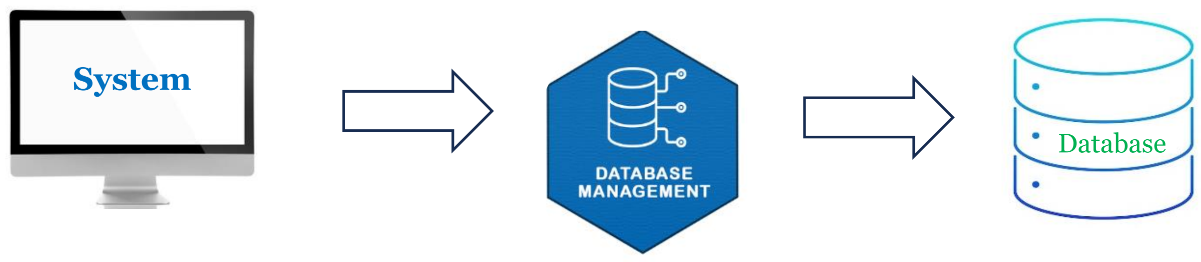
DATABASE

Database: It is a collection of data in a format that can be easily accessed.

To manage these databases, **Database Management System** (DBMS) are used.



DBMS: It is a software used to create, manage and organize data in a structured manner.

DBMS provide an environment to store and retrieve the data in convenient and efficient manner.

Example: A university database organizes the data about student, faculty, staff etc.

Key Features of DBMS:

- **Data Modeling:** Tools for defining the structure and relationships of data within a database.
- **Data Storage and Retrieval:** Manages storing and accessing data, enabling efficient searching and querying.
- **Concurrency Control:** Ensures multiple users can access the database simultaneously without conflicts.
- **Data Integrity and Security:** Enforces rules for data accuracy and restricts access to authorized users only.
- **Backup and Recovery:** Provides mechanisms to back up data and restore it after system failures.

DBMS can be classified into two types:

1. Relational Database Management System (RDBMS)
2. Non-Relational Database Management System (NoSQL or Non-SQL)

RDBMS: Data is organized in the form of tables and each table has a set of rows and columns. The data are related to each other through primary and foreign keys.

Example: MySQL, Oracle, PostgreSQL

ID	Name	Class
101	Charlie	9
102	Bob	11th
103	Alice	10

Advantages of RDBMS:

1. Every piece of information is stored in the form of tables.
2. Has primary key for unique identification of rows.
3. Has foreign key to ensure data integrity.
4. Uses indexes for faster data retrieval.
5. Give access privileges to ensure data security.

NoSQL: Data is organized in the form of key-value pairs, documents, graphs, or column-based. These are designed to handle large-scale, high-performance scenarios.

Example: MongoDB, Redis

```
{
      "RollNo": 1,
      "Class": 5th,
    "Name": "Bob"
  }
```

SQL INTRODUCTION

SQL stands for **Structured Query Language.** It is used for accessing and manipulating databases.

What can SQL do?

With the help of sql we can execute query, retrieve data, create database and tables, insert records, update records, delete record, create views and set permission on tales, procedure and views.

SYNTAX:

```
Select * from table_name;
```

NOTE: SQL keywords are not case sensitive: Select is same as SELECT.

SQL DATA TYPES:

Numeric	bit, tinyint, smallint, int, bigint, decimal, numeric, float, real.
Character/String	Char, varchar, text.
Date/Time	date, time, datetime, timestamp, year.
Miscellaneous	json, Xml.

1. Boolean: it can hold one of three possible values: true, false or null. We **Boolean** or **bool** keyword to declare column with the Boolean data type.
   - 1, yes, y, t, true values are converted to true
   - 0, no, false, f values are converted to false.
2. Character: MySQL provides three character data types: char(n), Varchar(n), Text.
   - **CHAR(n)**: It is the fixed-length character with space padded. If you insert a string that is shorter than the length of the column, MySQL pads spaces. (0-255)

     Note: If you insert a string that is longer than the length of the column, MySQL will issue an error.

- **VARCHAR(n)**: It is the variable-length character string. The VARCHAR(n) allows you to store up to n characters. MySQL does not pad spaces when the stored string is shorter than the length of the column. (0-255)
- **TEXT**: it is the variable-length character string. Theoretically, text data is a character string with unlimited length.

3. Numeric:

   By default all the numeric datatypes can have negative as well as positive values. This restrict the range so if we know there is only +ve values which is stored we use UNSIGNED attribute (0-255). **for eg:** salary can never be in negative or age

   create table employee (emp_id int, emp_name varchar(20), emp_salary unsigned);

   a) **Integer:**

      i. **Small integer** (**SMALLINT**) has a range -32, 768 to 32, 767 and has a size of 2-byte.

      ii. **Integer** (**INT**) has a range -2, 147, 483, 648 to 2, 147, 483, 647 and has a size of 4-byte.

      iii. **Serial** (**SERIAL**) works similar to the integers except these are automatically generated in the columns by PostgreSQL.

   b) **Floating-point number:**

      i. **float(n)** is used for floating-point numbers with n precision and can have a maximum of 8-bytes.

      ii. Double: Used for storing decimal numbers. (8-byte)

      iii. Decimal (p, s): Used for exact numeric representation. **p** is the precision and **s** is the scale.

4. Date/Time: This data type is used to store date-time data.

   i. **DATE** is used to store the dates only (YYY-MM-DD).

   ii. **TIME** is used to stores the time of day values (hh:mm:ss).

   iii. **TIMESTAMP/DATETIME:** is used to stores both date and time values (yyy-mm-dd hh:mm:ss)

CONSTRAINTS;

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

The following constraints are commonly used in SQL:

Constraint	Explanation	Example
NOT NULL	Ensure that a column cannot have a null value.	Create table student (id int not null, name varchar (20) not null);
UNIQUE	Ensure that all values in a column are different.	Create table student (id int not null unique, name varchar (20) not null);
PRIMARY KEY	A combination of NOT NULL and UNIQUE. Uniquely identifies each row in a table. A table can have only one field as primary key	Create table student (id int not null, name varchar (20) not null, primary key (id);
FOREIGN KEY	It is a field in a table which uniquely identifies each row of a another table. That is, this field points to primary key of another table.	Create table orders(o_id int not null, oreder_no int not null, c_id int, primary key (o_id), Foreign key(c_id) references customer(c_id);
CHECK	Ensure that the values in a column satisfies a specific condition.	Create table student (id int not null, name varchar (20) not null, age int not null check (age >=18);
DEFAULT	Sets a default value for a column if no value is specified.	Create table student (id int not null, name varchar (20) not null, age int default 18);

CREATE INDEX	Used to create and retrieve data from the database very quickly.	Create index index_name on table_name (column1, column2, ...); Or combination of columns Create index idx_pname on persons (lastname, firstname);

SQL COMMANDS GROUPS:

1. DQL: It stands for **Data Query Language.** It is used to retrieve data from the database.

SELECT	SELECT command shows the records of the specified table. It also shows the particular record of a particular column by using the WHERE clause.	Select column_1, column_2, ….., column_n from name_of_table; Retrieve the data from all the columns of the table Select * from table_name;

2. DDL: It stands for **Data Definition Language**. It is used to define the structure and schema of the database

   Following are the 5 DDL commands in sql:

CREATE	used to create databases, tables, triggers and other database objects.	Create database database_name;
DROP	used to delete/remove the database objects from the SQL database.	Drop database database_name; Drop table table_name;
ALTER	which changes or modifies the existing structure of the database, and it also changes the schema of database objects. We can also add and drop constraints of the table using the ALTER command.	Alter table name_of_table Add column_name column_definition;

TRUNCATE	which deletes or removes all the records from the table.	Truncate table table_name;
RENAME	is used to change the name of the database table.	Rename table old_table_name to new_table_name;

3. DML: It stands for **Data Manipulation Language**. It is used to manipulate data stored in the database.

   Following are the four main DML commands in SQL:

INSERT	allows users to insert data in database tables.	Insert into table_name (column_1 , column _2 , ... Column_n) values (value_1, value_ 2, ... Value_n) ;
UPDATE	allows users to update or modify the existing data in database tables.	Update table_name set [column_1= value_1 , ….., column_n = value_n] where condition ;
DELETE	allows SQL users to remove single or multiple existing records from the database tables.	Delete from table_name Where condition;

Note: Delete command of Data Manipulation Language does not delete the stored data permanently from the database. We use the WHERE clause with the DELETE command to select specific rows from the table.

4. DCL: It stands for **Data Control Language.** It deals with the control and security of data within the database.

GRANT	Assigns new privileges to a user account, allowing access to specific database objects, actions, or functions.	Grant privilege_type [(column_list)] on [object_type] object_name to user [with grant option];

REVOKE	Removes previously granted privileges from a user account, taking away their access to certain database objects or actions.	Revoke [grant option for] privilege_type[(column_list)] on [object_type] object_name from user [cascade];

5. TCL: It stands for **Transaction Control Language**. It is used to manage transactions within a database.

   Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group are successfully completed. If any of the tasks fail, the transaction fails.

   A transaction has only two results: **success or failure.**

BEGIN TRANSACTION	Starts a new transaction	Begin transaction [transaction_name];
COMMIT	Saves all changes made during the transaction	Commit;
ROOLBACK	Undoes all changes made during the transaction	Rollback;
SAVEPOINT	Creates a savepoint within the current transaction	Savepoint savepoint_name;

IMPORTANT Commands

Commands	Working	Syntax
SELECT	Exact data from database	Select * from table_name;
UPDATE	Update data in a database	Update table_name set col_1 = value1,… where condition;
DELETE	Delete data from a database	Delete from table_name where condition;

INSERT INTO	Insert new record into a database	Insert into table_name (col_1, col_2, col_3,..) values(value1, value2, value3,…);
CREATE DATABASE	Create a new database	Create database database_name;
ALTER DATABASE	Modifies a database.	Alter table employees add column phone varchar(20);
CREATE TABLE	Create a new table	Create table table_name;
ALTER TABLE	Modifies a table	Alter table table_name add col_name datatype;
DROP TABLE	Delete a table	Drop table table_name;
CREATE INDEX	Creates an index	Create index index_name on table_name (column1, column2, ...);
DROP INDEX	Deletes an index	Drop index table_name.index_name;

WHERE: it is used to filter records.

SELECT column1, column2, ... FROM table_name WHERE condition;

NOTE: The WHERE clause is not only used in SELECT statements, it is also used in UPDATE, DELETE, etc.

Operators in the WHERE Clause

Operator	**Description**	**Example**
=	Equal	Select * from customer where city = "Mumbai"
>	Greater Than	Select * from customer where price > 50;
<	Less Than	Select * from customer where price < 50;
>=	Greater Than or Equal	Select * from customer where price >= 50;
<=	Less Than or Equal	Select * from customer where price <= 50;
< >	Not Equal	Select * from customer where price < > 100;

LIKE	Search for a pattern	Select * from customer where city Like ‘s%’;
IN	Filters results based on a list of values in the WHERE clause.	Select * from customer where city IN (‘paris’, ‘London’);
BETWEEN	Filters results within a specified range in the WHERE clause.	Select * from customer where price between 20 and 30;
OR	Displays a record if **any** of the conditions are TRUE.	SELECT column1, column2, ... FROM table_name WHERE condition1 OR condition2 OR condition3 ...;
AND	The operator displays a record if **all** the conditions are TRUE.	SELECT column1, column2, ... FROM table_name WHERE condition1 AND condition2 AND condition3 ...;
NOT	The operator displays a record if the condition(s) is NOT TRUE.	SELECT column1, column2, ... FROM table_name WHERE NOT condition;

DISTINCT: It is used to return only distinct (different) values.

```
SELECT DISTINCT column1, column2, .. FROM table_name;
```

LIKE: It 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

Different Pattern Examples:

Finds any values that start with "a"	WHERE CustomerName LIKE 'a%'
Finds any values that end with "a"	WHERE CustomerName LIKE '%a'
Finds any values that have "or" in any position	WHERE CustomerName LIKE '%or%'
Finds any values that have "r" in the second position	WHERE CustomerName LIKE '_r%'
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 CustomerName LIKE 'a__%'
Finds any values that start with "a" and ends with "o".	WHERE ContactName LIKE 'a%o'

IS NULL	Checks for NULL values in the WHERE clause.	Select * from customers where email is null;
AS	Renames columns or expressions in query results.	Select first_name as "First Name", last_name as "Last Name" from employees;

ORDER BY: It is used to sort the result in ascending or descending order.

```
select column1, column2, .. from table_name order by column1, column2, ... asc | desc;
```

The ORDER BY keyword sorts the records in **ascending** order by default. To sort the records in **descending** order, use the DESC keyword.

Sorting by Multiple Columns:	List multiple columns sequentially in the ORDER BY clause. Sorting prioritizes the first column; subsequent columns sort ties.	SELECT first_name, last_name FROM employees ORDER BY last_name, first_name;
Sorting by Expressions:	Sort based on calculated expressions, not just columns.	SELECT product_name, price, price * 1.1 AS discounted_price FROM products ORDER BY discounted_price;
Sorting NULL Values:	By default, NULL values are smallest in ascending order and largest in descending order. Control NULL sorting with NULLS FIRST or NULLS LAST.	SELECT column_name FROM table_name ORDER BY column_name NULLS LAST;
Sorting by Position:	Sort using column positions in the ORDER BY clause instead of names.	SELECT product_name, price FROM products ORDER BY 2 DESC, 1 ASC;

GROUP BY: This is used to group rows that have the same values into together. It helps to organize data into groups so that we can do calculations, like finding totals or averages, for each group.

```
SELECT column1, aggregate_function(column2) FROM table_name GROUP BY column1;
```

Aggregation Functions:	Commonly used with GROUP BY to calculate values for each group. **Examples of aggregation functions:** COUNT, SUM, AVG, MAX, MIN.	SELECT department, AVG(salary) FROM employees GROUP BY department;
Grouping by Multiple Columns:	You can group by multiple columns by listing them in the GROUP BY clause. This creates hierarchical grouping based on the specified columns.	SELECT department, gender, AVG(salary) FROM employees GROUP BY department, gender;

HAVING:

The HAVING clause filters query results based on aggregate functions and groupings, which cannot be done with the WHERE clause.

```
SELECT col_1, function_name(col_2) FROM tablename WHERE condition GROUP BY
column1, column2  HAVING Condition  ORDER BY column1, column2;
```

Some important points:

- Having clause is used to filter data according to the conditions provided.
- Having a clause is generally used in reports of large data.
- Having clause is only used with the SELECT clause.
- The expression in the syntax can only have constants.
- In the query, ORDER BY is to be placed after the HAVING clause, if any.
- HAVING Clause is implemented in column operation.
- Having clause is generally used after GROUP BY.

AGGREGATE FUNCTIONS:

An aggregate function performs a calculation on a set of values and returns a single value.

Aggregate functions are often used with the GROUP BY clause in the SELECT statement.

Functionality with GROUP BY:

- The GROUP BY clause splits the result set into groups based on specified columns.
- The aggregate function returns a single value for each group.

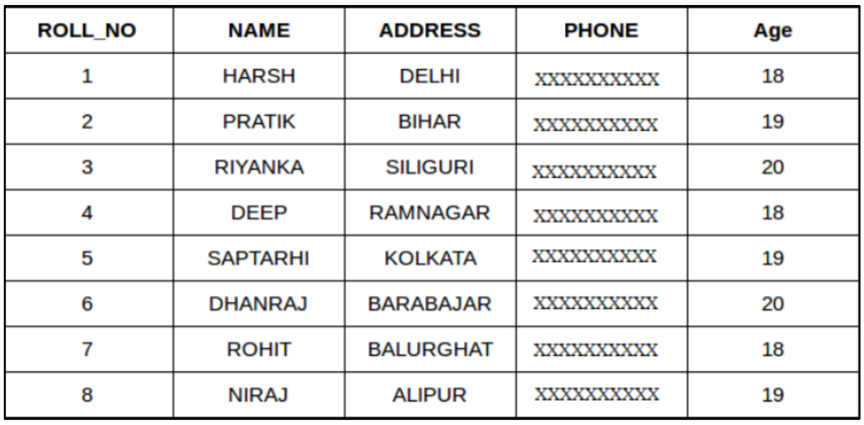
The most commonly used SQL aggregate functions are:

MIN ()	Returns the smallest value within the selected column	Select MIN (col_name) from table_name where condition;
MAX ()	Returns the largest value within the selected column	Select MAX (col_name) from table_name where condition;
COUNT ()	Returns the number of rows in a set	Select COUNT (col_name) from table_name where condition;
SUM ()	Returns the total sum of a numerical column	Select SUM (col_name) from table_name where condition;
AVG ()	Returns the average value of a numerical column	Select AVG (col_name) from table_name where condition;

JOINS:

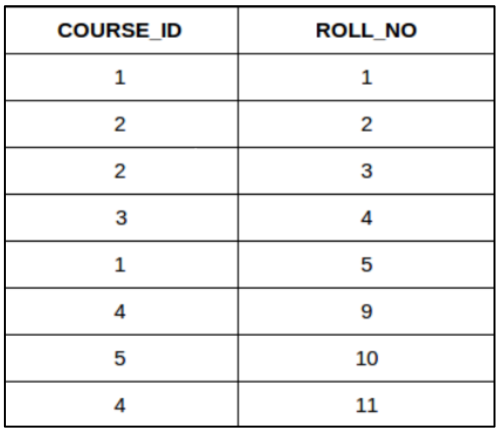
- ➢ SQL JOIN combines data from two or more tables based on a common column.
- ➢ The JOIN clause lets you access and query data from multiple tables by creating a logical relationship between them.
- ➢ JOIN uses common key values (like a shared ID) across tables to connect rows from those tables.
- ➢ You can use JOIN to combine data from more than two tables.
- ➢ JOIN is often used with the WHERE clause to filter the data that’s retrieved, making it more specific.

Example:



ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

Student



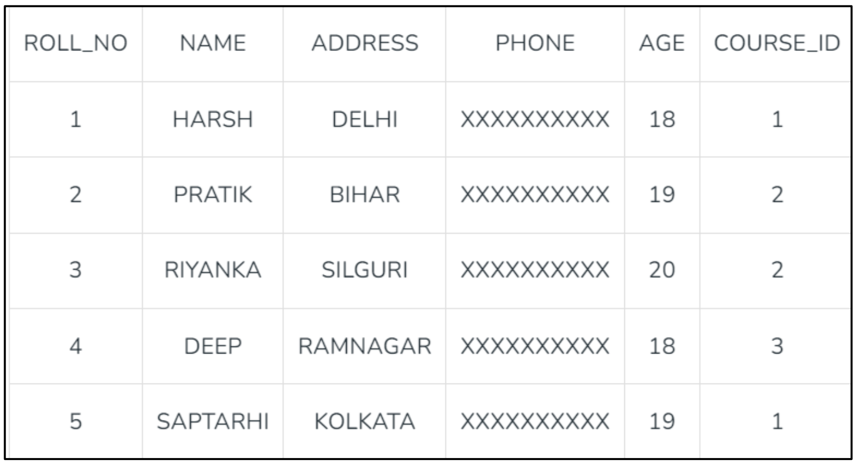
COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

StudentCourse

Both these tables are connected by one common key (column) i.e ROLL_NO.

We can perform a JOIN operation using the given SQL query:

SELECT s.roll_no, s.name, s.address, s.phone, s.age, sc.course_id FROM Student s JOIN StudentCourse sc ON s.roll_no = sc.roll_no;



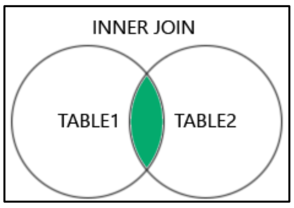
ROLL_NO	NAME	ADDRESS	PHONE	AGE	COURSE_ID
1	HARSH	DELHI	XXXXXXXXXX	18	1
2	PRATIK	BIHAR	XXXXXXXXXX	19	2
3	RIYANKA	SILGURI	XXXXXXXXXX	20	2
4	DEEP	RAMNAGAR	XXXXXXXXXX	18	3
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19	1

OUTPUT

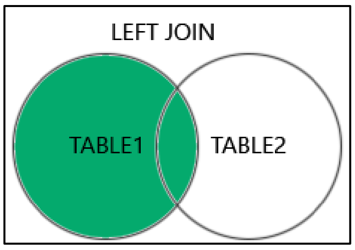
Different types of SQL JOINS:

1. INNER JOIN: Returns records that have matching values in both tables.

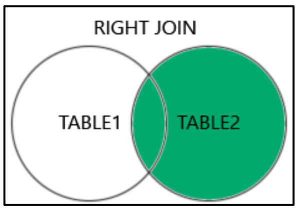
Note: We can also write Join instead of INNER JOIN. JOIN is same as INNER JOIN.



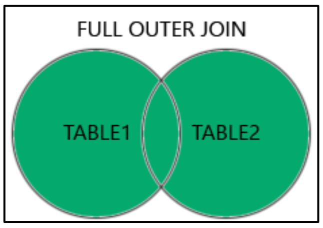
2. LEFT JOIN (OUTER): Returns all records from the left table, and the matched records from the right table.



3. RIGH JOIN (OUTER): Returns all records from the right table, and the matched records from the left table



4. FULL JOIN (OUTER): Returns all records when there is a match in either left or right table



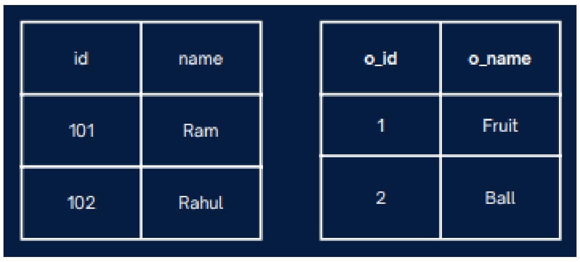
SYNTAX:

```
SELECT table1.column1,table1.column2,table2.column1,.... FROM table1 JOIN_NAME table2
ON table1.matching_column = table2.matching_column;
```

5. CROSS JOIN: It combines each row of the first table with every row of the second table.

It results in a new table where the number of rows is equal to the product of the number of rows in each table. **(m*n)**

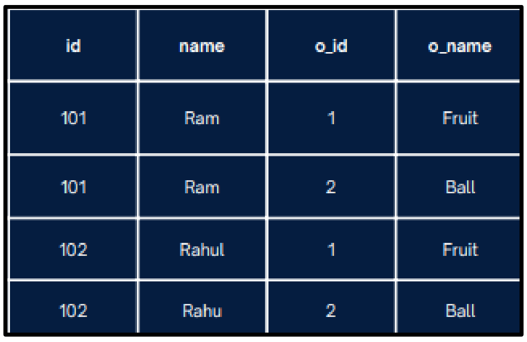
Customer **Order**



id	name
101	Ram
102	Rahul

o_id	o_name
1	Fruit
2	Ball

Example: **SELECT * FROM CUSTOMER CROSS JOIN ORDER;**



id	name	o_id	o_name
101	Ram	1	Fruit
101	Ram	2	Ball
102	Rahul	1	Fruit
102	Rahu	2	Ball

6. SELF JOIN: A self join in SQL is a type of join where a table is joined with itself. It is a type of inner join.

   Syntax:

   SELECT columns FROM table1 as t1 JOIN table2 as t2 on t1.column_name = t2.column_name;

❖ **UNION OPERATOR:** 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 columns must also have similar data types
- The columns in every SELECT statement must also be in the same order.

  SELECT columns_name FROM table1 UNION select columns_name FROM table2;

❖ **UNION ALL:** UNION operator in SQL is used to combine the results of two or more SELECT queries into a single result set and gives all rows by not removing duplicate rows.

```
SELECT columns_name FROM table1 UNION ALL select columns_name FROM table2;
```

Note: The column names in the result-set are usually equal to the column names in the first SELECT statement.

SUBQUERIES/ NESTED QUERIES:

SQL subquery is a query nested within another SQL statement. Whenever we want to retrieve data based on the result of another query we use nested queries.

How can we use Subqueries?

Subqueries can be used in multiple ways:

1. Subqueries can be used with clauses such as SELECT, INSERT, UPDATE, or DELETE to perform complex data retrieval.

   Syntax: `SELECT columns, (subquery) FROM tableName;`

2. Subqueries can be used with WHERE clause to filter data based on conditions.

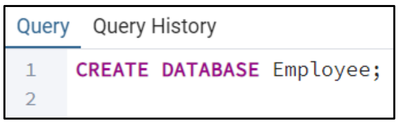
   Syntax: `SELECT * FROM tableName WHERE column name operator (subquery);`

3. Subqueries can also be used in the FROM clause.

   Syntax: `SELECT * FROM subquery AS altName ;`

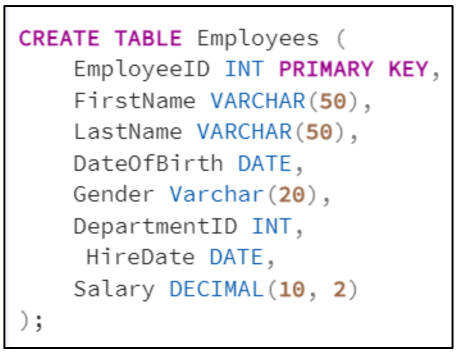
PRACTICE QUERY:

1. Write the correct SQL statement to create a new database called **Employee.**



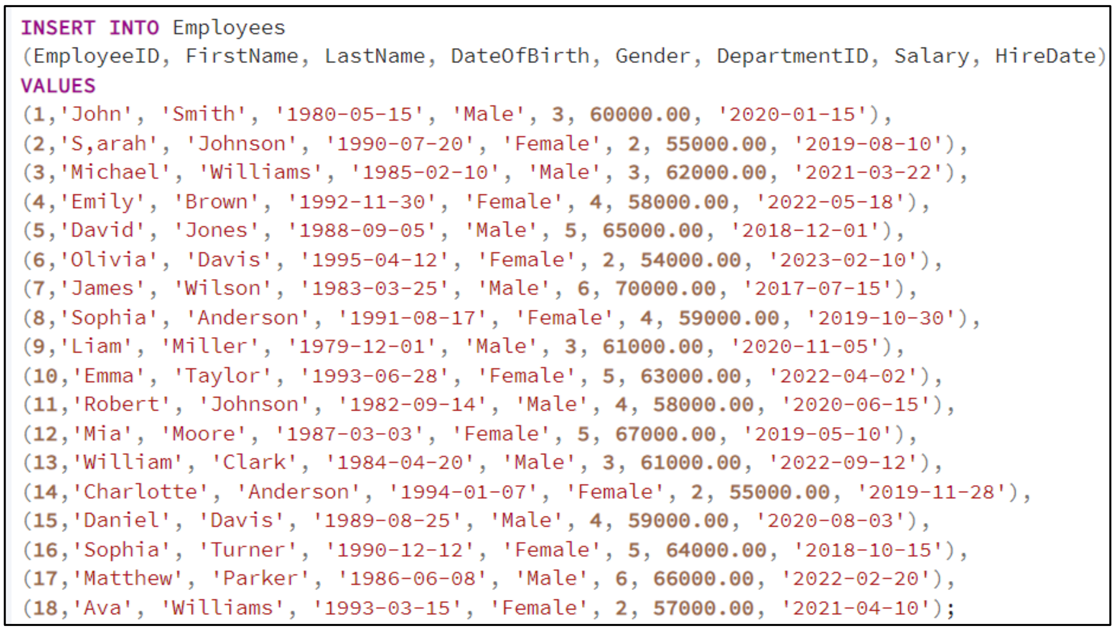
```sql
CREATE DATABASE Employee;
```

2. Write the correct SQL statement to create a new table called **Employees.**



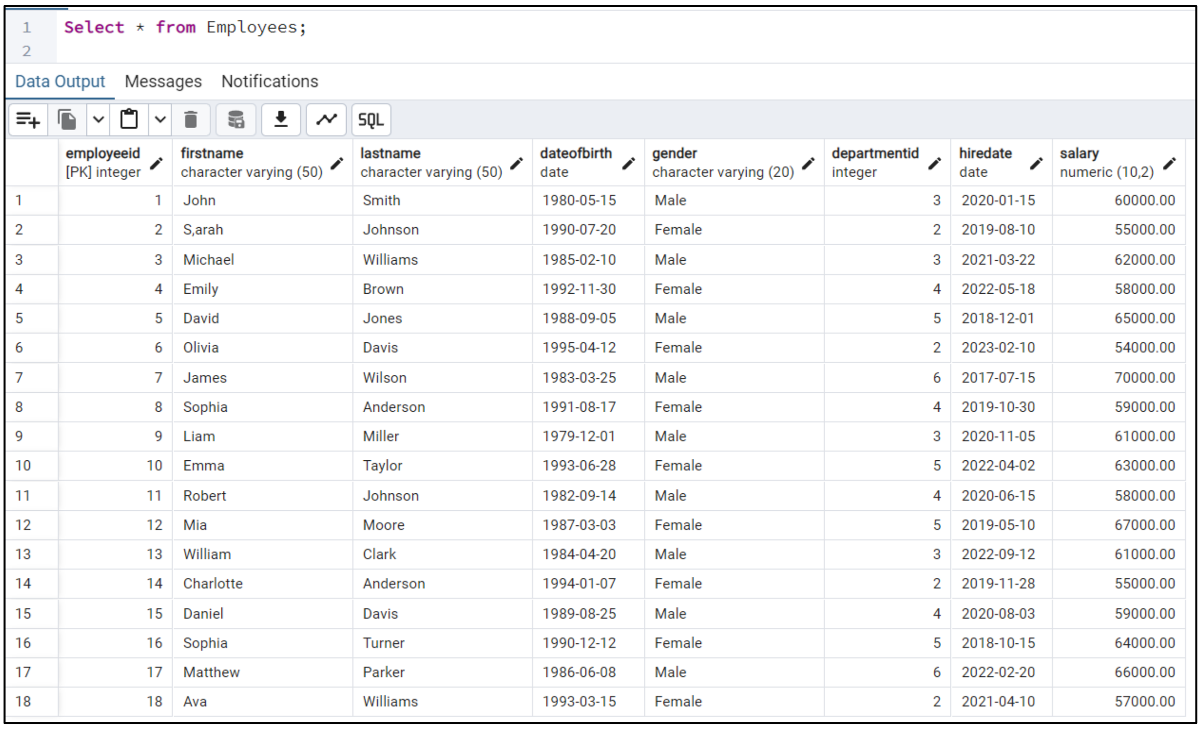
```sql
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    DateOfBirth DATE,
    Gender Varchar(20),
    DepartmentID INT,
     HireDate DATE,
    Salary DECIMAL(10, 2)
);
```

3. Insert a new record in the **Employees** table.



```sql
INSERT INTO Employees
(EmployeeID, FirstName, LastName, DateOfBirth, Gender, DepartmentID, Salary, HireDate)
VALUES
(1,'John', 'Smith', '1980-05-15', 'Male', 3, 60000.00, '2020-01-15'),
(2,'S,arah', 'Johnson', '1990-07-20', 'Female', 2, 55000.00, '2019-08-10'),
(3,'Michael', 'Williams', '1985-02-10', 'Male', 3, 62000.00, '2021-03-22'),
(4,'Emily', 'Brown', '1992-11-30', 'Female', 4, 58000.00, '2022-05-18'),
(5,'David', 'Jones', '1988-09-05', 'Male', 5, 65000.00, '2018-12-01'),
(6,'Olivia', 'Davis', '1995-04-12', 'Female', 2, 54000.00, '2023-02-10'),
(7,'James', 'Wilson', '1983-03-25', 'Male', 6, 70000.00, '2017-07-15'),
(8,'Sophia', 'Anderson', '1991-08-17', 'Female', 4, 59000.00, '2019-10-30'),
(9,'Liam', 'Miller', '1979-12-01', 'Male', 3, 61000.00, '2020-11-05'),
(10,'Emma', 'Taylor', '1993-06-28', 'Female', 5, 63000.00, '2022-04-02'),
(11,'Robert', 'Johnson', '1982-09-14', 'Male', 4, 58000.00, '2020-06-15'),
(12,'Mia', 'Moore', '1987-03-03', 'Female', 5, 67000.00, '2019-05-10'),
(13,'William', 'Clark', '1984-04-20', 'Male', 3, 61000.00, '2022-09-12'),
(14,'Charlotte', 'Anderson', '1994-01-07', 'Female', 2, 55000.00, '2019-11-28'),
(15,'Daniel', 'Davis', '1989-08-25', 'Male', 4, 59000.00, '2020-08-03'),
(16,'Sophia', 'Turner', '1990-12-12', 'Female', 5, 64000.00, '2018-10-15'),
(17,'Matthew', 'Parker', '1986-06-08', 'Male', 6, 66000.00, '2022-02-20'),
(18,'Ava', 'Williams', '1993-03-15', 'Female', 2, 57000.00, '2021-04-10');
```

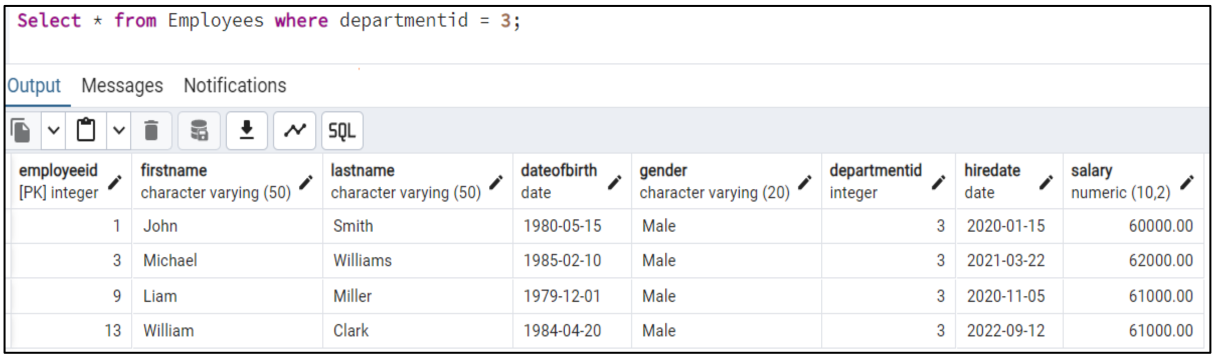
4. Return data from the **Employees** table.



```
Select * from Employees;
```

	employeeid [PK] integer	firstname character varying (50)	lastname character varying (50)	dateofbirth date	gender character varying (20)	departmentid integer	hiredate date	salary numeric (10,2)
1	1	John	Smith	1980-05-15	Male	3	2020-01-15	60000.00
2	2	S,arah	Johnson	1990-07-20	Female	2	2019-08-10	55000.00
3	3	Michael	Williams	1985-02-10	Male	3	2021-03-22	62000.00
4	4	Emily	Brown	1992-11-30	Female	4	2022-05-18	58000.00
5	5	David	Jones	1988-09-05	Male	5	2018-12-01	65000.00
6	6	Olivia	Davis	1995-04-12	Female	2	2023-02-10	54000.00
7	7	James	Wilson	1983-03-25	Male	6	2017-07-15	70000.00
8	8	Sophia	Anderson	1991-08-17	Female	4	2019-10-30	59000.00
9	9	Liam	Miller	1979-12-01	Male	3	2020-11-05	61000.00
10	10	Emma	Taylor	1993-06-28	Female	5	2022-04-02	63000.00
11	11	Robert	Johnson	1982-09-14	Male	4	2020-06-15	58000.00
12	12	Mia	Moore	1987-03-03	Female	5	2019-05-10	67000.00
13	13	William	Clark	1984-04-20	Male	3	2022-09-12	61000.00
14	14	Charlotte	Anderson	1994-01-07	Female	2	2019-11-28	55000.00
15	15	Daniel	Davis	1989-08-25	Male	4	2020-08-03	59000.00
16	16	Sophia	Turner	1990-12-12	Female	5	2018-10-15	64000.00
17	17	Matthew	Parker	1986-06-08	Male	6	2022-02-20	66000.00
18	18	Ava	Williams	1993-03-15	Female	2	2021-04-10	57000.00

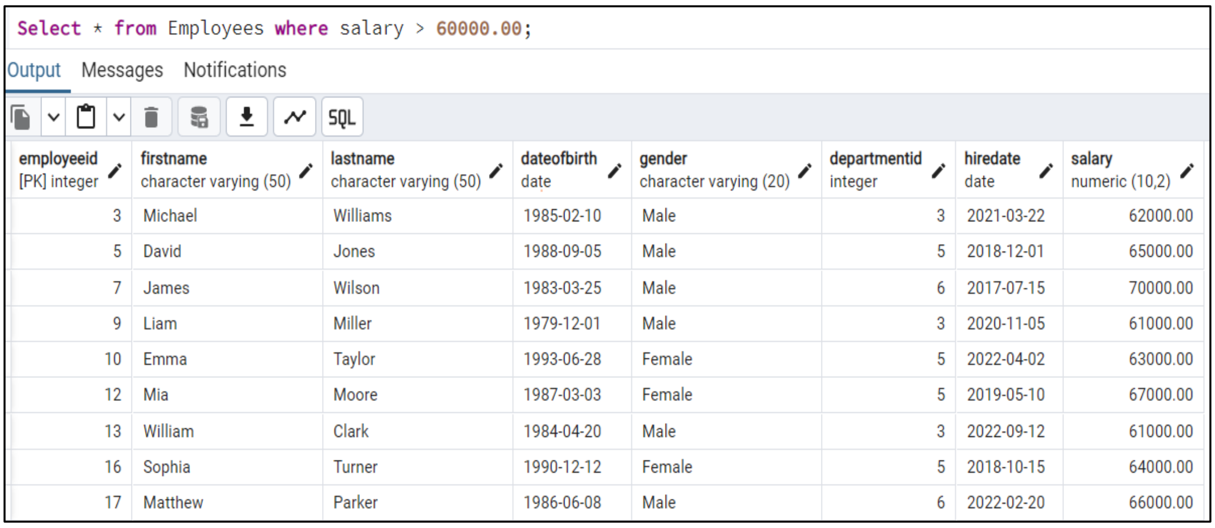
5. Select all records where the department_id = **3**.



```
Select * from Employees where departmentid = 3;
```

employeeid [PK] integer	firstname character varying (50)	lastname character varying (50)	dateofbirth date	gender character varying (20)	departmentid integer	hiredate date	salary numeric (10,2)
1	John	Smith	1980-05-15	Male	3	2020-01-15	60000.00
3	Michael	Williams	1985-02-10	Male	3	2021-03-22	62000.00
9	Liam	Miller	1979-12-01	Male	3	2020-11-05	61000.00
13	William	Clark	1984-04-20	Male	3	2022-09-12	61000.00

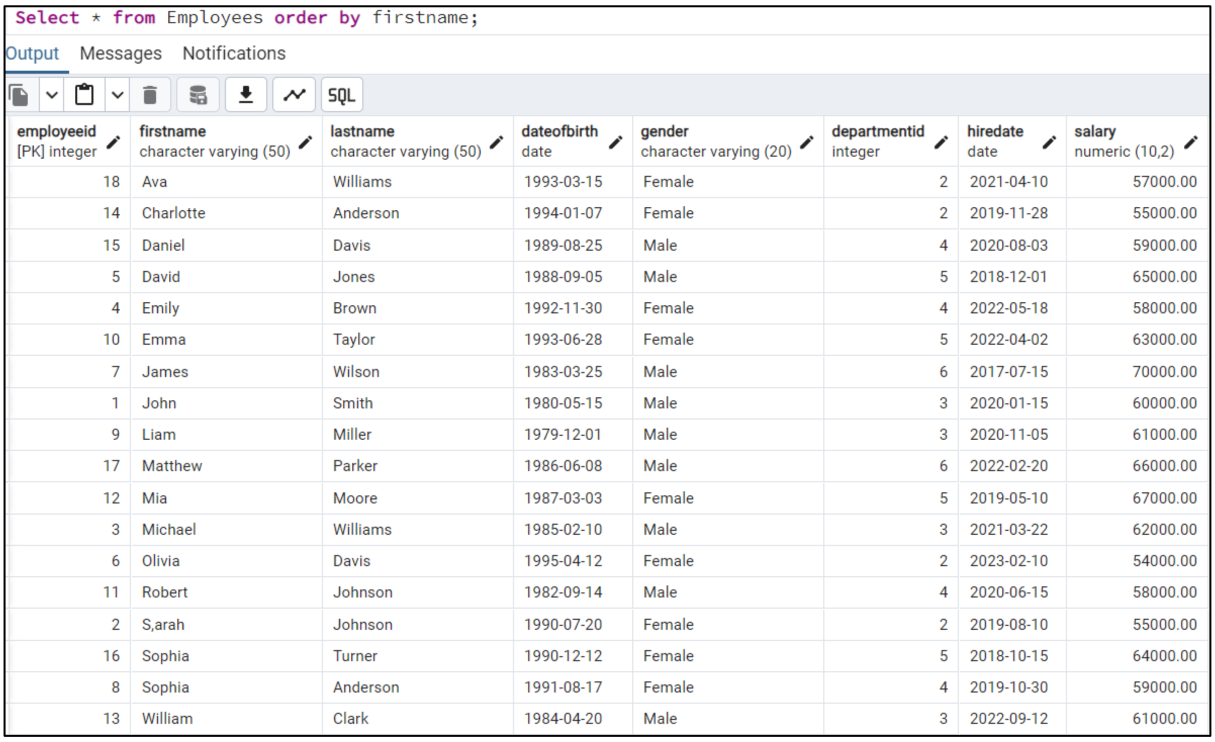
6. Select all records where the salary greater than **60,000**.



```
Select * from Employees where salary > 60000.00;
```

employeeid [PK] integer	firstname character varying (50)	lastname character varying (50)	dateofbirth date	gender character varying (20)	departmentid integer	hiredate date	salary numeric (10,2)
3	Michael	Williams	1985-02-10	Male	3	2021-03-22	62000.00
5	David	Jones	1988-09-05	Male	5	2018-12-01	65000.00
7	James	Wilson	1983-03-25	Male	6	2017-07-15	70000.00
9	Liam	Miller	1979-12-01	Male	3	2020-11-05	61000.00
10	Emma	Taylor	1993-06-28	Female	5	2022-04-02	63000.00
12	Mia	Moore	1987-03-03	Female	5	2019-05-10	67000.00
13	William	Clark	1984-04-20	Male	3	2022-09-12	61000.00
16	Sophia	Turner	1990-12-12	Female	5	2018-10-15	64000.00
17	Matthew	Parker	1986-06-08	Male	6	2022-02-20	66000.00

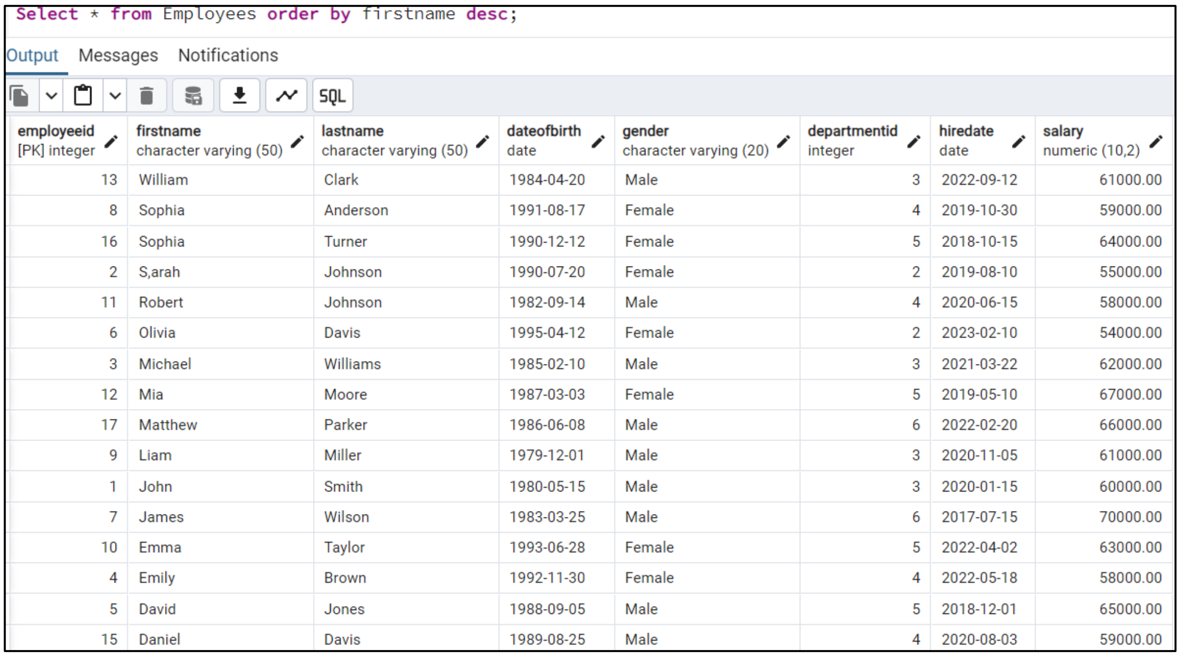
7. Select all records from the **Employees** table, sort the result alphabetically by the column firstname.



```
Select * from Employees order by firstname;
```

employeeid [PK] integer	firstname character varying (50)	lastname character varying (50)	dateofbirth date	gender character varying (20)	departmentid integer	hiredate date	salary numeric (10,2)
18	Ava	Williams	1993-03-15	Female	2	2021-04-10	57000.00
14	Charlotte	Anderson	1994-01-07	Female	2	2019-11-28	55000.00
15	Daniel	Davis	1989-08-25	Male	4	2020-08-03	59000.00
5	David	Jones	1988-09-05	Male	5	2018-12-01	65000.00
4	Emily	Brown	1992-11-30	Female	4	2022-05-18	58000.00
10	Emma	Taylor	1993-06-28	Female	5	2022-04-02	63000.00
7	James	Wilson	1983-03-25	Male	6	2017-07-15	70000.00
1	John	Smith	1980-05-15	Male	3	2020-01-15	60000.00
9	Liam	Miller	1979-12-01	Male	3	2020-11-05	61000.00
17	Matthew	Parker	1986-06-08	Male	6	2022-02-20	66000.00
12	Mia	Moore	1987-03-03	Female	5	2019-05-10	67000.00
3	Michael	Williams	1985-02-10	Male	3	2021-03-22	62000.00
6	Olivia	Davis	1995-04-12	Female	2	2023-02-10	54000.00
11	Robert	Johnson	1982-09-14	Male	4	2020-06-15	58000.00
2	S,arah	Johnson	1990-07-20	Female	2	2019-08-10	55000.00
16	Sophia	Turner	1990-12-12	Female	5	2018-10-15	64000.00
8	Sophia	Anderson	1991-08-17	Female	4	2019-10-30	59000.00
13	William	Clark	1984-04-20	Male	3	2022-09-12	61000.00

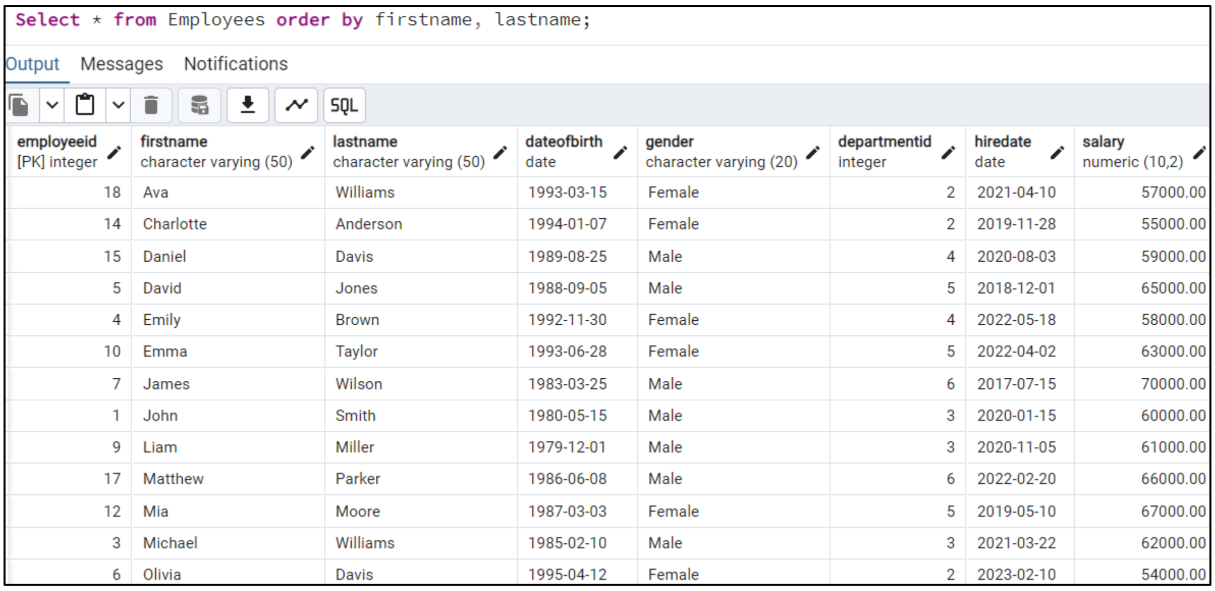
8. Select all records from the **Employees** table, sort the result *reversed* alphabetically by the column firstname.



```
Select * from Employees order by firstname desc;
```

employeeid [PK] integer	firstname character varying (50)	lastname character varying (50)	dateofbirth date	gender character varying (20)	departmentid integer	hiredate date	salary numeric (10,2)
13	William	Clark	1984-04-20	Male	3	2022-09-12	61000.00
8	Sophia	Anderson	1991-08-17	Female	4	2019-10-30	59000.00
16	Sophia	Turner	1990-12-12	Female	5	2018-10-15	64000.00
2	S,arah	Johnson	1990-07-20	Female	2	2019-08-10	55000.00
11	Robert	Johnson	1982-09-14	Male	4	2020-06-15	58000.00
6	Olivia	Davis	1995-04-12	Female	2	2023-02-10	54000.00
3	Michael	Williams	1985-02-10	Male	3	2021-03-22	62000.00
12	Mia	Moore	1987-03-03	Female	5	2019-05-10	67000.00
17	Matthew	Parker	1986-06-08	Male	6	2022-02-20	66000.00
9	Liam	Miller	1979-12-01	Male	3	2020-11-05	61000.00
1	John	Smith	1980-05-15	Male	3	2020-01-15	60000.00
7	James	Wilson	1983-03-25	Male	6	2017-07-15	70000.00
10	Emma	Taylor	1993-06-28	Female	5	2022-04-02	63000.00
4	Emily	Brown	1992-11-30	Female	4	2022-05-18	58000.00
5	David	Jones	1988-09-05	Male	5	2018-12-01	65000.00
15	Daniel	Davis	1989-08-25	Male	4	2020-08-03	59000.00

9. Select all records from the **Employees** table, sort the result alphabetically first by the column firstname. Then by lastname.



```
Select * from Employees order by firstname, lastname;
```

employeeid [PK] integer	firstname character varying (50)	lastname character varying (50)	dateofbirth date	gender character varying (20)	departmentid integer	hiredate date	salary numeric (10,2)
18	Ava	Williams	1993-03-15	Female	2	2021-04-10	57000.00
14	Charlotte	Anderson	1994-01-07	Female	2	2019-11-28	55000.00
15	Daniel	Davis	1989-08-25	Male	4	2020-08-03	59000.00
5	David	Jones	1988-09-05	Male	5	2018-12-01	65000.00
4	Emily	Brown	1992-11-30	Female	4	2022-05-18	58000.00
10	Emma	Taylor	1993-06-28	Female	5	2022-04-02	63000.00
7	James	Wilson	1983-03-25	Male	6	2017-07-15	70000.00
1	John	Smith	1980-05-15	Male	3	2020-01-15	60000.00
9	Liam	Miller	1979-12-01	Male	3	2020-11-05	61000.00
17	Matthew	Parker	1986-06-08	Male	6	2022-02-20	66000.00
12	Mia	Moore	1987-03-03	Female	5	2019-05-10	67000.00
3	Michael	Williams	1985-02-10	Male	3	2021-03-22	62000.00
6	Olivia	Davis	1995-04-12	Female	2	2023-02-10	54000.00