DATABASE

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

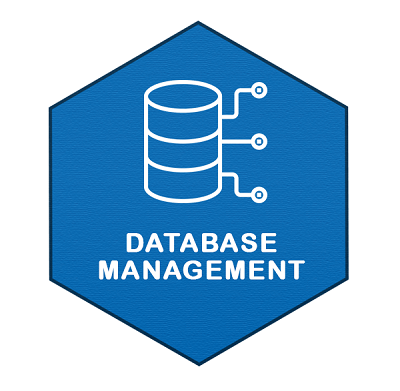
To manage these databases, **Database Management System** (DBMS) are used.



**System**



Database



**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. When we insert data into a Boolean column, PostgreSQL converts it to a Boolean value

* 1, yes, y, t, true values are converted to true
* 0, no, false, f values are converted to false.

1. Character: PostgreSQL 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, PostgreSQL pads spaces.

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

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

1. Numeric:
2. **Integer:**
3. **Small integer**(**SMALLINT**) has a range -32, 768 to 32, 767 and has a size of 2-byte.
4. **Integer**(**INT**) has a range -2, 147, 483, 648 to 2, 147, 483, 647 and has a size of 4-byte.
5. **Seria**l (**SERIAL**) works similar to the integers except these are automatically generated in the columns by PostgreSQL.
6. **Floating-point number:**
7. **float(n)** is used for floating-point numbers with n precision and can have a maximum of 8-bytes.
8. **float8** or **real**is used to represent 4-byte floating-point numbers.
9. A real number N(d,p) meaning with d number of digits and p number of decimal points after, are part of **numeric**or **numeric(d, p)**. These are generally very precise.
10. Date/Time: This data type is used to store date-time data. PostgreSQL has 5 temporal data type:
11. **DATE**is used to store the dates only.
12. **TIME**is used to stores the time of day values.
13. **TIMESTAMP** is used to stores both date and time values.
14. **TIMESTAMPTZ** is used to store a timezone-aware timestamp data type.

**IMP:** For More Data Type visit <https://www.postgresql.org/docs/current/datatype.html>

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. DDL: It stands for **Data Definition Language**.

The DDL Commands in Structured Query Language are used to create and modify the schema of the database and its objects.

**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; |

1. DML: It stands for **Data Manipulation Language**.

* The DML commands in Structured Query Language change the data present in the SQL database.
* We can easily access, store, modify, update and delete the existing records from the database using DML commands.

**Following are the four main DML commands in SQL:**

|  |  |  |
| --- | --- | --- |
| 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; |
| 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.

1. DCL: It stands for **Data Control Language.**

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

|  |  |  |
| --- | --- | --- |
| 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]; |

1. TCL: It stands for **Transaction Control Language**.

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 | To specify multiple possible values for a column | Select \* from customer where city IN (‘paris’, ‘London’); |
| BETWEEN | Between a certain range | Select \* from customer where price between 20 and 30; |

**AND, OR and NOT:**

* The WHERE clause can be combined with AND, OR, and NOT operators.
* The AND and OR operators are used to filter records based on more than one condition:
* The OR operator displays a record if **any** of the conditions are TRUE.

SELECT column1, column2, ... FROM table\_name WHERE condition1 OR condition2 OR condition3 ...;

* The AND operator displays a record if **all** the conditions are TRUE.

SELECT column1, column2, ... FROM table\_name WHERE condition1 AND condition2 AND condition3 ...;

* The NOT 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' |

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;

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: The GROUP BY clause groups rows from a table based on one or more columns.

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.

SELECT Department, sum(Salary) as Salary FROM employee GROUP BY department HAVING SUM(Salary) >= 50000;

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.

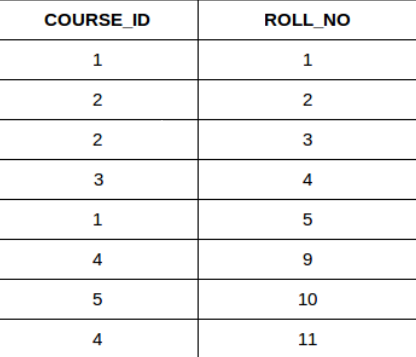
You can use JOIN to combine data from more than two tables.

JOIN is often used with the WHERE clause to filter the data that’s retrieved, making it more specific.

**Example:**



Student

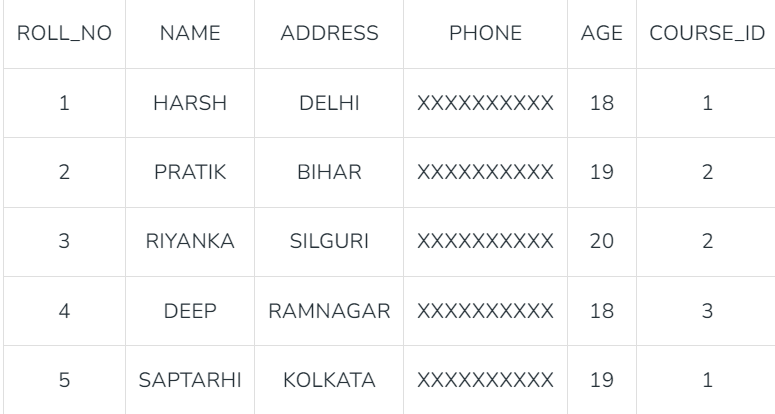


StudentCourse

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;



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.



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



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



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

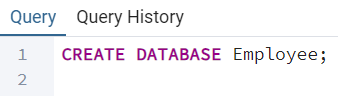


SYNTAX:

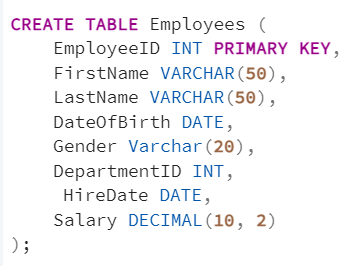
SELECT table1.column1,table1.column2,table2.column1,.... FROM table1 JOIN\_NAME table2 ON table1.matching\_column = table2.matching\_column;

**PRACTICE QUERY:**

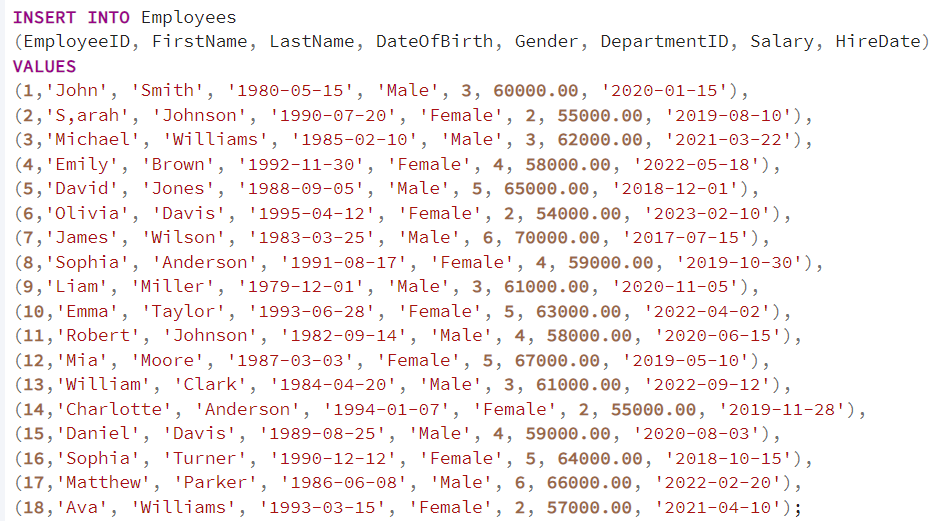
1. Write the correct SQL statement to create a new database called **Employee.**



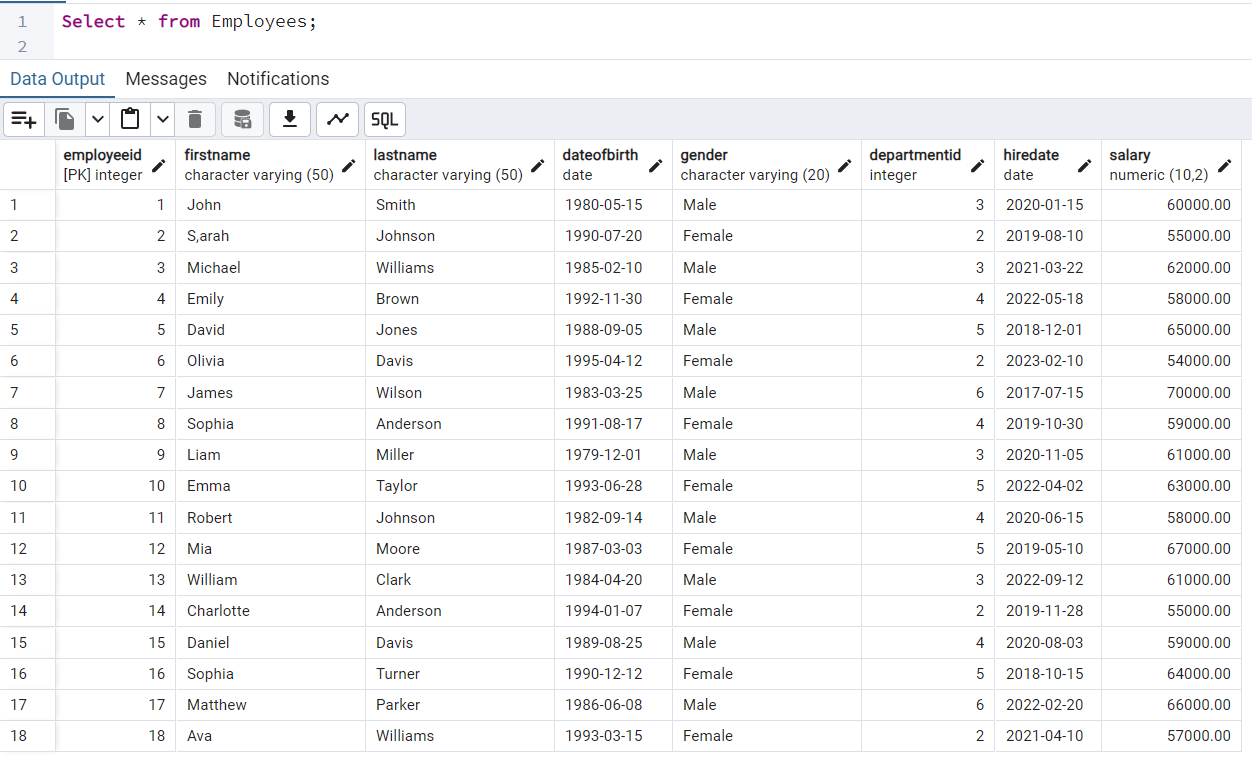
1. Write the correct SQL statement to create a new table called **Employees.**



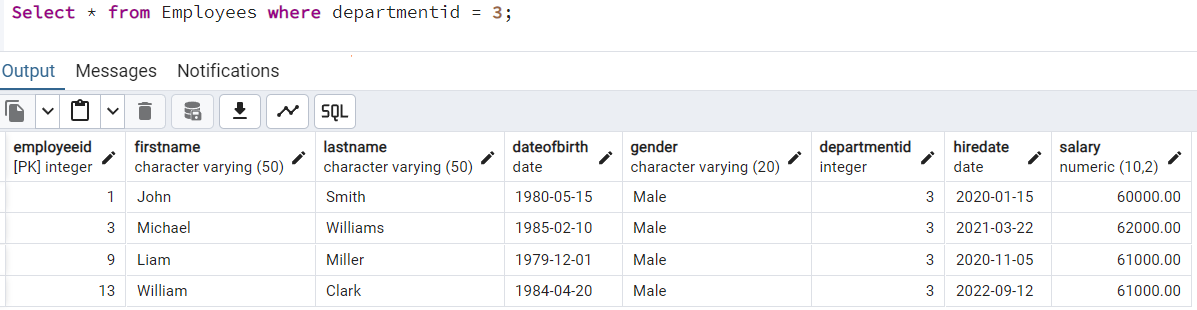
1. Insert a new record in the **Employees** table.



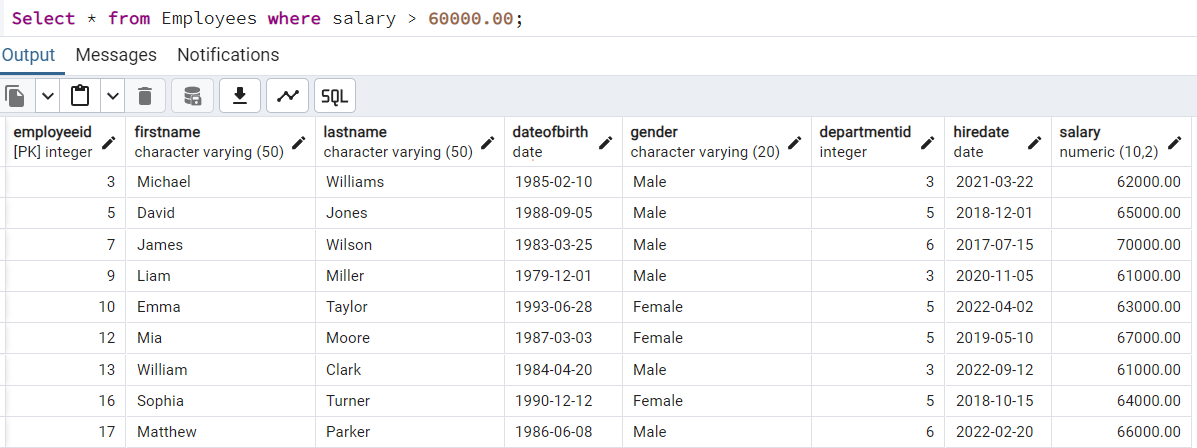
1. Return data from the **Employees** table.



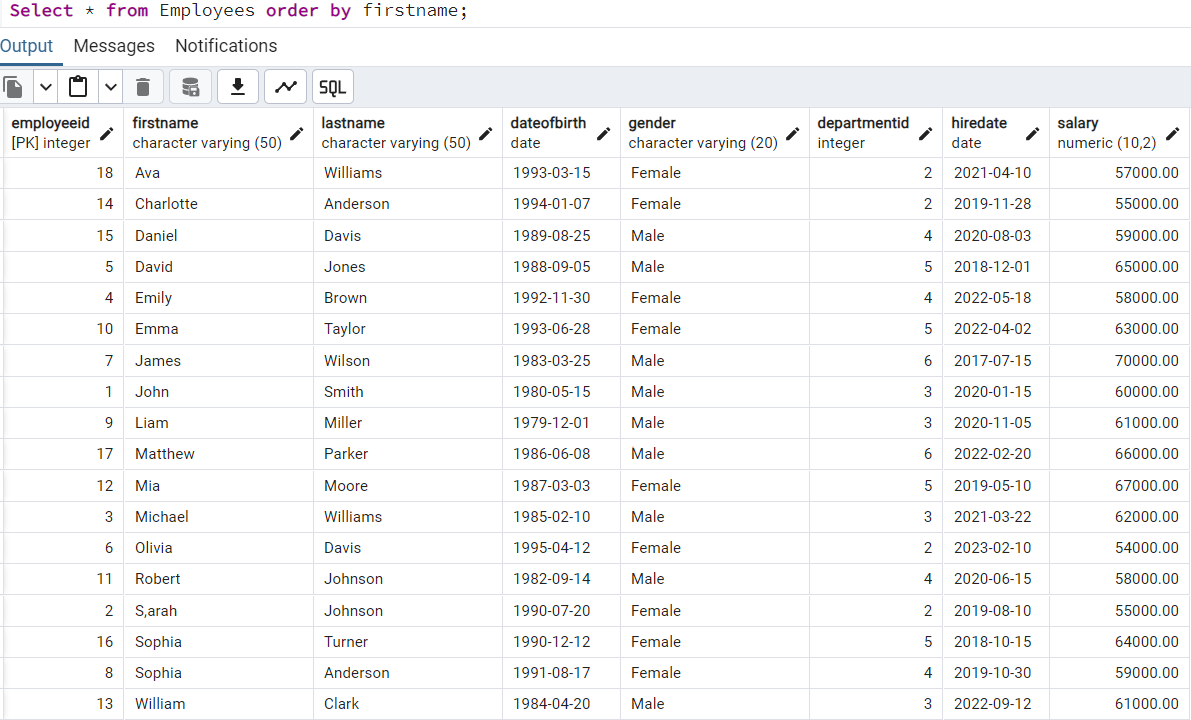
5. Select all records where the department\_id = **3**.



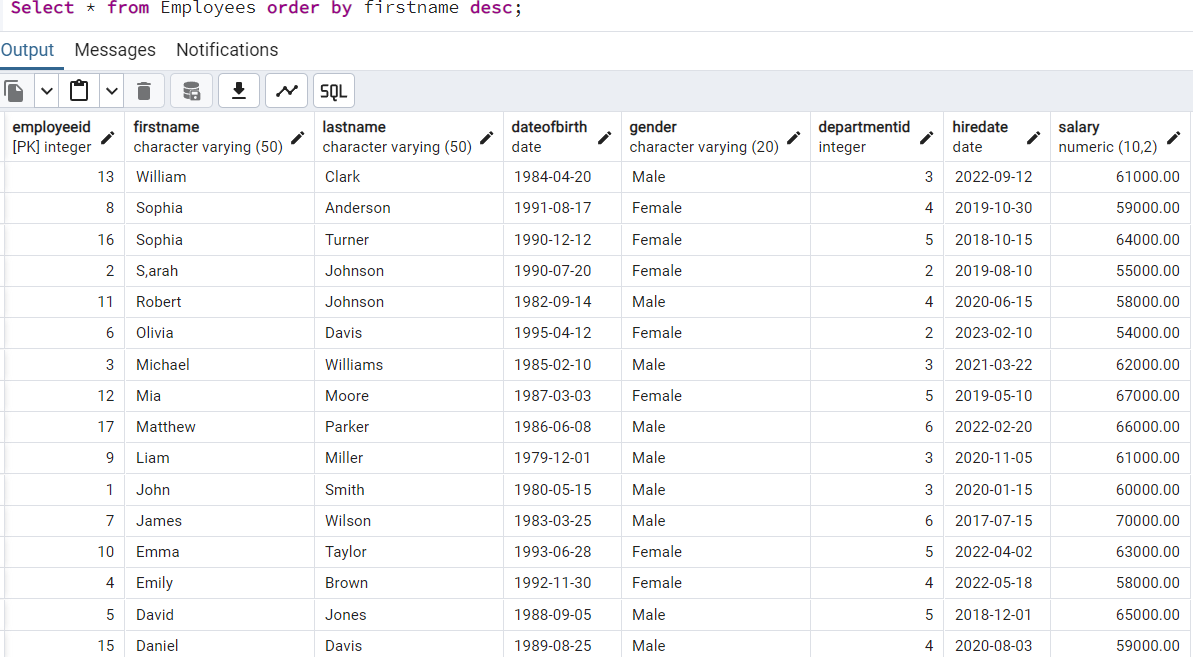
6. Select all records where the salary greater than **60,000**.



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



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



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

