number of columns.
    - The column must also have similar data types.
    - The columns in every SELECT statement must also be in the same order.
    - **Syntax**:
      - SELECT CN'S FROM TN1
        UNION
        SELECT CN'S FROM TN2**;**
    - **Example**:
      - SELECT Name, Location FROM Facebook
        UNION
        SELECT Name, Location FROM LinkedIn**;**



facebook

Name	Location
Alex	San Francisco
Matt	San Francisco
Zeke	Los Angeles

linkedin

Name	Location
Matt	San Francisco
Ruby	San Francisco
Zeke	Los Angeles

Concatenated

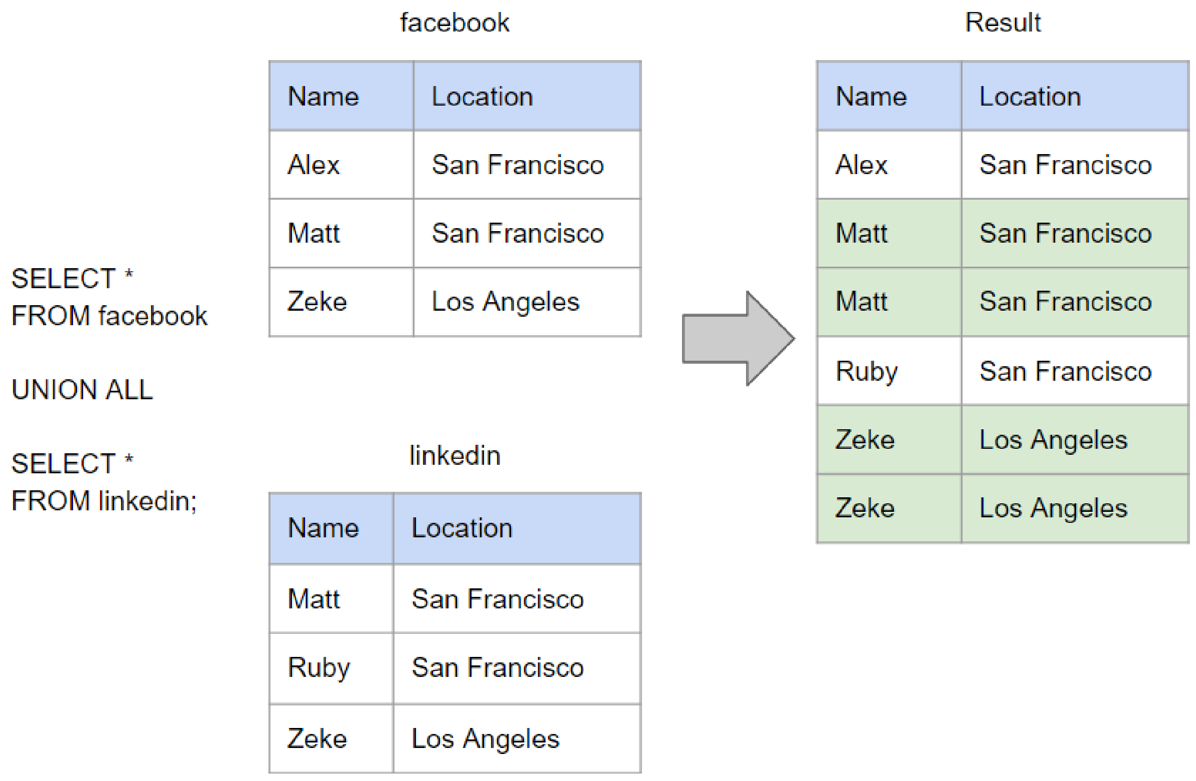
Name	Location
Alex	San Francisco
Matt	San Francisco
Matt	San Francisco
Ruby	San Francisco
Zeke	Los Angeles
Zeke	Los Angeles

Duplicates Removed

Name	Location
Alex	San Francisco
Matt	San Francisco
Ruby	San Francisco
Zeke	Los Angeles

Note: If some customers or suppliers have the same city, each city will only be listed once, because UNION selects only distinct values. Use UNION ALL to also select duplicate values!

- **UNION ALL**
  - The UNION operator selects only distinct values by default. To allows duplicate values, use UNION ALL
  - UNION ALL operator is used to combine the results sets of 2 or more select statements.
  - It returns all rows from the query and it does not remove duplicate rows between the various select statements.
    - **Syntax**:
      - SELECT CN'S FROM TN1
        UNION ALL
        SELECT CN'S FROM TN2**;**
    - **Example**:
      - SELECT NAME FROM STUDENTS
        UNION ALL
        SELECT NAME FROM STUDENTS2**;**



```
SELECT *
FROM facebook

UNION ALL

SELECT *
FROM linkedin;
```

facebook

Name	Location
Alex	San Francisco
Matt	San Francisco
Zeke	Los Angeles

linkedin

Name	Location
Matt	San Francisco
Ruby	San Francisco
Zeke	Los Angeles

Result

Name	Location
Alex	San Francisco
Matt	San Francisco
Matt	San Francisco
Ruby	San Francisco
Zeke	Los Angeles
Zeke	Los Angeles

- **SELECT INTO**
    - SELECT INTO statement is used to create table from an existing table by copying the existing table's columns.
        - Copy all columns into a new Table

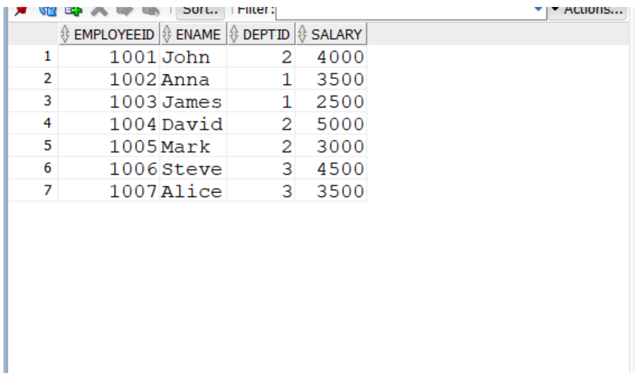
          Syntax:

          SELECT *
          INTO new_Table [IN DB]
          FROM old_table
          Where condition;

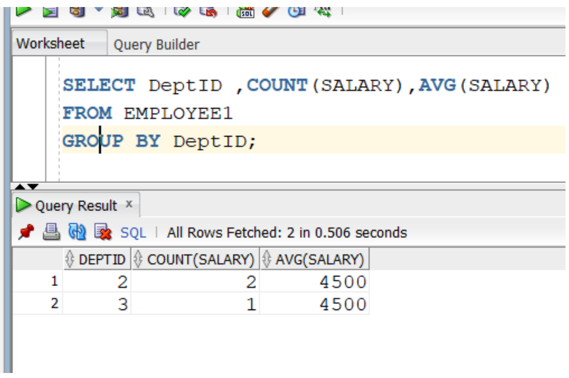
          Ex:

          SELECT *
          INTO VCTC [IN DatabaseName]
          FROM Velocity;

- **Group By**
    - The GROUP BY statement groups that have the same values into summary rows, like "find the number of customers in each country".
    - The GROUP BY statement is often used with aggregate functions like COUNT() , MAX(), SUM(), AVG() , MIN()
    - **Syntax:**
        - SELECT CN's
          FROM TN
          WHERE condition
          GROUP BY CN's**;**
    - **Example**
        - SELECT DeptID ,COUNT(SALARY),AVG(SALARY)
          FROM EMPLOYEE1
          GROUP BY DeptID;



	EMPLOYEEID	ENAME	DEPTID	SALARY
1	1001	John	2	4000
2	1002	Anna	1	3500
3	1003	James	1	2500
4	1004	David	2	5000
5	1005	Mark	2	3000
6	1006	Steve	3	4500
7	1007	Alice	3	3500

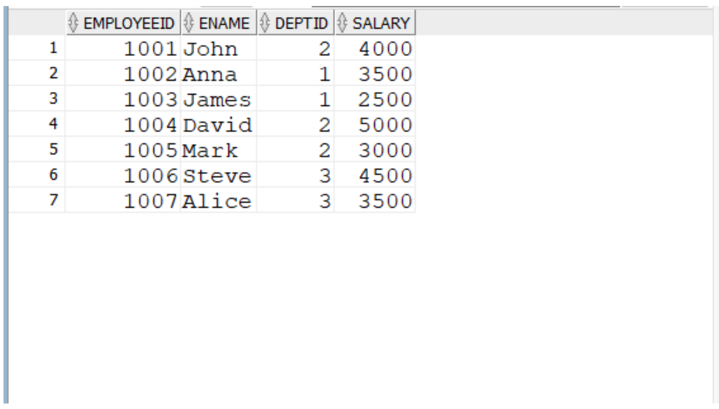


```
SELECT DeptID ,COUNT(SALARY),AVG(SALARY)
FROM EMPLOYEE1
GROUP BY DeptID;
```

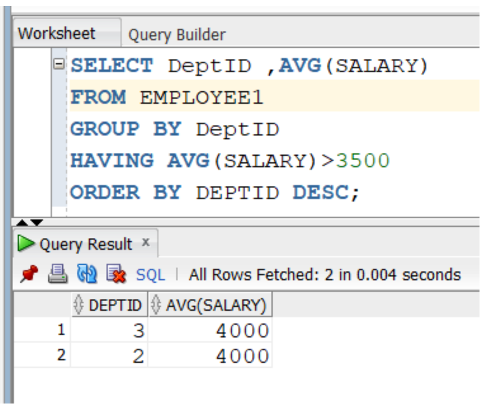
All Rows Fetched: 2 in 0.506 seconds

	DEPTID	COUNT(SALARY)	AVG(SALARY)
1	2	2	4500
2	3	1	4500

- **Having Clause**
  - The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.
  - **Syntax**
    - SELECT CN's
      FROM TN
      WHERE condition
      GROUP BY CN's
      HAVING condition
      ORDER BY CN's**;**
  - **Example**:
    - SELECT DeptID ,AVG(SALARY)
      FROM EMPLOYEE1
      GROUP BY DeptID
      HAVING AVG (SALARY)>3500
      ORDER BY DEPTID DESC**;**



	EMPLOYEEID	ENAME	DEPTID	SALARY
1	1001	John	2	4000
2	1002	Anna	1	3500
3	1003	James	1	2500
4	1004	David	2	5000
5	1005	Mark	2	3000
6	1006	Steve	3	4500
7	1007	Alice	3	3500



```
SELECT DeptID ,AVG(SALARY)
FROM EMPLOYEE1
GROUP BY DeptID
HAVING AVG(SALARY)>3500
ORDER BY DEPTID DESC;
```

	DEPTID	AVG(SALARY)
1	3	4000
2	2	4000

- **SQL Table Constraints**
  - Constraint can be specified when the table is created with CREATE TABLE statement, or after the table is created with the ALTER TABLE.
  - **Syntax:**
    - CREATE TABLE TN **(**
      Column1 datatype **constraint**,
      Column2 datatype **constraint**,
      Column3 datatype **constraint**,
      ……
      );
  - **This constraint are as follow:**
    - **NOT NULL:**

      Ensure that a column cannot have a NULL value.

    - **UNIQUE:**
      - Ensure that all values in a column are different.
      - Both the unique and primary key constraints provide a guarantee for **uniqueness** for a column or set of columns.

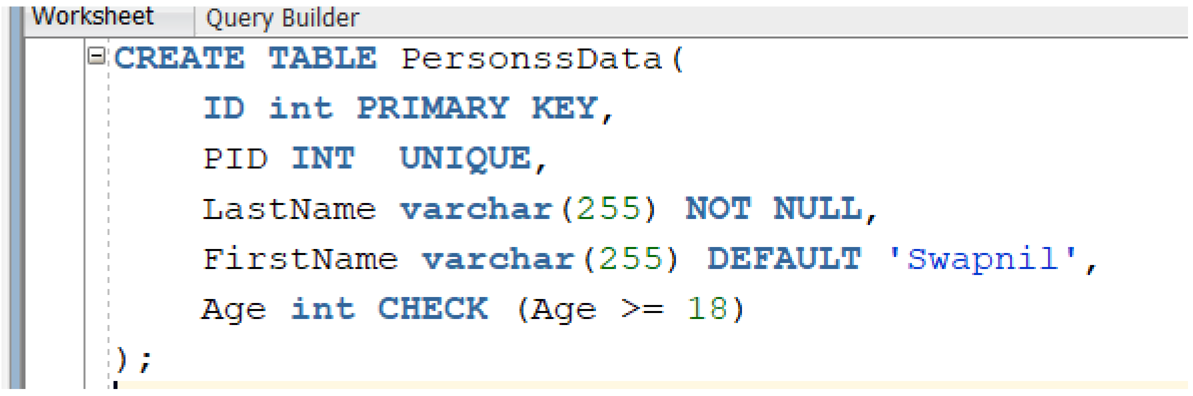
- A primary key constraint automatically has a UNIQUE constraint.
- However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

▪ **PRIMARY KEY:**
  - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table.
  - PRIMARY KEY must contain UNIQUE values, and cannot contain NULL values.
  - A table can have only one PRIMARY KEY.

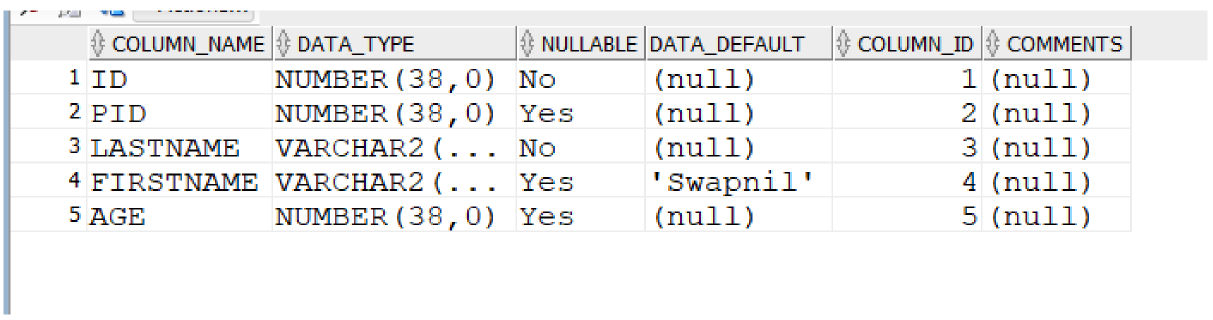
▪ **CHECK:**
  - Ensures that the values in column satisfy a specific condition.
  - It is used to limit the value range that can be placed in a column.
  - If you define a CHECK constraint on a column it will allow only certain values for this column.
  - If you define a CHECK constraint on a table it can limit values in certain columns based on values in other columns in the row.

▪ **DEFAULT:**
  - Sets a default value for a column if no value is specified.
  - The default value will be added to all new records, if no other value is specified.

Example on above Constraint:



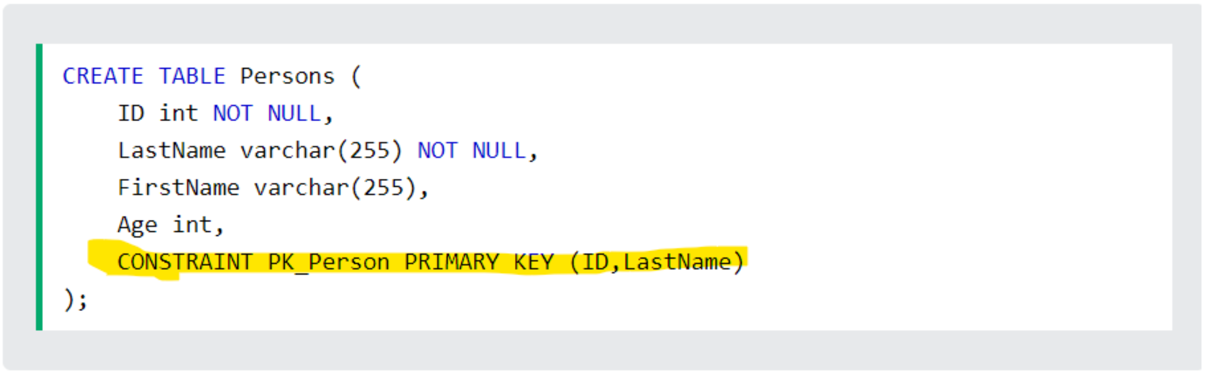
```
CREATE TABLE PersonssData(
      ID int PRIMARY KEY,
      PID INT  UNIQUE,
      LastName varchar(255) NOT NULL,
      FirstName varchar(255) DEFAULT 'Swapnil',
      Age int CHECK (Age >= 18)
);
```



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ID	NUMBER(38,0)	No	(null)	1	(null)
2	PID	NUMBER(38,0)	Yes	(null)	2	(null)
3	LASTNAME	VARCHAR2(...	No	(null)	3	(null)
4	FIRSTNAME	VARCHAR2(...	Yes	'Swapnil'	4	(null)
5	AGE	NUMBER(38,0)	Yes	(null)	5	(null)

NOTE: there is always only one primary key in Tables, if we look somewhere there is two primary keys in table then this concept is called as **Composite KEY.**

Ex:

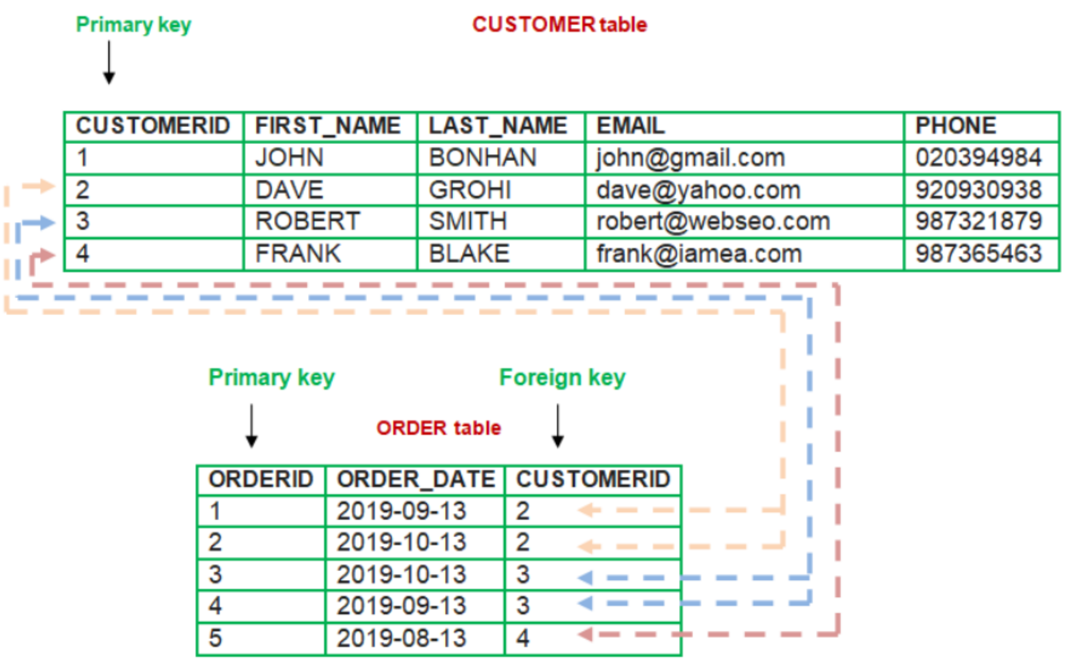


```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

Note: In the example above there is only ONE `PRIMARY KEY` (PK_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName).

- **FOREIGN KEY:**
  - Prevents actions that would destroy link between tables.
  - A FOREIGN KEY is a field or (collection of fields) in one table that refers the PRIMARY KEY in another table.
  - The table with FOREIGN KEY is called the child table, and the table with PRIMARY KEY is called referenced or parent table.

A primary key-foreign key relationship



CUSTOMERID	FIRST_NAME	LAST_NAME	EMAIL	PHONE
1	JOHN	BONHAN	john@gmail.com	020394984
2	DAVE	GROHI	dave@yahoo.com	920930938
3	ROBERT	SMITH	robert@webseo.com	987321879
4	FRANK	BLAKE	frank@iamea.com	987365463

ORDERID	ORDER_DATE	CUSTOMERID
1	2019-09-13	2
2	2019-10-13	2
3	2019-10-13	3
4	2019-09-13	3
5	2019-08-13	4

Syntax/Example

```
CREATE TABLE ORDER (
 ORDERID int PRIMARY KEY,
 ORDER_DATE DATE NOT NULL,
 CUSTOEMRID INT NOT NULL,
 CUSTOMERID INT FOREIGN KEY REFERENCES CUSTOMER(CUSTOMERID)
 );
```

- **CREATE INDEX:**
  - It is used to create and retrieve data from the database very quickly.
  - **CREATE INDEX** statement s used to create indexes in table.
  - **Syntax:**
    - CREATE INDEX index_name
      ON table_name *(*column1, column2, ...*);*
  - **Example:**
    - CREATE INDEX index1
      ON EMPLOYEE1 (DEPTID, ENAME);

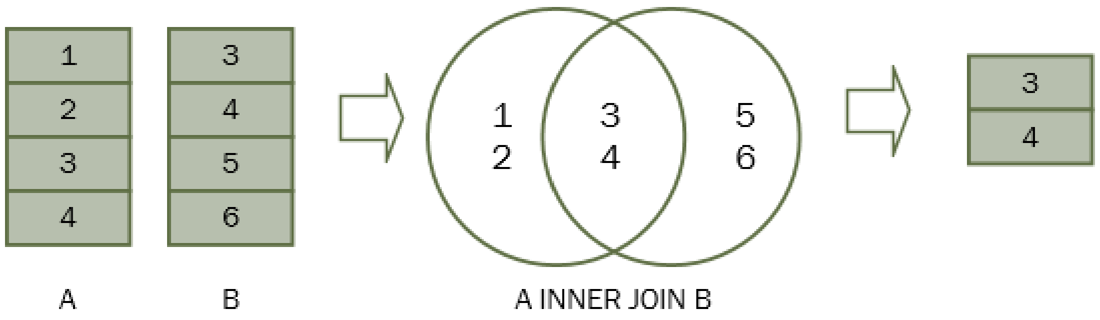
- **Views**
  - A view is a virtual table based on the result-set of an SQL statement.
  - A View contains rows and columns just look like a real table. The fields in a view are fields from one or more Real Tables in database.
  - Syntax:
    - CREATE VIEW view_name AS
      SELECT CN"S
      FROM TN
      WHERE condition**;**
  - **Example:**
    - CREATE VIEW Employee_View AS
       SELECT * FROM EMPLOYEE1;
    - CREATE VIEW Employee_View AS
       SELECT ID, NAME, SALARY
      FROM EMPLOYEE1
      WHERE SALARY > 15000;

Primary Key	**Foreign Key**	**Unique Key**
The PRIMARY KEY constraint uniquely identifies each record in a table. Primary keys must contain UNIQUE values	A FOREIGN KEY is a key used to link two tables together. A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.	The UNIQUE constraint ensures that all values in a column are different.
Primary key cannot have a NULL value.	Foreign key can accept multiple null value.	Unique Constraint may have a NULL value.
Each table can have only one primary key.	We can have more than one foreign key in a table.	Each table can have more than one Unique Constraint.
Primary key is clustered index	Foreign keys do not automatically create an index, clustered or non-clustered	Unique key is a unique non-clustered index

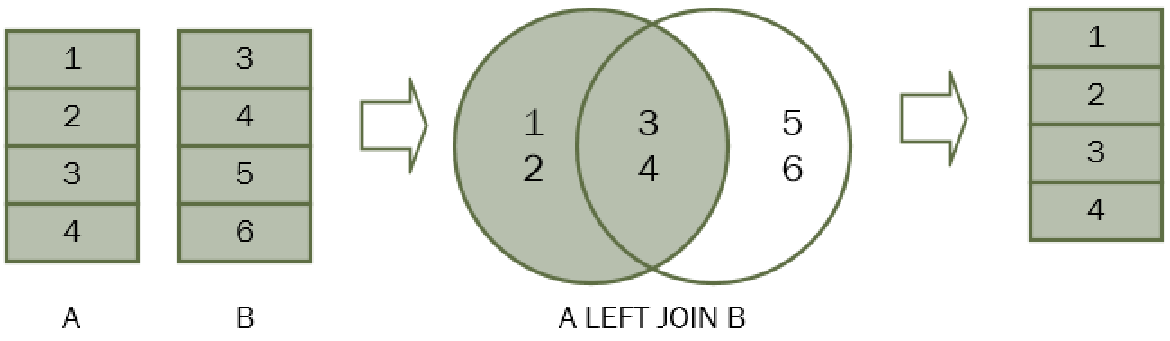
- **JOINS**
  - A join clause is used to combine rows from two or more tables, based on a related column between them.

- **Different types in SQL JOINS**
  - **INNER JOIN**
    - It returns that have matching values in both tables.
    - **Syntax (T1 = TABLE1 AND T2 = TABLE2)**
      - SELECT CN's
        FROM T1
        INNER JOIN T2
        ON T1.CN = T2.CN**'s;**
      - **HERE , T1.CN IS PRIMARY KEY AND T2.CN IS FOREIGN KEY**

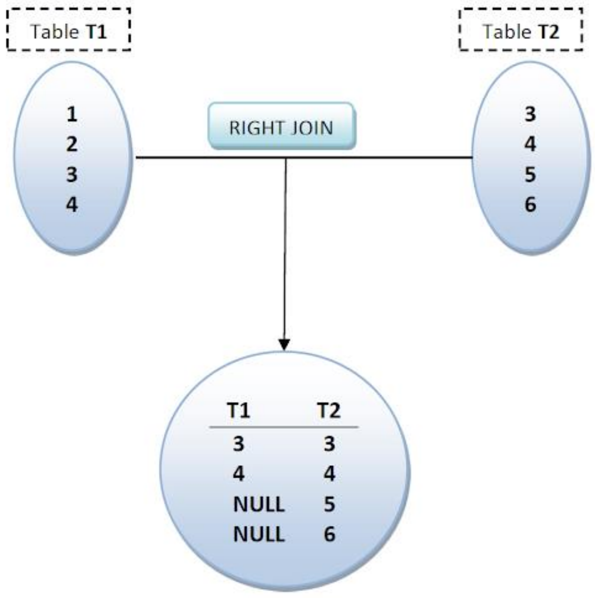
    Note: Example of this we will see after its theory



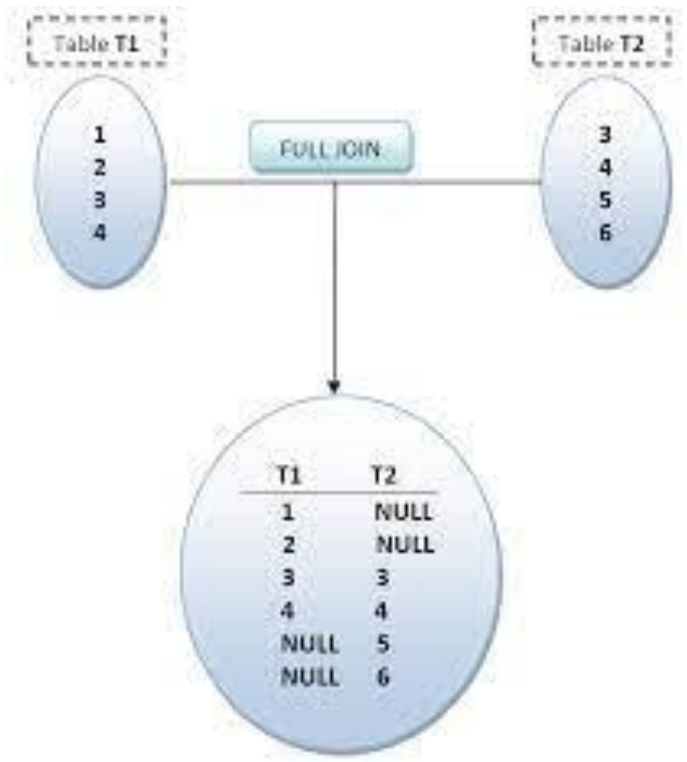
  - **LEFT JOIN / LEFT OUTER JOIN**
    - It returns all records from the left table, and match records from the right table.
      - **Syntax:**
        SELECT CN's
        FROM T1
        LEFT JOIN T2
        ON T1.CN = T2.CN**'s;**



- **RIGHT JOIN / RIGHT OUTER JOIN**
  - Returns all records from the right table, and the matched records from the left table
  - In some database right join is called as right outer join.
    - **Syntax:**
      SELECT CN's
      FROM T1
      RIGHT JOIN T2
      ON T1.CN = T2.CN**'s;**

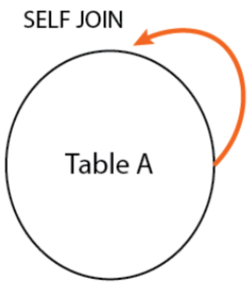


- **FULL JOIN / FULL OUTER JOIN**
  - Returns all records when there is a match in either left or right table.
    - **Syntax:**
      SELECT CN's
      FROM T1
      FULL JOIN T2
      ON T1.CN = T2.CN**'s;**



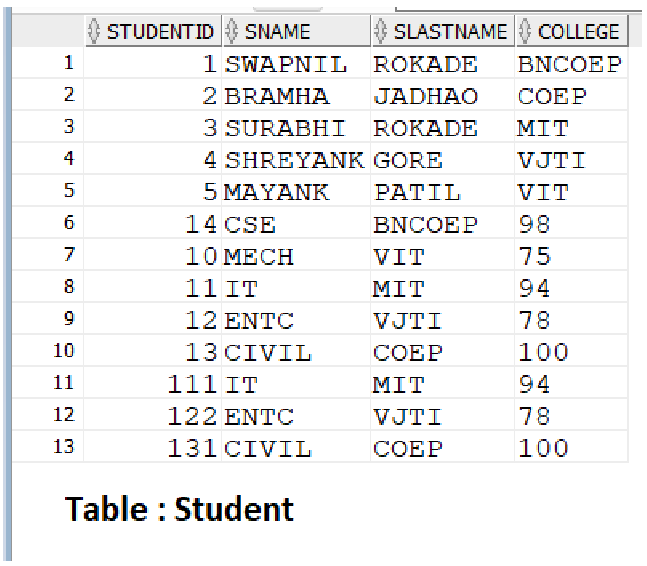
T1	T2
1	NULL
2	NULL
3	3
4	4
NULL	5
NULL	6

- **SELF JOIN**
  - A self-join is a regular join, but the table is joined with itself.



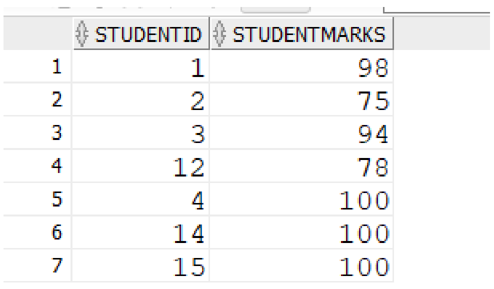
    - **Syntax:**
      SELECT CN's
      FROM TABLE1 T1, TABLE1 T2
      WHERE conditions**;**

- **Example on All Joins**
    - **First we create two tables and insert data into that table**



	STUDENTID	SNAME	SLASTNAME	COLLEGE
1	1	SWAPNIL	ROKADE	BNCOEP
2	2	BRAMHA	JADHAO	COEP
3	3	SURABHI	ROKADE	MIT
4	4	SHREYANK	GORE	VJTI
5	5	MAYANK	PATIL	VIT
6	14	CSE	BNCOEP	98
7	10	MECH	VIT	75
8	11	IT	MIT	94
9	12	ENTC	VJTI	78
10	13	CIVIL	COEP	100
11	111	IT	MIT	94
12	122	ENTC	VJTI	78
13	131	CIVIL	COEP	100

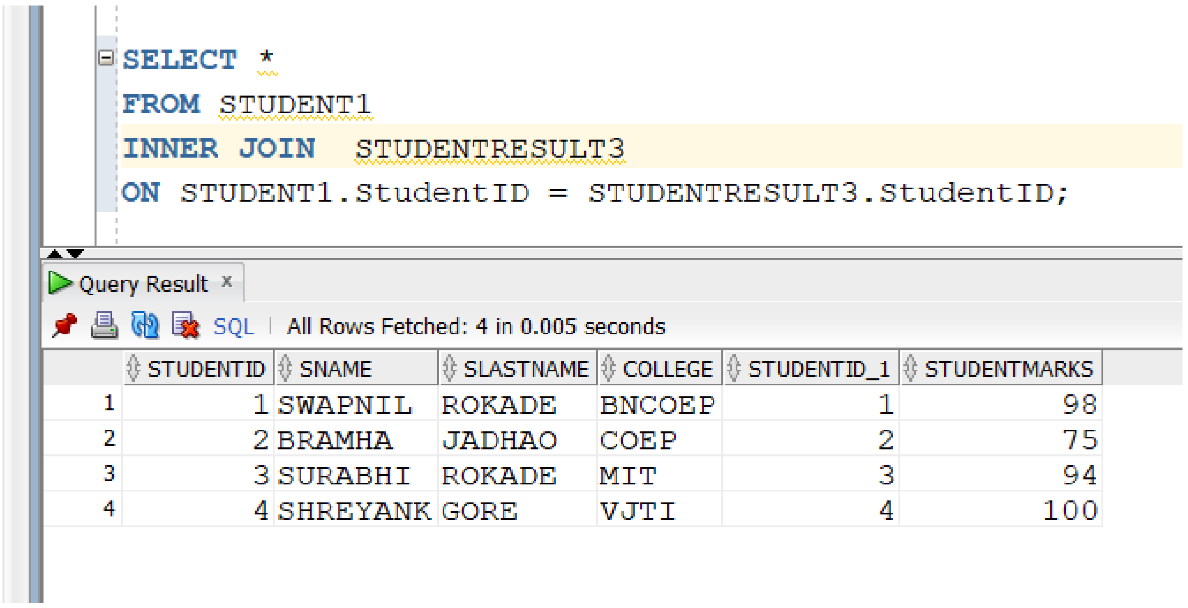
Table : Student



	STUDENTID	STUDENTMARKS
1	1	98
2	2	75
3	3	94
4	12	78
5	4	100
6	14	100
7	15	100

TABLE : STUDENTRESULT

- **INNER JOIN Example:**

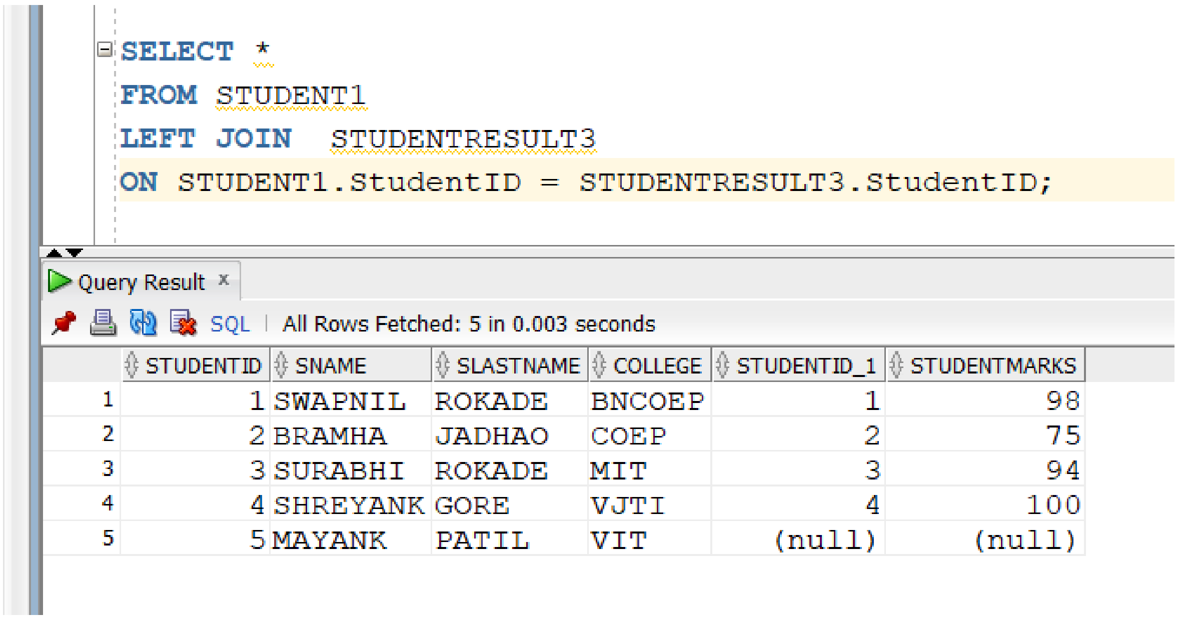


```
SELECT *
FROM STUDENT1
INNER JOIN  STUDENTRESULT3
ON STUDENT1.StudentID = STUDENTRESULT3.StudentID;
```

All Rows Fetched: 4 in 0.005 seconds

	STUDENTID	SNAME	SLASTNAME	COLLEGE	STUDENTID_1	STUDENTMARKS
1	1	SWAPNIL	ROKADE	BNCOEP	1	98
2	2	BRAMHA	JADHAO	COEP	2	75
3	3	SURABHI	ROKADE	MIT	3	94
4	4	SHREYANK	GORE	VJTI	4	100

- **LEFT JOIN Example:**



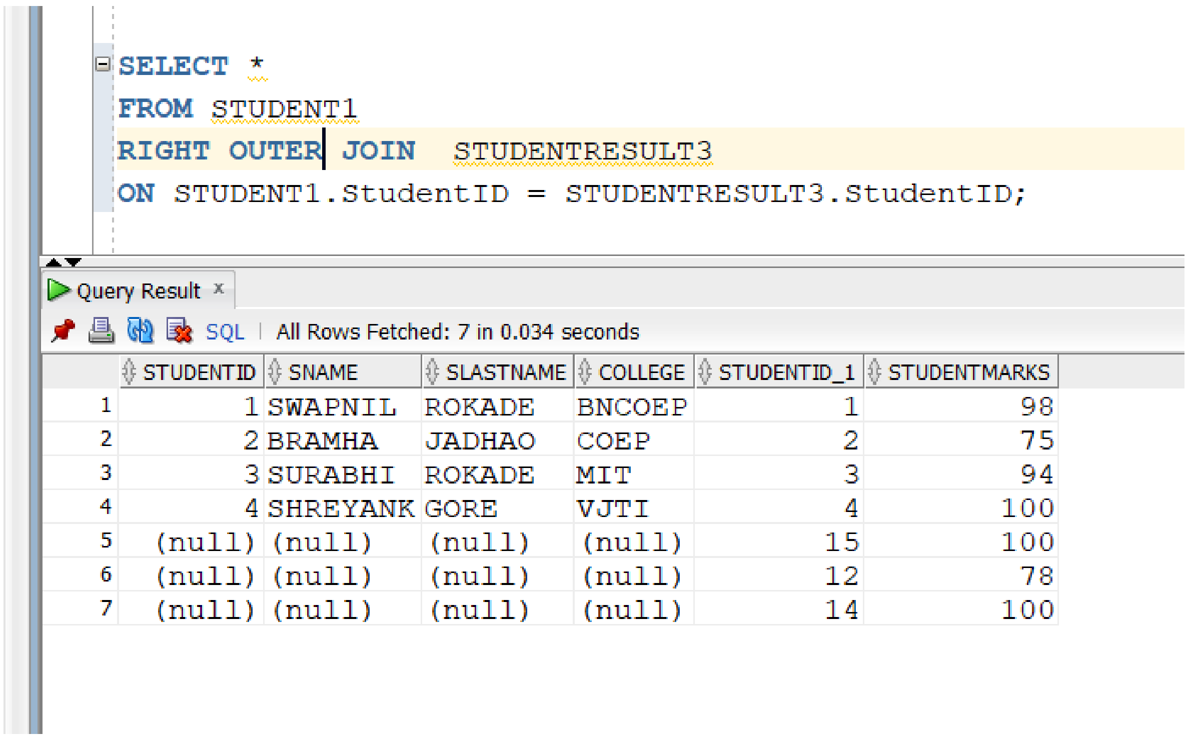
```
SELECT *
FROM STUDENT1
LEFT JOIN  STUDENTRESULT3
ON STUDENT1.StudentID = STUDENTRESULT3.StudentID;
```

Query Result

All Rows Fetched: 5 in 0.003 seconds

	STUDENTID	SNAME	SLASTNAME	COLLEGE	STUDENTID_1	STUDENTMARKS
1	1	SWAPNIL	ROKADE	BNCOEP	1	98
2	2	BRAMHA	JADHAO	COEP	2	75
3	3	SURABHI	ROKADE	MIT	3	94
4	4	SHREYANK	GORE	VJTI	4	100
5	5	MAYANK	PATIL	VIT	(null)	(null)

- **RIGHT JOIN Example:**



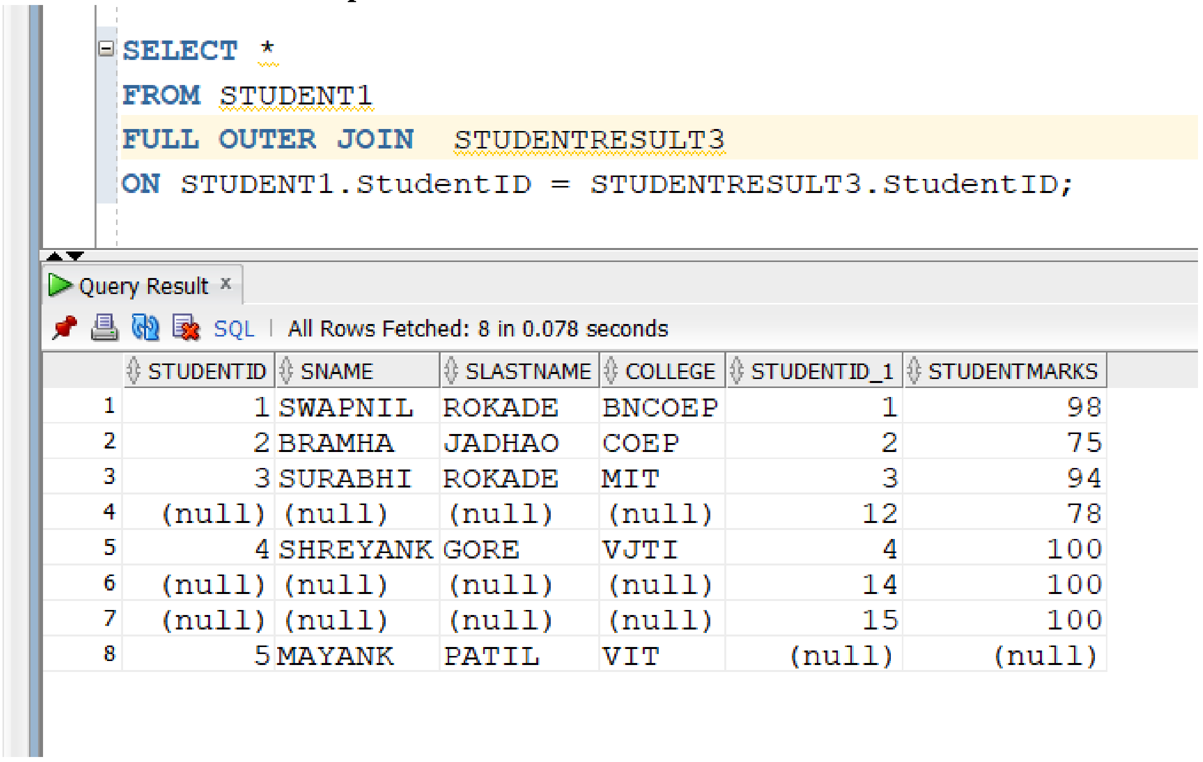
```
SELECT *
FROM STUDENT1
RIGHT OUTER JOIN  STUDENTRESULT3
ON STUDENT1.StudentID = STUDENTRESULT3.StudentID;
```

Query Result

All Rows Fetched: 7 in 0.034 seconds

	STUDENTID	SNAME	SLASTNAME	COLLEGE	STUDENTID_1	STUDENTMARKS
1	1	SWAPNIL	ROKADE	BNCOEP	1	98
2	2	BRAMHA	JADHAO	COEP	2	75
3	3	SURABHI	ROKADE	MIT	3	94
4	4	SHREYANK	GORE	VJTI	4	100
5	(null)	(null)	(null)	(null)	15	100
6	(null)	(null)	(null)	(null)	12	78
7	(null)	(null)	(null)	(null)	14	100

- **FULL JOIN Example:**



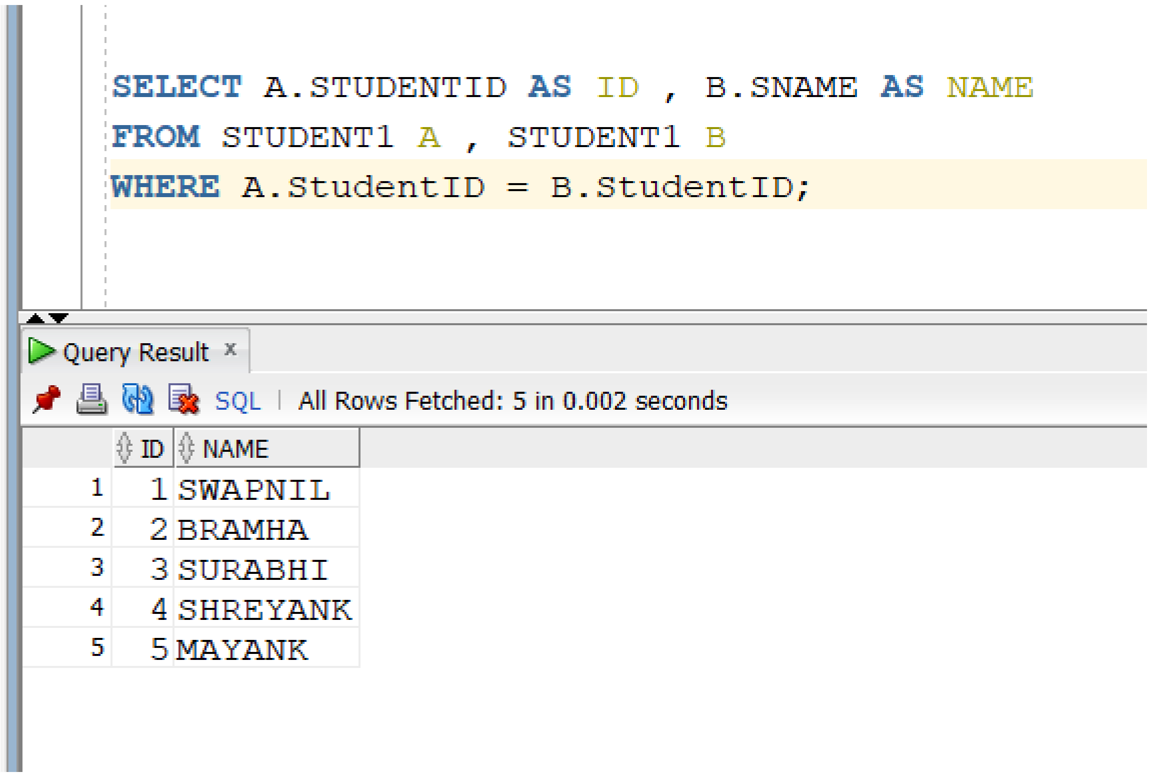
```
SELECT *
FROM STUDENT1
FULL OUTER JOIN  STUDENTRESULT3
ON STUDENT1.StudentID = STUDENTRESULT3.StudentID;
```

Query Result ×

SQL | All Rows Fetched: 8 in 0.078 seconds

	STUDENTID	SNAME	SLASTNAME	COLLEGE	STUDENTID_1	STUDENTMARKS
1	1	SWAPNIL	ROKADE	BNCOEP	1	98
2	2	BRAMHA	JADHAO	COEP	2	75
3	3	SURABHI	ROKADE	MIT	3	94
4	(null)	(null)	(null)	(null)	12	78
5	4	SHREYANK	GORE	VJTI	4	100
6	(null)	(null)	(null)	(null)	14	100
7	(null)	(null)	(null)	(null)	15	100
8	5	MAYANK	PATIL	VIT	(null)	(null)

- **SELF JOIN Example:**



```
SELECT A.STUDENTID AS ID , B.SNAME AS NAME
FROM STUDENT1 A , STUDENT1 B
WHERE A.StudentID = B.StudentID;
```

Query Result ×

SQL | All Rows Fetched: 5 in 0.002 seconds

	ID	NAME
1	1	SWAPNIL
2	2	BRAMHA
3	3	SURABHI
4	4	SHREYANK
5	5	MAYANK

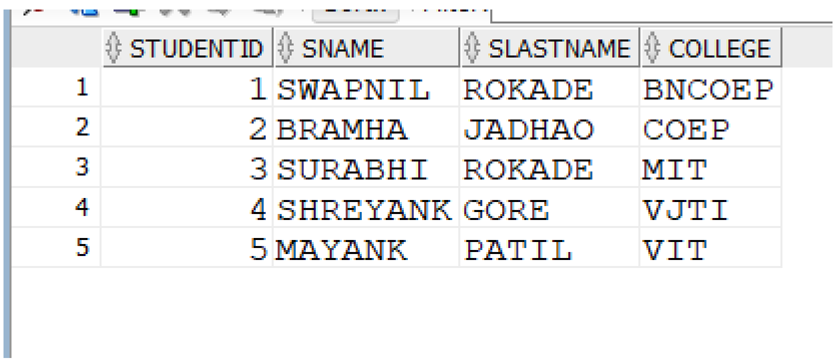
❖ Stored Procedures

a. A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.
b. So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.
c. You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.
d. **SYNTAX**
   i.
   ```
   CREATE PROCEDURE procedure_name
   AS
   sql_statement
   GO;
   ```
   ii.
   ```
   EXEC procedure_name;
   ```
e. Ex;
   i.
   ```
   CREATE PROCEDURE SelectAllCustomers
   AS
   SELECT * FROM Customers
   GO;
   ```
   ii.
   ```
   EXEC SelectAllCustomers;
   ```

SOME QUESTIONS & ANSWERS

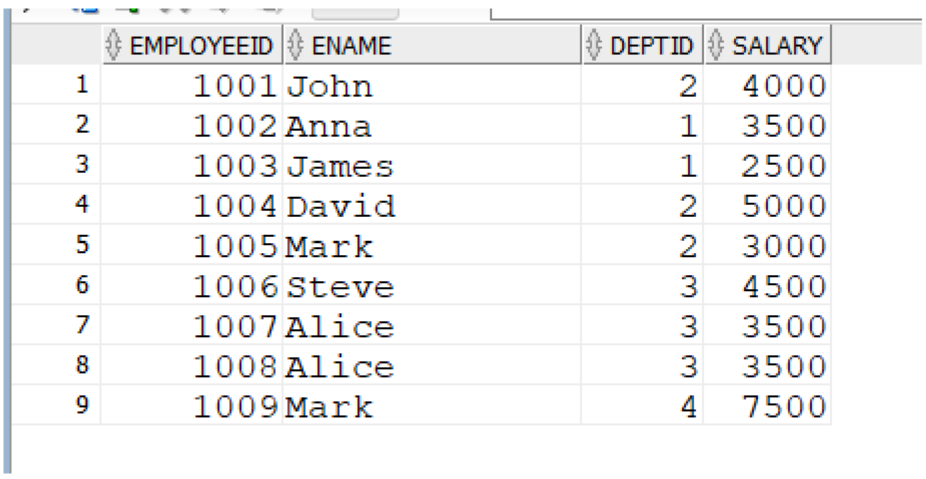
❖ **SOME QUESTIONS (USING FOLLOWING TABLES)**

- **Table STUDENT1**



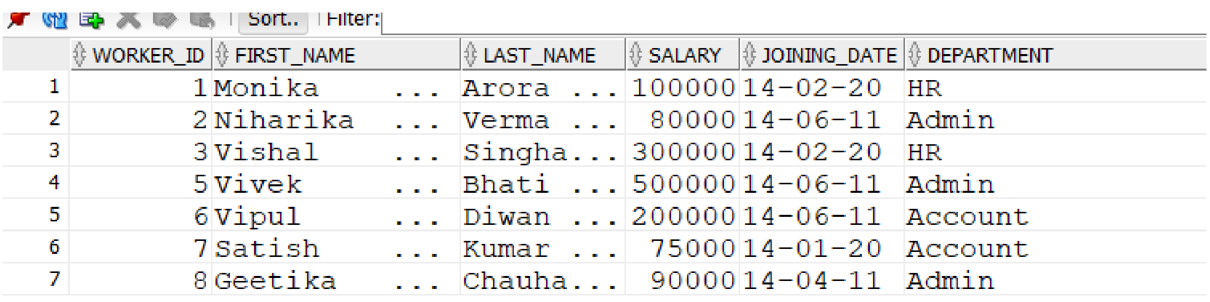
	STUDENTID	SNAME	SLASTNAME	COLLEGE
1	1	SWAPNIL	ROKADE	BNCOEP
2	2	BRAMHA	JADHAO	COEP
3	3	SURABHI	ROKADE	MIT
4	4	SHREYANK	GORE	VJTI
5	5	MAYANK	PATIL	VIT

- **Table EMMPLOYEE1**



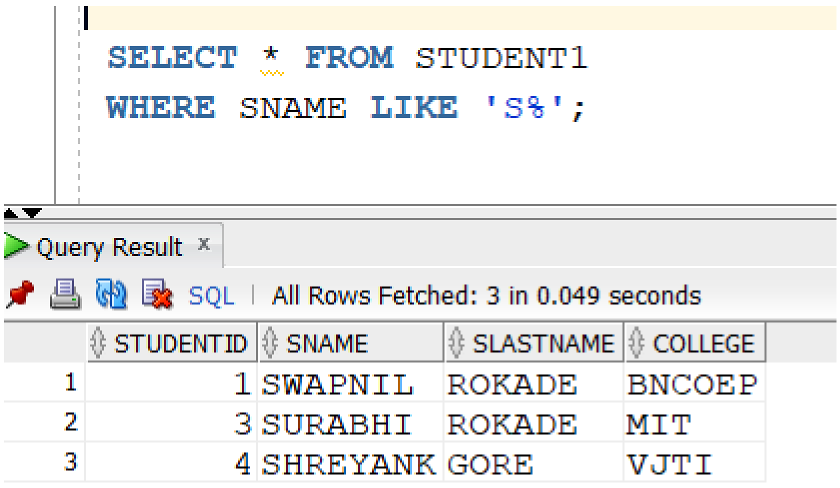
	EMPLOYEEID	ENAME	DEPTID	SALARY
1	1001	John	2	4000
2	1002	Anna	1	3500
3	1003	James	1	2500
4	1004	David	2	5000
5	1005	Mark	2	3000
6	1006	Steve	3	4500
7	1007	Alice	3	3500
8	1008	Alice	3	3500
9	1009	Mark	4	7500

- **Table WORKER**


Sort..
Filter:

	WORKER_ID	FIRST_NAME	LAST_NAME	SALARY	JOINING_DATE	DEPARTMENT
1	1	Monika ...	Arora ...	100000	14-02-20	HR
2	2	Niharika ...	Verma ...	80000	14-06-11	Admin
3	3	Vishal ...	Singha...	300000	14-02-20	HR
4	5	Vivek ...	Bhati ...	500000	14-06-11	Admin
5	6	Vipul ...	Diwan ...	200000	14-06-11	Account
6	7	Satish ...	Kumar ...	75000	14-01-20	Account
7	8	Geetika ...	Chauha...	90000	14-04-11	Admin

2. How to write a query to show the details of a student from Students table whose name start with S?



```
SELECT * FROM STUDENT1
WHERE SNAME LIKE 'S%';
```

Query Result ×

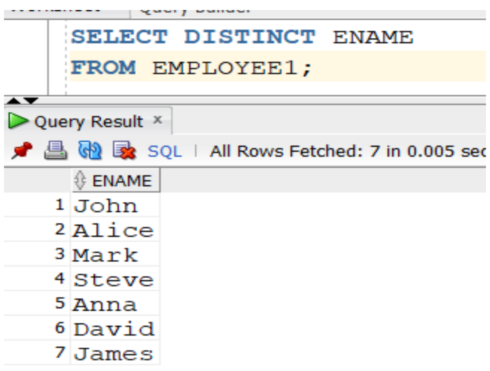
SQL | All Rows Fetched: 3 in 0.049 seconds

	STUDENTID	SNAME	SLASTNAME	COLLEGE
1	1	SWAPNIL	ROKADE	BNCOEP
2	3	SURABHI	ROKADE	MIT
3	4	SHREYANK	GORE	VJTI

3. What is the syntax to add a record to a table?
   a. Syntax:
   INSERT INTO TABLE
   VALUES (VALUE1, VALUE2, VALUE3);
   b. Example
   INSERT INTO STUDENT1
   VALUES (6,'MAYUR','NARSING','DYPATIL');

4. What is the syntax of GROUP BY in SQL?
   a. SELECT CN's
   FROM TN
   WHERE condition
   GROUP BY CN's;

5. Write a SQL SELECT query that only returns each name only once from a table?



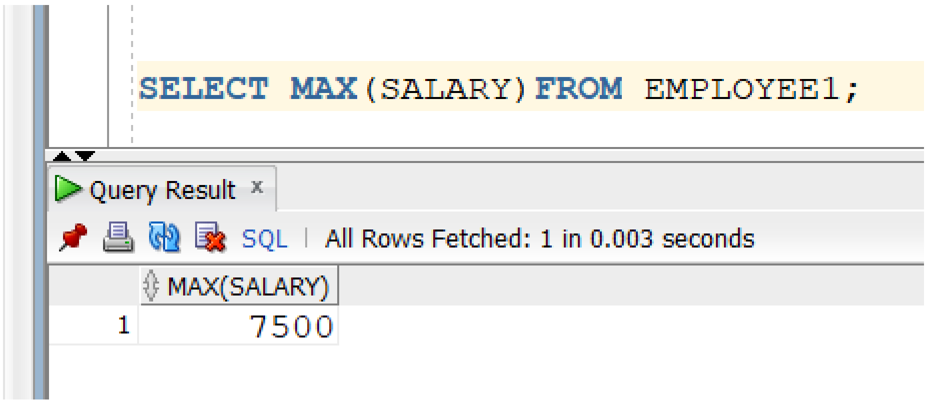
```
SELECT DISTINCT ENAME
FROM EMPLOYEE1;
```

Query Result ×

SQL | All Rows Fetched: 7 in 0.005 sec

	ENAME
1	John
2	Alice
3	Mark
4	Steve
5	Anna
6	David
7	James

6. Write an SQL query to get the first maximum salary of an employee from a table named employee1_table.

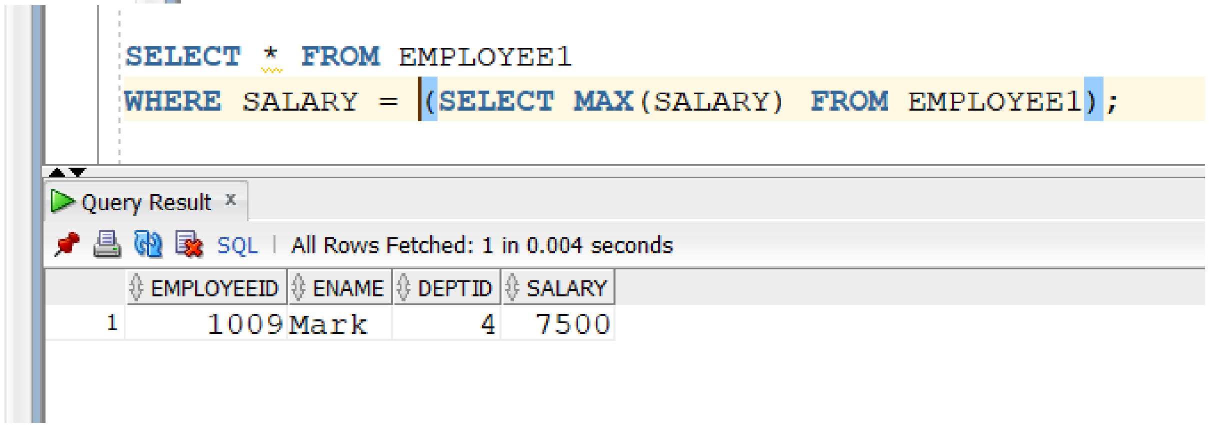


```
SELECT MAX(SALARY) FROM EMPLOYEE1;
```

Query Result ×

SQL | All Rows Fetched: 1 in 0.003 seconds

```
    MAX(SALARY)
1          7500
```



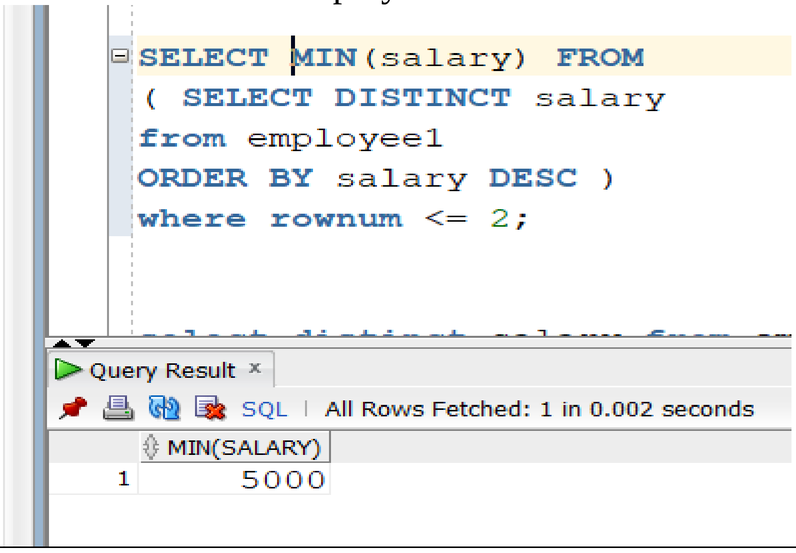
```
SELECT * FROM EMPLOYEE1
WHERE SALARY = (SELECT MAX(SALARY) FROM EMPLOYEE1);
```

Query Result ×

SQL | All Rows Fetched: 1 in 0.004 seconds

```
    EMPLOYEEID ENAME DEPTID SALARY
1         1009 Mark       4   7500
```

7. Write an SQL query to get the second maximum salary of an employee from a table named employee1 table.



```
SELECT MIN(salary) FROM
( SELECT DISTINCT salary
from employee1
ORDER BY salary DESC )
where rownum <= 2;
```

Query Result ×

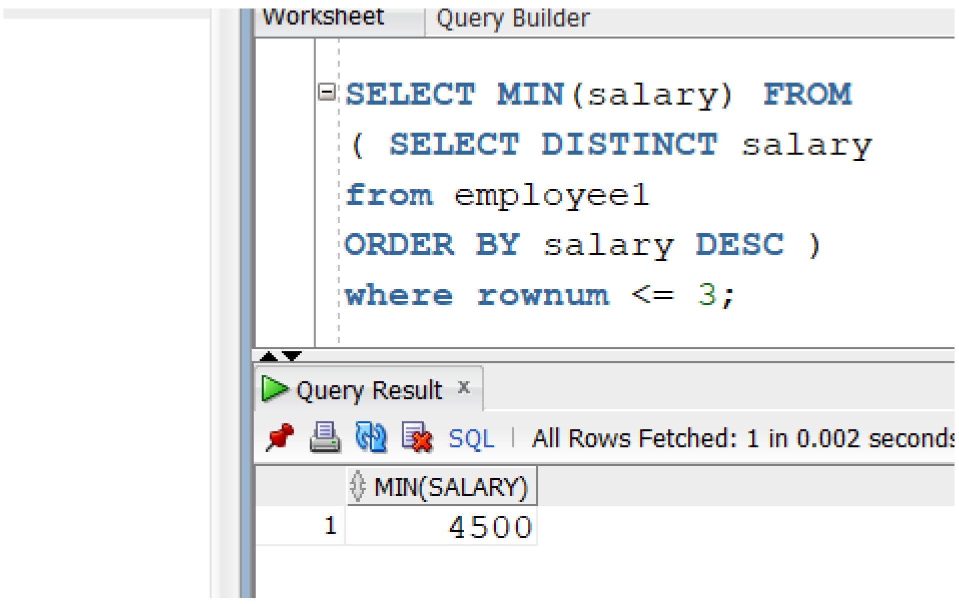
SQL | All Rows Fetched: 1 in 0.002 seconds

```
    MIN(SALARY)
1          5000
```

In order to calculate the second highest salary use rownum < = 2

In order to calculate the third highest salary use rownum <= 3

8. Write an SQL query to get the third maximum salary of an employee from a table named employee1 table.



Worksheet | Query Builder

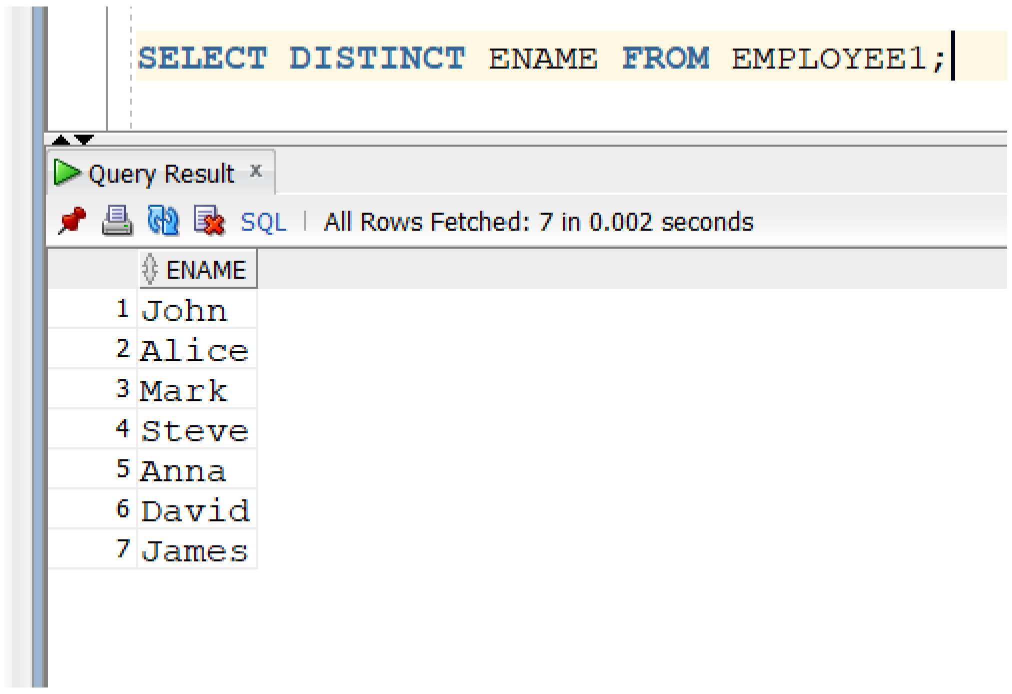
```
SELECT MIN(salary) FROM
( SELECT DISTINCT salary
from employee1
ORDER BY salary DESC )
where rownum <= 3;
```

Query Result

SQL | All Rows Fetched: 1 in 0.002 seconds

```
   MIN(SALARY)
1         4500
```

9. Write an SQL query to fetch unique values from a table?



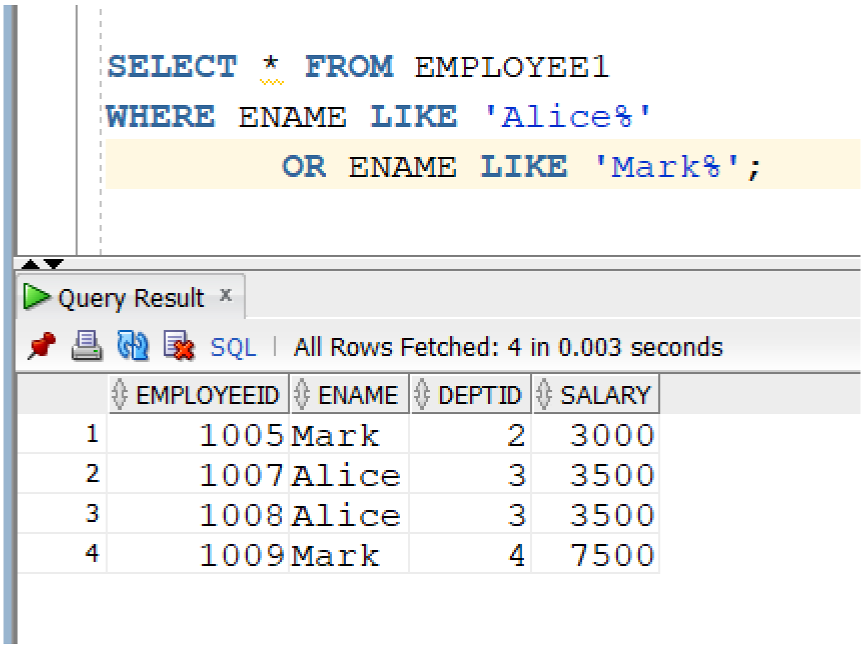
```
SELECT DISTINCT ENAME FROM EMPLOYEE1;
```

Query Result

SQL | All Rows Fetched: 7 in 0.002 seconds

```
  ENAME
1 John
2 Alice
3 Mark
4 Steve
5 Anna
6 David
7 James
```

10. Write an SQL query to fetch data from table whose name start with Alice, Mark?



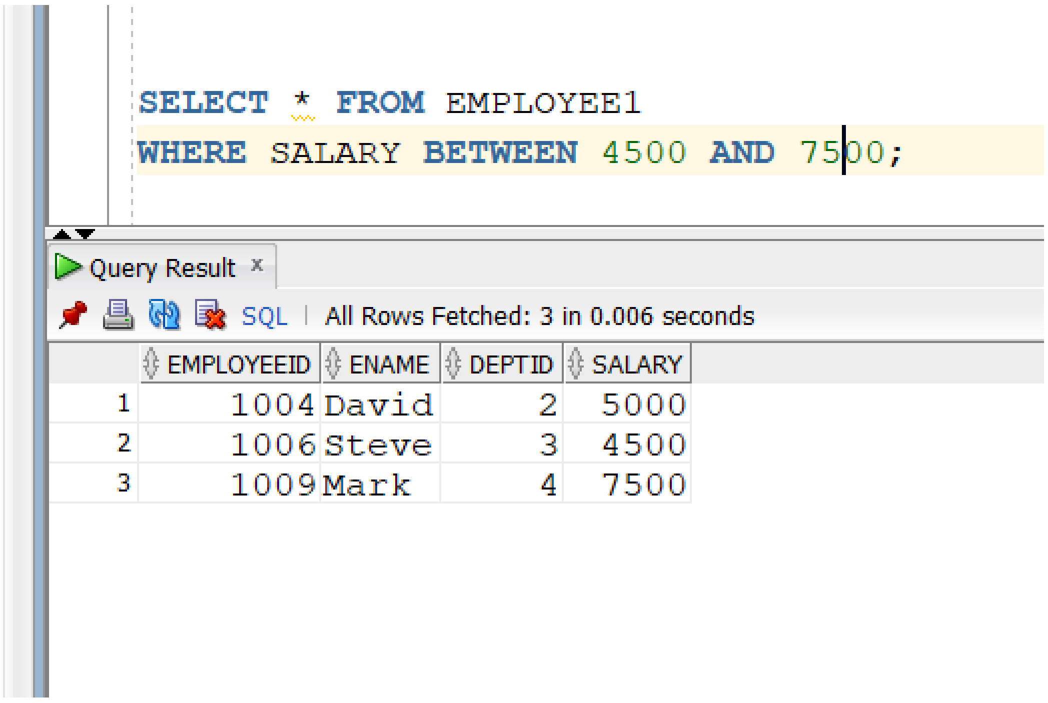
```
SELECT * FROM EMPLOYEE1
WHERE ENAME LIKE 'Alice%'
        OR ENAME LIKE 'Mark%';
```

Query Result ×

SQL | All Rows Fetched: 4 in 0.003 seconds

	EMPLOYEEID	ENAME	DEPTID	SALARY
1	1005	Mark	2	3000
2	1007	Alice	3	3500
3	1008	Alice	3	3500
4	1009	Mark	4	7500

11. Write an SQL query to print details of the Workers whose SALARY lies between 4500 and 7500 From EMPLOYEE table.



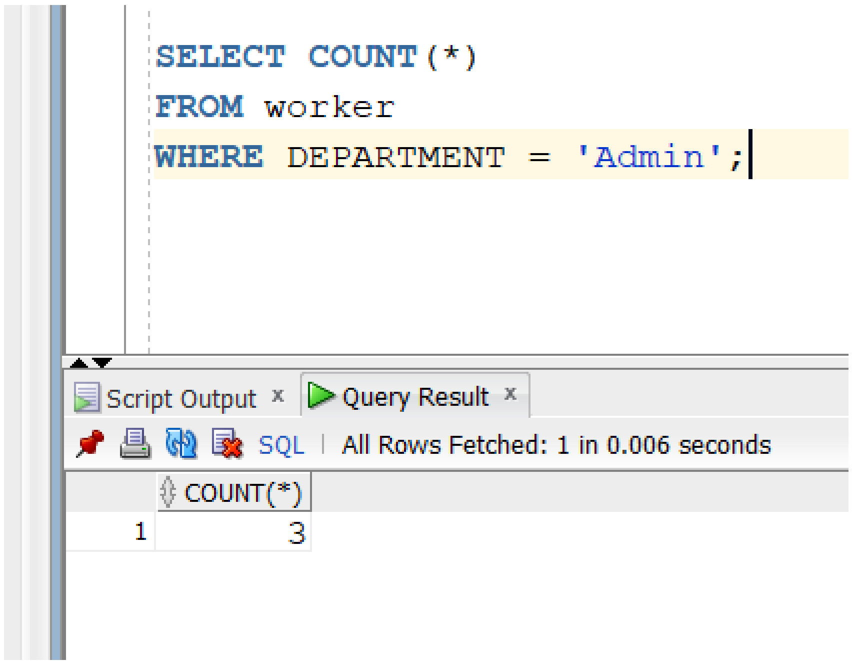
```
SELECT * FROM EMPLOYEE1
WHERE SALARY BETWEEN 4500 AND 7500;
```

Query Result ×

SQL | All Rows Fetched: 3 in 0.006 seconds

	EMPLOYEEID	ENAME	DEPTID	SALARY
1	1004	David	2	5000
2	1006	Steve	3	4500
3	1009	Mark	4	7500

12. Write an SQL query to fetch the count of employees working in the department ‘Admin’.
    a. Use Worker TABLE



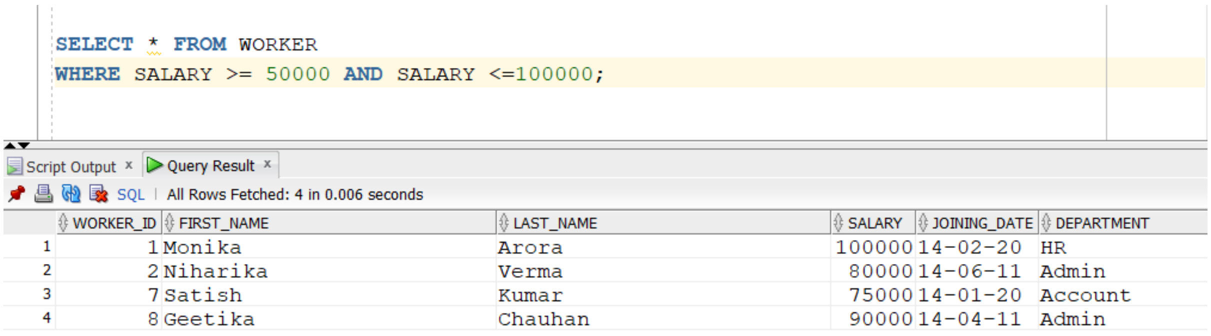
```
SELECT COUNT(*)
FROM worker
WHERE DEPARTMENT = 'Admin';
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 1 in 0.006 seconds

```
   COUNT(*)
1         3
```

13. Write an SQL query to fetch worker names with salaries >= 50000 and <= 100000.



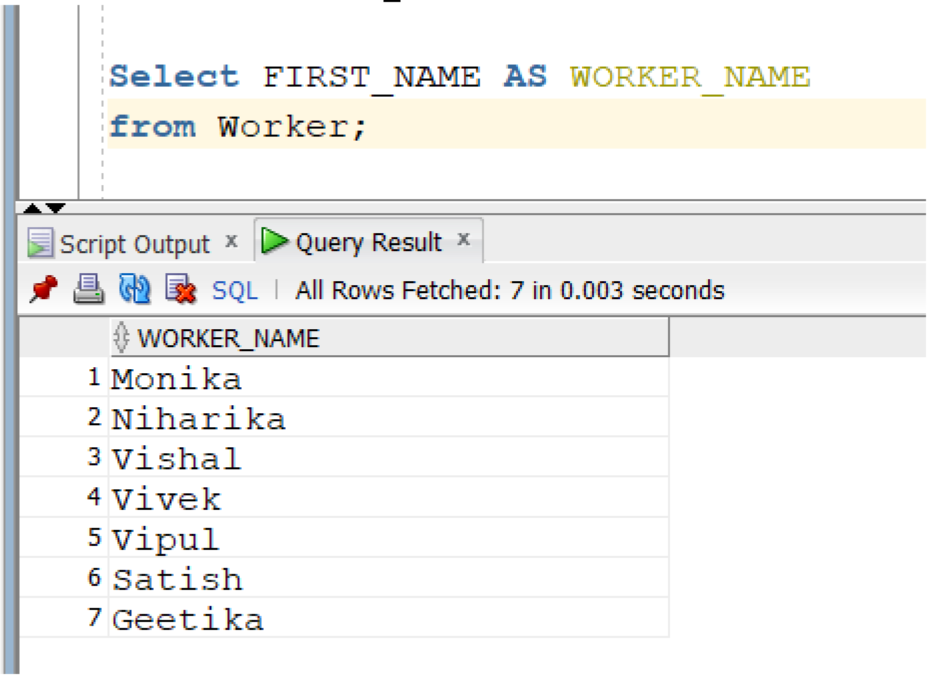
```
SELECT * FROM WORKER
WHERE SALARY >= 50000 AND SALARY <=100000;
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 4 in 0.006 seconds

```
  WORKER_ID FIRST_NAME   LAST_NAME   SALARY JOINING_DATE DEPARTMENT
1         1 Monika       Arora       100000 14-02-20     HR
2         2 Niharika     Verma        80000 14-06-11     Admin
3         7 Satish       Kumar        75000 14-01-20     Account
4         8 Geetika      Chauhan      90000 14-04-11     Admin
```

14. Write an SQL query to fetch "FIRST_NAME" from Worker table using the alias name as <WORKER_NAME>.



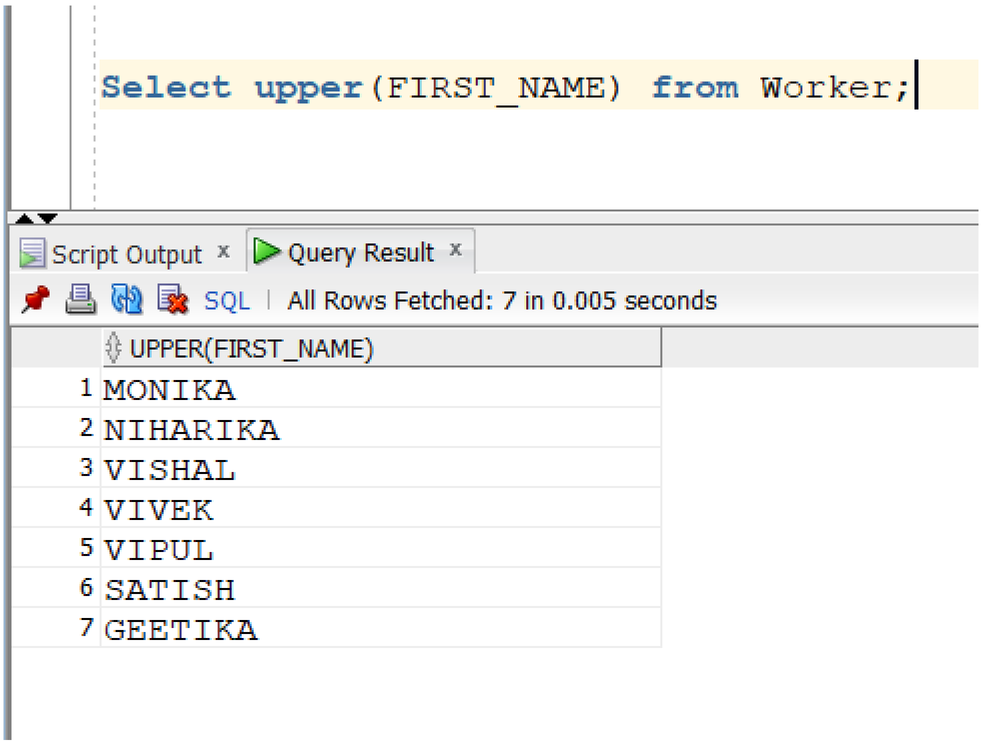
```
Select FIRST_NAME AS WORKER_NAME
from Worker;
```

```
All Rows Fetched: 7 in 0.003 seconds

  WORKER_NAME
1 Monika
2 Niharika
3 Vishal
4 Vivek
5 Vipul
6 Satish
7 Geetika
```

15. Write an SQL query to fetch "FIRST_NAME" from Worker table in upper case

a. **Note: We Can Use Some String Functions Like String In Java As Follow LIKE upper, lower, replace etc.**



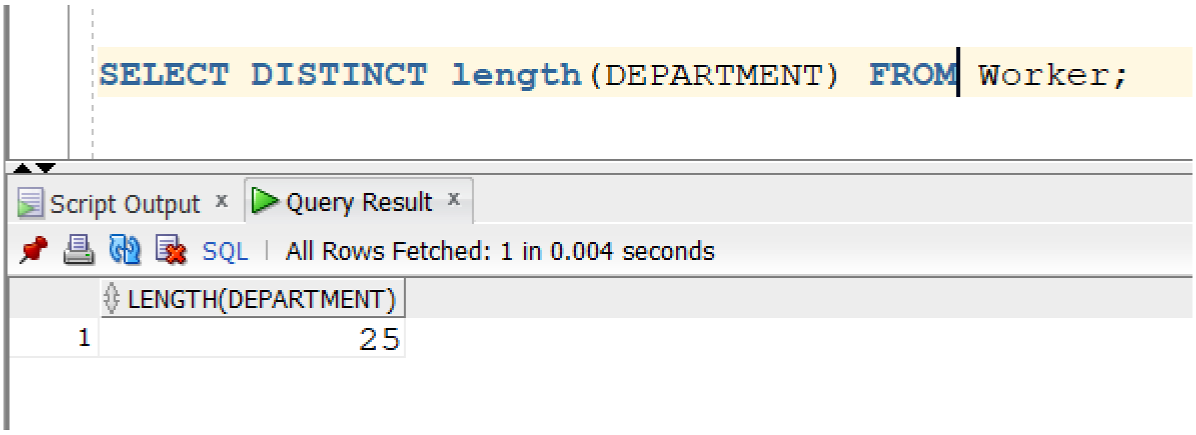
```
Select upper(FIRST_NAME) from Worker;
```

```
All Rows Fetched: 7 in 0.005 seconds

  UPPER(FIRST_NAME)
1 MONIKA
2 NIHARIKA
3 VISHAL
4 VIVEK
5 VIPUL
6 SATISH
7 GEETIKA
```

16. Write an SQL query that fetches the unique values of DEPARTMENT from Worker table and prints its length.

a. Here, total characters in length will calculate from worker table.

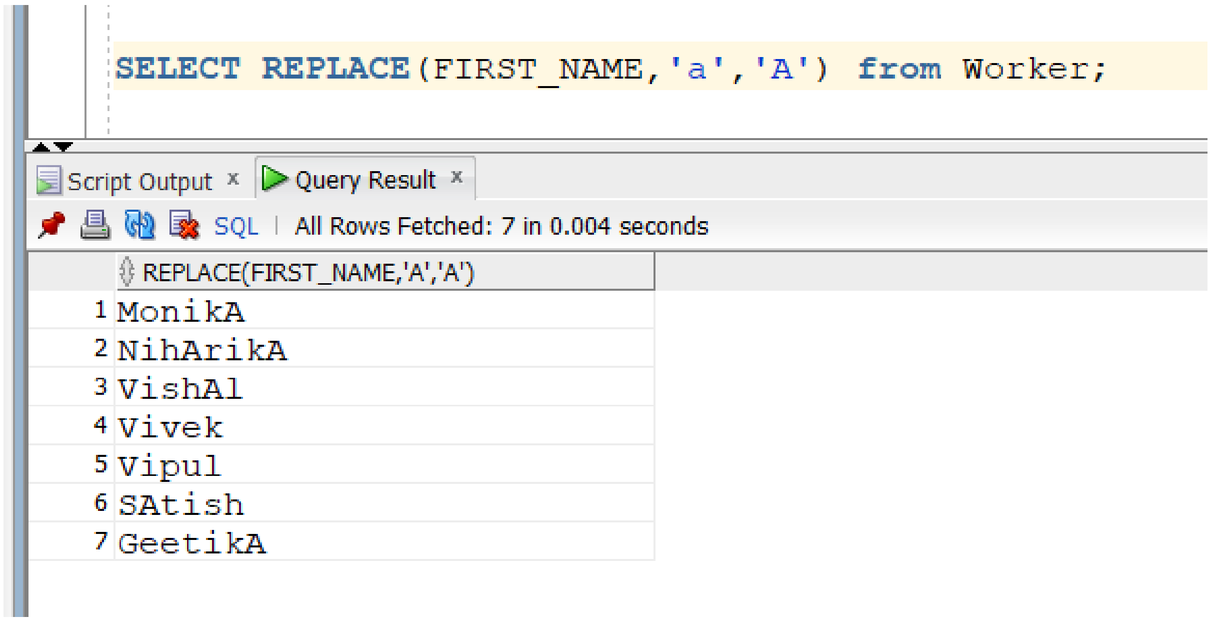


```
SELECT DISTINCT length(DEPARTMENT) FROM Worker;
```

All Rows Fetched: 1 in 0.004 seconds

	LENGTH(DEPARTMENT)
1	25

17. Write an SQL query to print the FIRST_NAME from **Worker** table after **replacing 'a' with 'A'.**



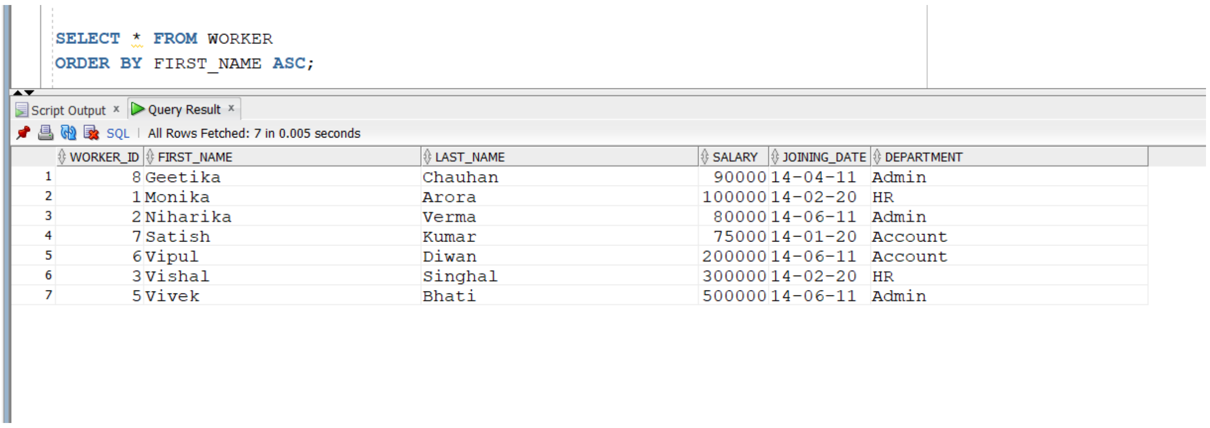
```
SELECT REPLACE(FIRST_NAME,'a','A') from Worker;
```

All Rows Fetched: 7 in 0.004 seconds

	REPLACE(FIRST_NAME,'A','A')
1	MonikA
2	NihArikA
3	VishAl
4	Vivek
5	Vipul
6	SAtish
7	GeetikA

18. Write an SQL query to print all Worker details from the Worker table order by FIRST_NAME **Ascending**.

ASCENDING

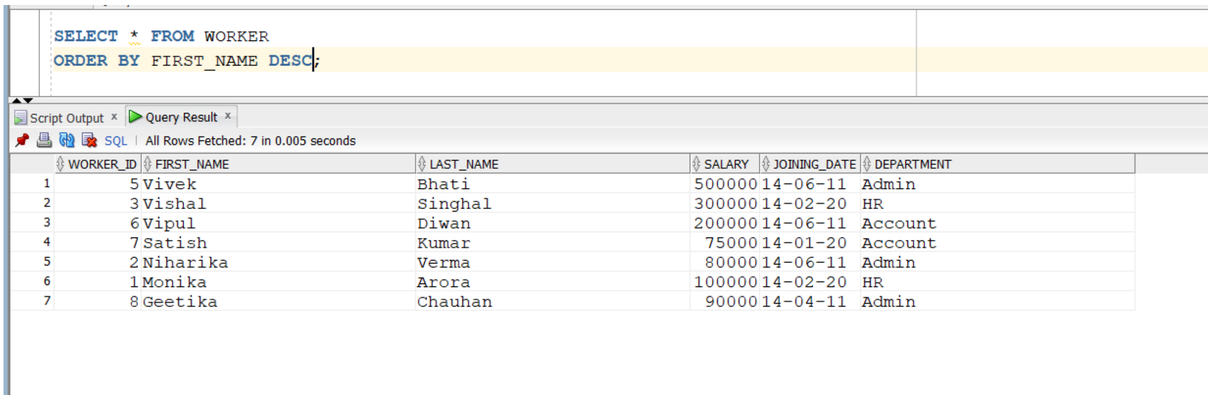


```
SELECT * FROM WORKER
ORDER BY FIRST_NAME ASC;
```

All Rows Fetched: 7 in 0.005 seconds

```
   WORKER_ID FIRST_NAME  LAST_NAME   SALARY JOINING_DATE DEPARTMENT
1          8 Geetika     Chauhan      90000 14-04-11     Admin
2          1 Monika      Arora       100000 14-02-20     HR
3          2 Niharika    Verma        80000 14-06-11     Admin
4          7 Satish      Kumar        75000 14-01-20     Account
5          6 Vipul       Diwan       200000 14-06-11     Account
6          3 Vishal      Singhal     300000 14-02-20     HR
7          5 Vivek       Bhati       500000 14-06-11     Admin
```

DESCENDING

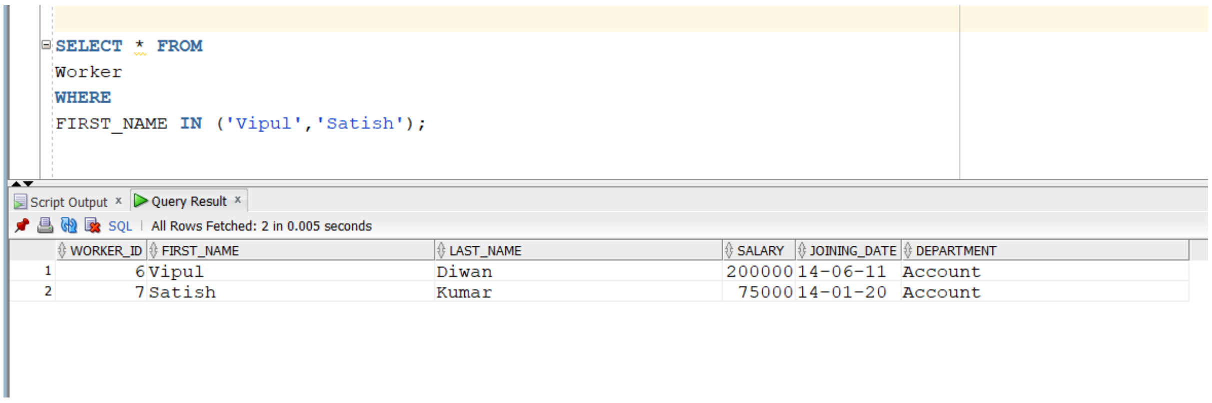


```
SELECT * FROM WORKER
ORDER BY FIRST_NAME DESC;
```

All Rows Fetched: 7 in 0.005 seconds

```
   WORKER_ID FIRST_NAME  LAST_NAME   SALARY JOINING_DATE DEPARTMENT
1          5 Vivek       Bhati       500000 14-06-11     Admin
2          3 Vishal      Singhal     300000 14-02-20     HR
3          6 Vipul       Diwan       200000 14-06-11     Account
4          7 Satish      Kumar        75000 14-01-20     Account
5          2 Niharika    Verma        80000 14-06-11     Admin
6          1 Monika      Arora       100000 14-02-20     HR
7          8 Geetika     Chauhan      90000 14-04-11     Admin
```

19. Write an SQL query to print details for Workers with the first name as “Vipul” and “Satish” from Worker table.

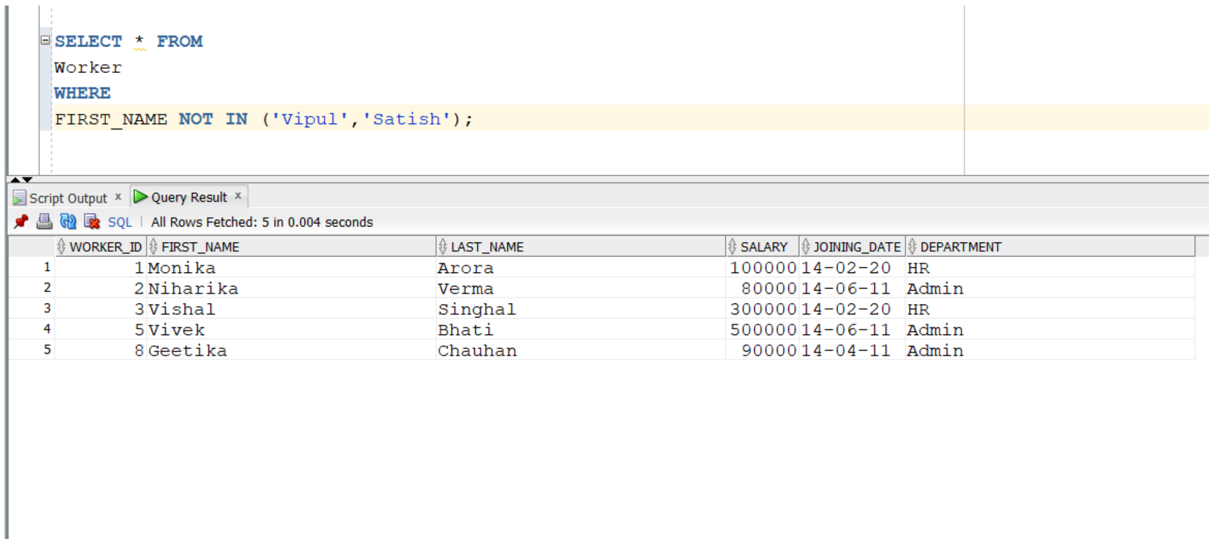


```
SELECT * FROM
Worker
WHERE
FIRST_NAME IN ('Vipul','Satish');
```

All Rows Fetched: 2 in 0.005 seconds

```
   WORKER_ID FIRST_NAME  LAST_NAME   SALARY JOINING_DATE DEPARTMENT
1          6 Vipul       Diwan       200000 14-06-11     Account
2          7 Satish      Kumar        75000 14-01-20     Account
```

20. Write an SQL query to print details of workers **excluding** first names, “Vipul” and “Satish” from Worker table.



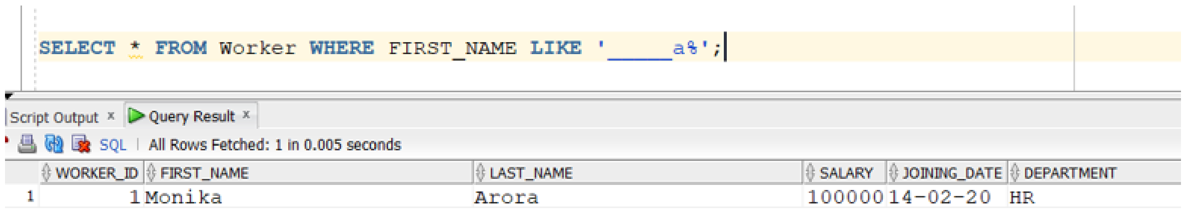
```
SELECT * FROM
Worker
WHERE
FIRST_NAME NOT IN ('Vipul','Satish');
```

Script Output × Query Result ×

SQL | All Rows Fetched: 5 in 0.004 seconds

	WORKER_ID	FIRST_NAME	LAST_NAME	SALARY	JOINING_DATE	DEPARTMENT
1	1	Monika	Arora	100000	14-02-20	HR
2	2	Niharika	Verma	80000	14-06-11	Admin
3	3	Vishal	Singhal	300000	14-02-20	HR
4	5	Vivek	Bhati	500000	14-06-11	Admin
5	8	Geetika	Chauhan	90000	14-04-11	Admin

21. Write an SQL query to print details of the Workers whose FIRST_NAME ends with ‘h’ and contains six alphabets



```
SELECT * FROM Worker WHERE FIRST_NAME LIKE '_____a%';
```

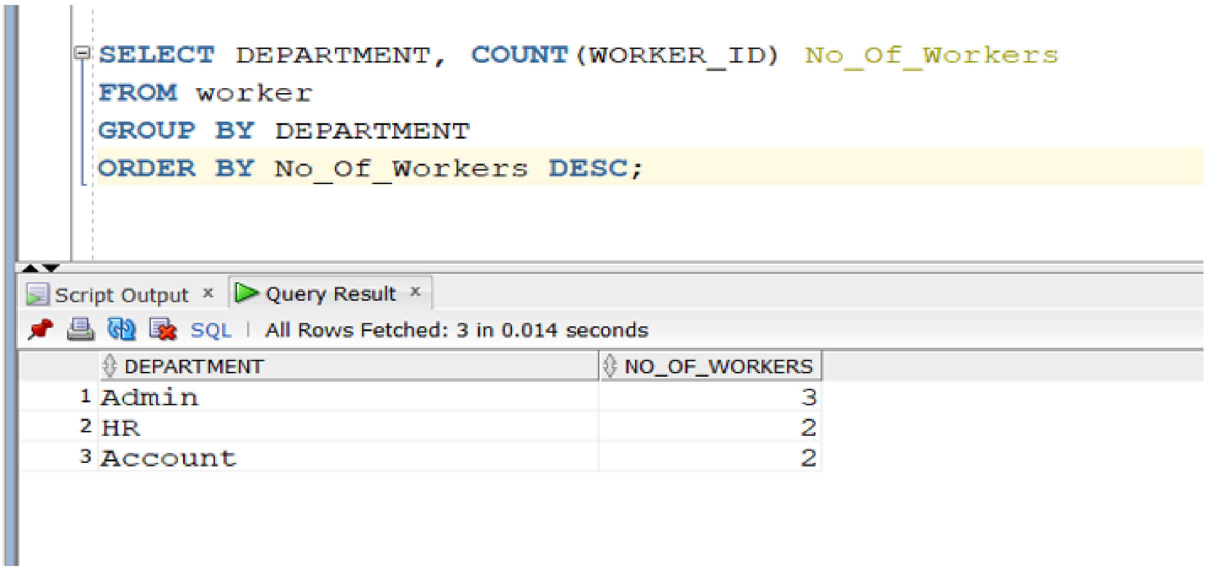
Script Output × Query Result ×

SQL | All Rows Fetched: 1 in 0.005 seconds

	WORKER_ID	FIRST_NAME	LAST_NAME	SALARY	JOINING_DATE	DEPARTMENT
1	1	Monika	Arora	100000	14-02-20	HR

22. Write an SQL query to fetch the no. of workers for each department in the descending order.

a. Here, No_of_workers is not the table column, we just count the worker ID and store its value in No_Of_worker which is temporary column.



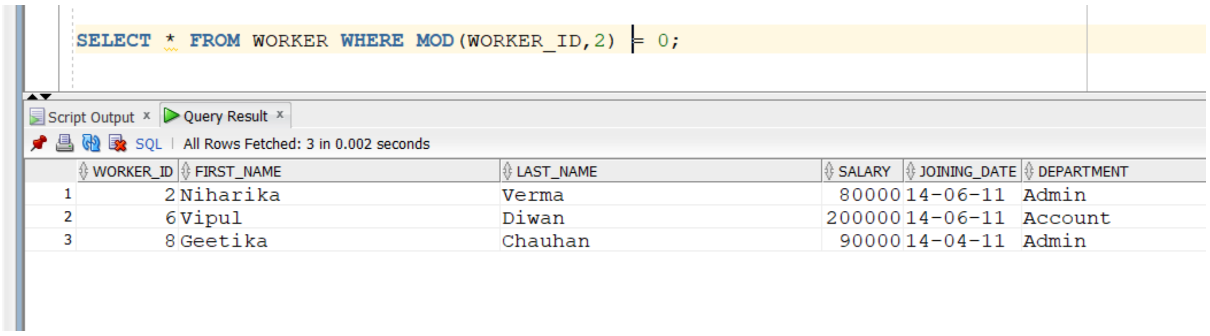
```
SELECT DEPARTMENT, COUNT(WORKER_ID) No_Of_Workers
FROM worker
GROUP BY DEPARTMENT
ORDER BY No_Of_Workers DESC;
```

Script Output × Query Result ×

SQL | All Rows Fetched: 3 in 0.014 seconds

	DEPARTMENT	NO_OF_WORKERS
1	Admin	3
2	HR	2
3	Account	2

23. Write an SQL query to show only even rows from a table.



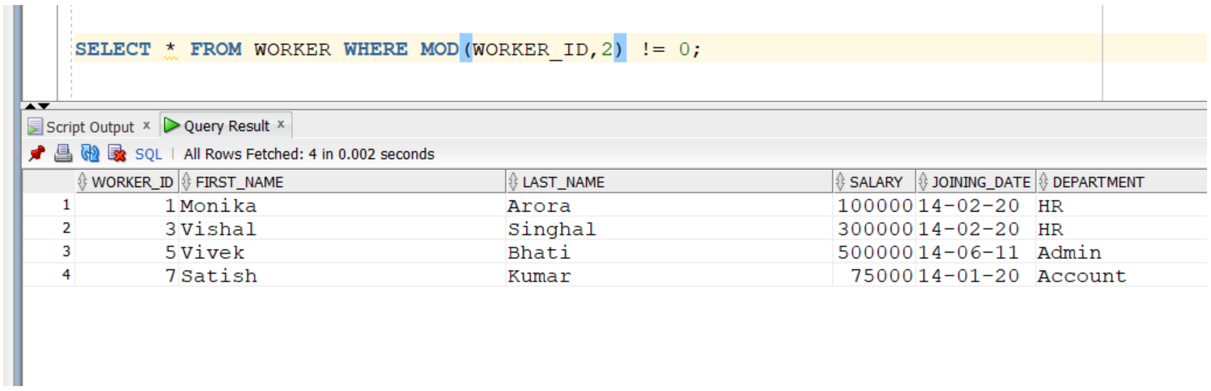
```sql
SELECT * FROM WORKER WHERE MOD(WORKER_ID,2) = 0;
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 3 in 0.002 seconds

```
   WORKER_ID FIRST_NAME   LAST_NAME   SALARY  JOINING_DATE DEPARTMENT
1          2 Niharika     Verma        80000  14-06-11     Admin
2          6 Vipul        Diwan       200000  14-06-11     Account
3          8 Geetika      Chauhan      90000  14-04-11     Admin
```

24. Write an SQL query to show only even rows from a table.



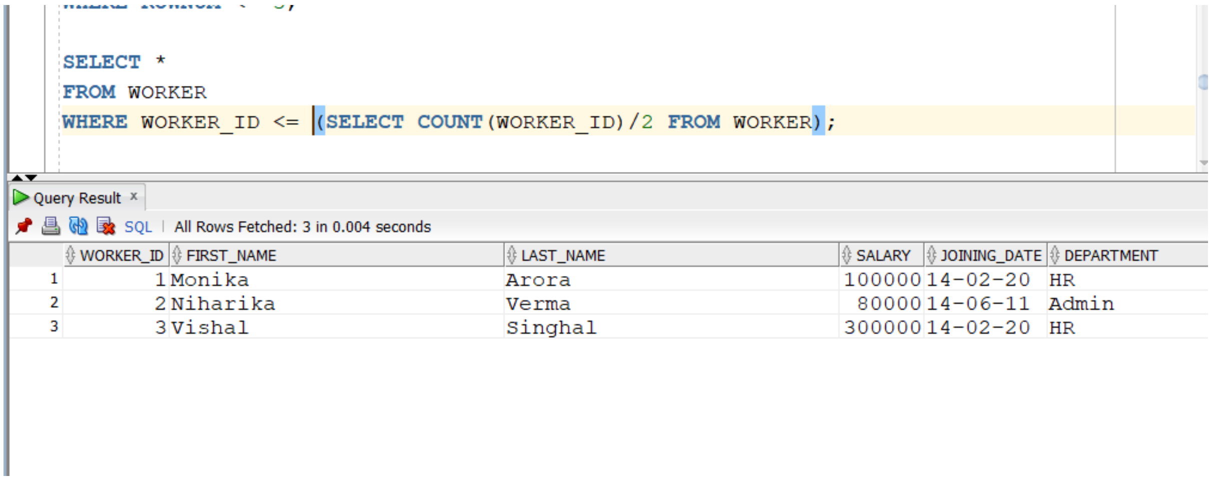
```sql
SELECT * FROM WORKER WHERE MOD(WORKER_ID,2) != 0;
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 4 in 0.002 seconds

```
   WORKER_ID FIRST_NAME   LAST_NAME   SALARY  JOINING_DATE DEPARTMENT
1          1 Monika       Arora       100000  14-02-20     HR
2          3 Vishal       Singhal     300000  14-02-20     HR
3          5 Vivek        Bhati       500000  14-06-11     Admin
4          7 Satish       Kumar        75000  14-01-20     Account
```

25. Write an SQL query to fetch the first 50% records from a table



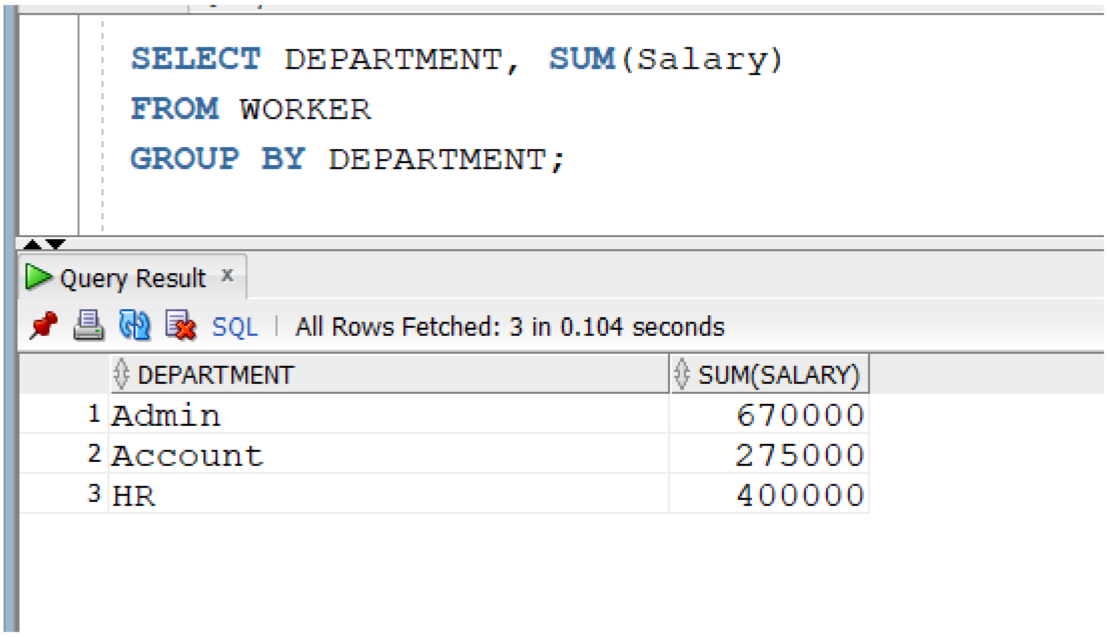
```sql
SELECT *
FROM WORKER
WHERE WORKER_ID <= (SELECT COUNT(WORKER_ID)/2 FROM WORKER);
```

Query Result ×

SQL | All Rows Fetched: 3 in 0.004 seconds

```
   WORKER_ID FIRST_NAME   LAST_NAME   SALARY  JOINING_DATE DEPARTMENT
1          1 Monika       Arora       100000  14-02-20     HR
2          2 Niharika     Verma        80000  14-06-11     Admin
3          3 Vishal       Singhal     300000  14-02-20     HR
```

26. Write an SQL query to fetch departments along with the total salaries paid for each of them.



```
SELECT DEPARTMENT, SUM(Salary)
FROM WORKER
GROUP BY DEPARTMENT;
```

Query Result

SQL | All Rows Fetched: 3 in 0.104 seconds

	DEPARTMENT	SUM(SALARY)
1	Admin	670000
2	Account	275000
3	HR	400000