

Table of Contents

[1. List the different types of relationships in SQL and give examples. 2](#_Toc164006540)

[2. What is Normalization? 2](#_Toc164006541)

[3. Modify query to show the population of Germany. 2](#_Toc164006542)

[4. Select the query which gives the name of countries beginning with U. 3](#_Toc164006543)

[5. Select the answer which shows the problem with this SQL code - the intended result should be the continent of France: 3](#_Toc164006544)

[6. Select the code which shows the countries that end in A or L. 4](#_Toc164006545)

[7. As per the Given table, select the query which produces table. 4](#_Toc164006546)

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

1. **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?

**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.**

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.**

# 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 world** **WHERE** population **BETWEEN** 1000000 **AND** 1250000 **SELECT** world

**Solution: The correct answer is:**

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