Table of Contents

[1. What is a Query? 1](#_Toc165375553)

[2. What is the SELECT statement? 1](#_Toc165375554)

[3. What is the WHERE clause? 2](#_Toc165375555)

[4. What is the Primary key? 2](#_Toc165375556)

[5. What is a Database? 3](#_Toc165375557)

[6. List the different types of relationships in SQL and give examples. 3](#_Toc165375558)

[7. What is Normalization? 4](#_Toc165375559)

[8. Modify query to show the population of Germany. 4](#_Toc165375560)

[9. Select the query which gives the name of countries beginning with U. 4](#_Toc165375561)

[10. Select the answer which shows the problem with this SQL code - the intended result should be the continent of France: 5](#_Toc165375562)

[11. Select the code which shows the countries that end in A or L. 5](#_Toc165375563)

[12. As per the Given table, select the query which produces table. 6](#_Toc165375564)

# What is a Query?

A query is very important in SQL day to day scenarios,.

Without query we cannot retrieve any information from the database.

Q**uery**is a structured request for information from a database.

We can easily pull information or data from database.

* Queries typically involve selecting certain fields or columns from one or more tables, filtering rows based on specific conditions, sorting the results, and sometimes performing calculations or aggregations on the data.
* Queries can be simple or complex, depending on the information us need and the structure of the database.
* Using queries we can retrieve data from different tables, arrange it, and display it based on specific commands.

# What is the SELECT statement?

The **SELECT statement** in SQL is fundamental for retrieving data from a database. allows us to specify which columns of a table to fetch and the criteria for selecting rows.

* **SELECT**: This keyword indicates that us are about to specify the columns us want to retrieve from the database.
* **column1, column2, ...**: These are the names of the columns us want to retrieve data from. Us can specify one or more columns separated by commas. If us want to retrieve all columns, us can use the asterisk (\*) wildcard symbol instead of listing individual column names.
* **FROM**: This keyword indicates the table or tables from which us want to retrieve data.
* **table\_name**: This is the name of the table from which us want to retrieve data. If us're retrieving data from multiple tables, us would list them here, separated by commas.

The syntax of a SELECT statement is:

Syntax: SELECT column1, column2, ... FROM table\_name;

# What is the WHERE clause?

Using **WHERE clause** in SQL is a component used to filter records based on specific conditions.

The general syntax of a WHERE clause is as follows:

**SELECT**: This keyword indicates that us are about to specify the columns us want to retrieve from the database.

**column1, column2, ...**: These are the names of the columns us want to retrieve data from. Us can specify one or more columns separated by commas. If us want to retrieve all columns, us can use the asterisk (\*) wildcard symbol instead of listing individual column names.

**FROM**: This keyword indicates the table or tables from which us want to retrieve data.

**table\_name**: This is the name of the table from which us want to retrieve data.

**WHERE**: This keyword introduces the WHERE clause, which is used to specify the conditions that must be met for rows to be included in the query results.

**Condition**: This is the condition or set of conditions that determine which rows will be included in the query results. Conditions can include comparisons between column values (such as age > 30), logical operators (such as AND, OR, NOT), and other expressions.

Syntax: SELECT column1, column2, ... FROM table\_name WHERE condition;

# What is the Primary key?

* [A primary key is poweful in SQL](https://www.lifewire.com/primary-key-definition-1019179" \t "_blank)
* It serves as a unique identifier for a particular record within the table.
* The primary key constraint ensures that the values in the primary key column(s) are unique for every row and cannot contain null values.

Example:

In the below example, the "employee\_id" column is specified as the primary key for the "employees" table. This means that each employee must have a unique "employee\_id" value, and the "employee\_id" column cannot contain null values. In a table named "employees," the column "employee\_id" could be designated as the primary key.

Each employee record would have a unique value in the "employee\_id" column, allowing us to uniquely identify and access any employee's information using their respective employee ID.

**CREATE TABLE employees (**

**employee\_id INT PRIMARY KEY,**

**name VARCHAR(50),**

**department VARCHAR(50)**

**);**

.

# What is a Database?

* Database serves as the backbone of modern information systems, enabling organizations to store, retrieve, and manipulate data effectively and securely.
* Databases are widely used in various applications and industries, including business, finance, healthcare, education, and e-commerce, to store and manage critical data for decision-making, analysis, and operations.
* Databases serve as centralized repositories for storing various types of information, such as customer data, product catalogs, financial records, and much more. Data in a database is organized into structured formats, such as tables, rows, and columns.

A **database** is an organized collection of **structured information** or **data**, typically stored in a computer system.

* It is designed to efficiently manage, retrieve, and manipulate large volumes of data according to specific requirements and objectives.

This structure allows for easy retrieval, manipulation, and analysis of data.

Examples of databases include relational databases like MySQL, PostgreSQL, Oracle, and SQL Server, as well as non-relational databases (NoSQL databases) like MongoDB, Cassandra, and Redis.

# List the different types of relationships in SQL and give examples.

**One-to-One Relationship**:

In a one-to-one relationship, **one record** from the first table is associated with **one record** from another table.

Example: Consider an “Employee” table and an “EmployeeDetails” table. Each employee has a corresponding row in the “EmployeeDetails” table that stores passport details. [This forms a one-to-one relationship](https://www.tutorialsteacher.com/sqlserver/tables-relations).

1. **One-to-Many Relationship**:

In a one-to-many relationship, one record from the first table can be linked to one or more rows in another table.

Example: The “Employee” table has an “EmployeeID” (primary key), and the “Address” table stores addresses of employees. Each employee can have multiple addresses (e.g., home, office, permanent). [This is a one-to-many relationship](https://www.tutorialsteacher.com/sqlserver/tables-relations).

1. **Many-to-Many Relationship**:

In a many-to-many relationship, each record in one table can be associated with multiple records in another table, and vice versa.

Example: Consider a library database. Books can have multiple authors, and authors can write multiple books. To model this, we introduce a junction table (e.g., “book authors”) that links books to authors. [It contains foreign keys referencing the primary keys of both the “books” and “authors” tables](https://www.sqltutorial.net/sql-many-to-many-relationship.html).

# What is Normalization?

**There are 3 types of Normalizations**

1. **First Normal Form (1NF): Ensures that each column in a table contains atomic values (values that cannot be divided further) and that each column has a unique name.**
2. **Second Normal Form (2NF): Building on 1NF, this form eliminates partial dependencies. It means that all non-key attributes (attributes that are not part of the primary key) are fully functionally dependent on the entire primary key.**
3. **Third Normal Form (3NF): Building on 2NF, this form eliminates transitive dependencies. It means that all non-key attributes are dependent only on the primary key and not on other non-key attributes.**

**Normalization is the process of organizing data in a database efficiently. This process involves breaking down large tables into smaller, related tables and defining relationships between them.**

**It ensures that data is structured efficiently, making it easier to manage and query.**

# Modify query to show the population of Germany.

**Task:**Modify query to show the population of Germany.

SELECT population FROM world

WHERE name = 'France'

**Solution:**This query will retrieve the population from the "world" table where the country name is 'Germany'.

SELECT population FROM world

WHERE name = 'Germany';

# Select the query which gives the name of countries beginning with U.

**Task:**Select the query which gives the name of countries beginning with U.

**SELECT**name**FROM**world**WHERE**name**LIKE**'%U'

**SELECT**name**FROM**world**WHERE**name**LIKE**'U%'

**SELECT**name**FROM**world**WHERE**name**LIKE**'%u%'

**Solution:**This query uses the **LIKE** operator with the pattern 'U%' to select countries whose names start with the letter 'U'.

Query: SELECT name FROM world

WHERE name LIKE 'U%';

# Select the answer which shows the problem with this SQL code - the intended result should be the continent of France:

**Task:**Select the answer which shows the problem with this SQL code - the intended result should be the continent of France:

SELECT continent FROM world WHERE 'name' = 'France'

1. continent should be 'continent’
2. 'name' should be name
3. 'France' should be "France“
4. 'France' should be France
5. 'France' should be France

**Solution:**The problem with the provided SQL code is that it is using the string literal 'name' instead of the column name ‘name’. Here's the corrected version:

SELECT continent FROM world WHERE name = 'France';

The correct answer is : b) 'name' should be name.

# Select the code which shows the countries that end in A or L.

**Task:**Select the code which shows the countries that end in A or L.

SELECT name FROM world WHERE name LIKE 'a%' AND name LIKE 'l%'

SELECT name FROM world WHERE name LIKE 'a%' OR name LIKE 'l%'

SELECT name FROM world WHERE name LIKE '%a' AND name LIKE '%l'

SELECT name FROM world WHERE name LIKE '%a' OR 'l%'

SELECT name FROM world WHERE name LIKE '%a' OR name LIKE '%l'

**Solution**:This query uses the LIKE operator with patterns '%a' and '%l' to select countries whose names end with the letters 'a' or 'l'. The '%' symbol acts as a wildcard, matching any sequence of characters.

Top of Form

Correct Answer is: SELECT name FROM world

WHERE name LIKE '%a' OR name LIKE '%l';

# As per the Given table, select the query which produces table.

**Task:**

**A table with numbers and a few black text

Description automatically generated with medium confidence**

Given the table on the above, select the query which produces this table below.

**A table with numbers and letters

Description automatically generated**

**FROM**world**SELECT**name, population**BETWEEN**1000000**AND**1250000

**SELECT**name, population**FROM**world**WHERE**population**BETWEEN**1000000**AND**1250000

**FROM** name, population **FROM worldWHERE**population**BETWEEN**1000000**AND**1250000**SELECT**world

**Solution:The correct answer is:**

**SELECT**name, population**FROM**world**WHERE**population**BETWEEN** 1000000 **AND** 1250000