**SQL Assignment- World**

**Task-1: List the different types of relationships in relational databases and provide examples.**

Relational databases use several types of relationships to define how data in different tables is connected. Here are the main types of relationships, along with examples:

**1. One-to-One (1:1) Relationship**

Each row in one table is linked to one and only one row in another table.

Example: Tables: User and UserProfile.

Explanation: Each user in the User table has one corresponding profile in the UserProfile table. For instance, a user's ID is the primary key in the User table and is also a foreign key in the UserProfile table.

**2. One-to-Many (1:M) Relationship**

A row in one table can be associated with multiple rows in another table.

Example: Tables: Customer and Order.

Explanation: Each customer in the Customer table can place multiple orders in the Order table. The CustomerID is the primary key in the Customer table and a foreign key in the Order table.

**3. Many-to-One (M:1) Relationship**

Multiple rows in one table are associated with a single row in another table.

Example: Tables: Order and Customer.

Explanation: Multiple orders in the Order table are placed by one customer in the Customer table.

**4. Many-to-Many (M:M) Relationship**

Rows in one table can be associated with multiple rows in another table, and vice versa.

Example: Tables: Student and Course.

Explanation: A student can enroll in multiple courses, and a course can have multiple students. This relationship is usually managed with a junction table (e.g., Enrollment) that contains foreign keys from both the Student and Course tables.

**5. Self-Referencing (Recursive) Relationship**

A table is related to itself, meaning rows in the table are related to other rows in the same table.

Example: Table: Employee.

Explanation: An employee may report to another employee. The Employee table might have a ManagerID column that references the EmployeeID in the same table, creating a hierarchy.

**Task-2: What is Normalization and why is it important to database development?**

Normalization is a systematic process used in database design to organize data in a way that reduces redundancy and ensures data integrity. The primary goal of normalization is to structure a database so that each piece of data is stored in only one place, making the database more efficient and easier to maintain.

**Why Normalization is Important in Database Development:**

**1)Reduces Data Redundancy:**

By ensuring that the same data is not stored in multiple places, normalization minimizes duplication. This reduces the amount of storage space needed and decreases the likelihood of inconsistencies in the data.

Ex: In a non-normalized database, customer addresses might be stored in multiple tables. If a customer changes their address, it must be updated in every table, which increases the risk of errors.

**2)Ensures Data Integrity:**

Normalization enforces rules that keep data consistent and accurate. When data is updated, normalized databases ensure that all relevant records are updated accordingly.

Ex: If a customer’s ID is changed, a normalized database ensures that all related orders reflect the change, preventing orphaned records or data mismatches.

**3)Improves Data Organization:**

Normalization helps organize data into logical tables with clear relationships, making the database easier to understand and work with.

Ex: In a normalized database, related data is split into different tables (e.g., customers, orders, products), with clear links between them. This organization makes it easier to retrieve and manage data.

**4)Facilitates Maintenance and Scalability:**

A normalized database is easier to maintain because changes need to be made in only one place. It is also more scalable, allowing for easier addition of new data types or relationships.

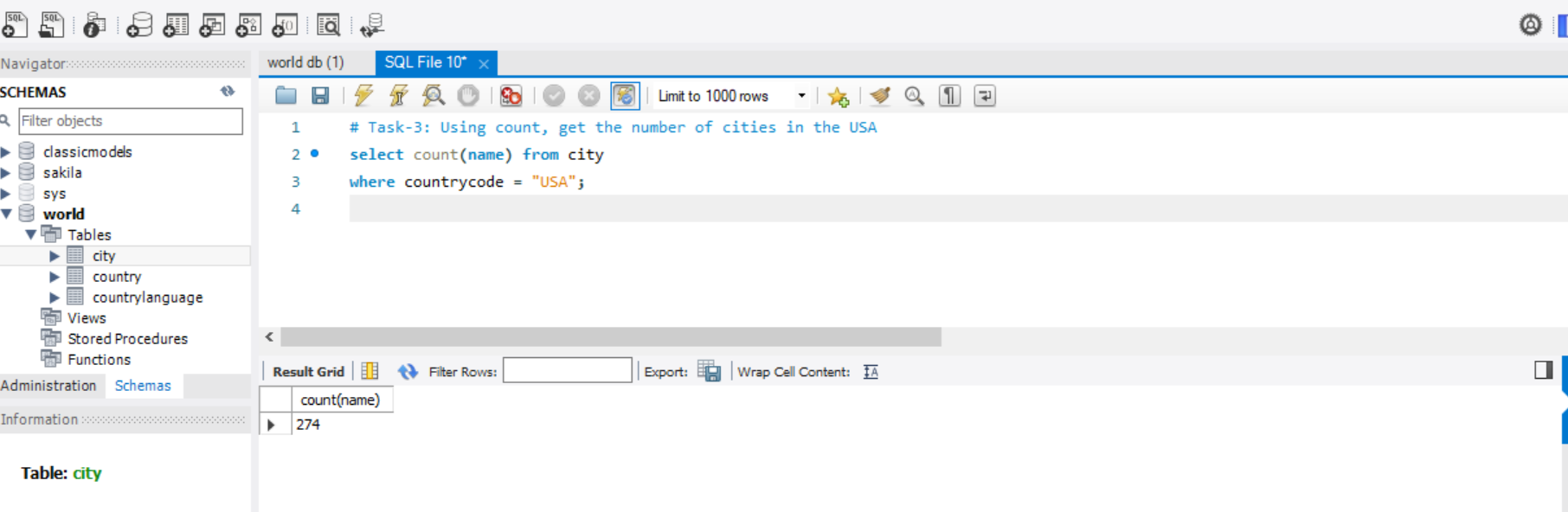
Ex: If a new data field is required, such as a customer’s preferred contact method, it can be added without needing to duplicate existing data.

**5)Optimizes Query Performance:**

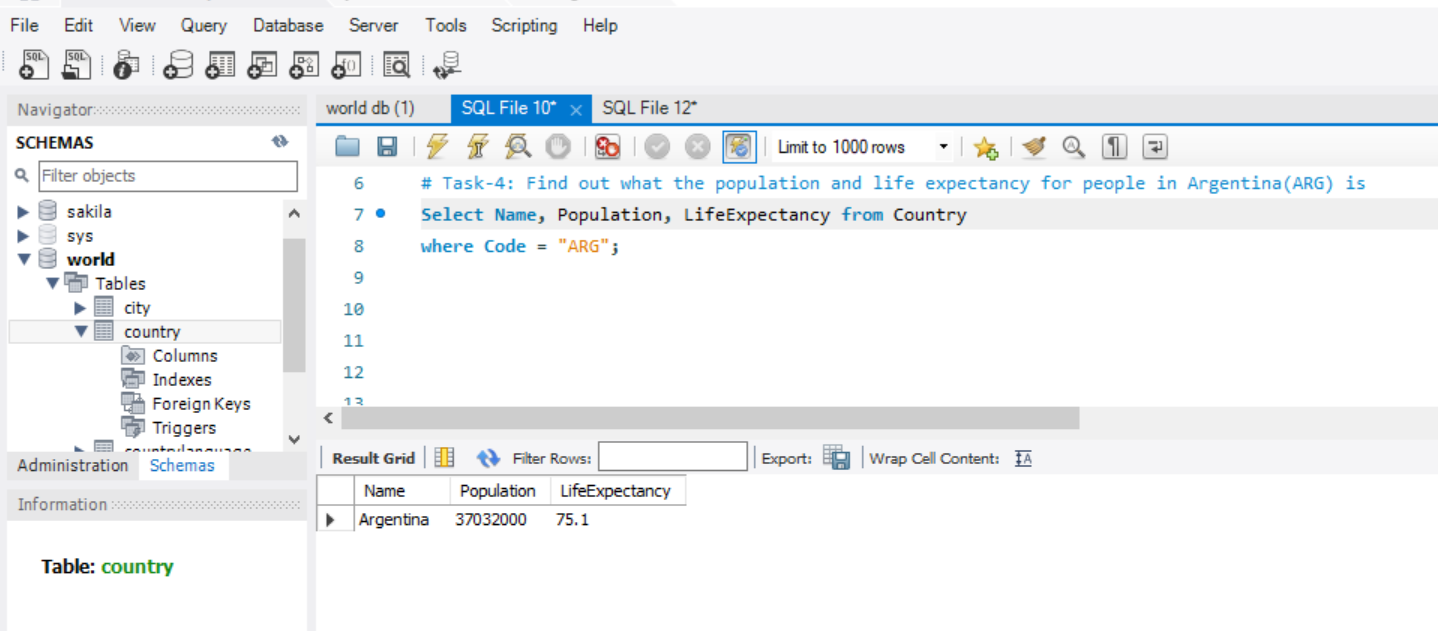
Normalized databases can improve performance by reducing the amount of duplicated data that needs to be processed during queries. Although some queries may become more complex, the overall efficiency of data retrieval can be enhanced.

Ex: A query that looks up customer orders only needs to access the Orders table and join it with the Customers table, rather than searching through a single large, denormalized table.

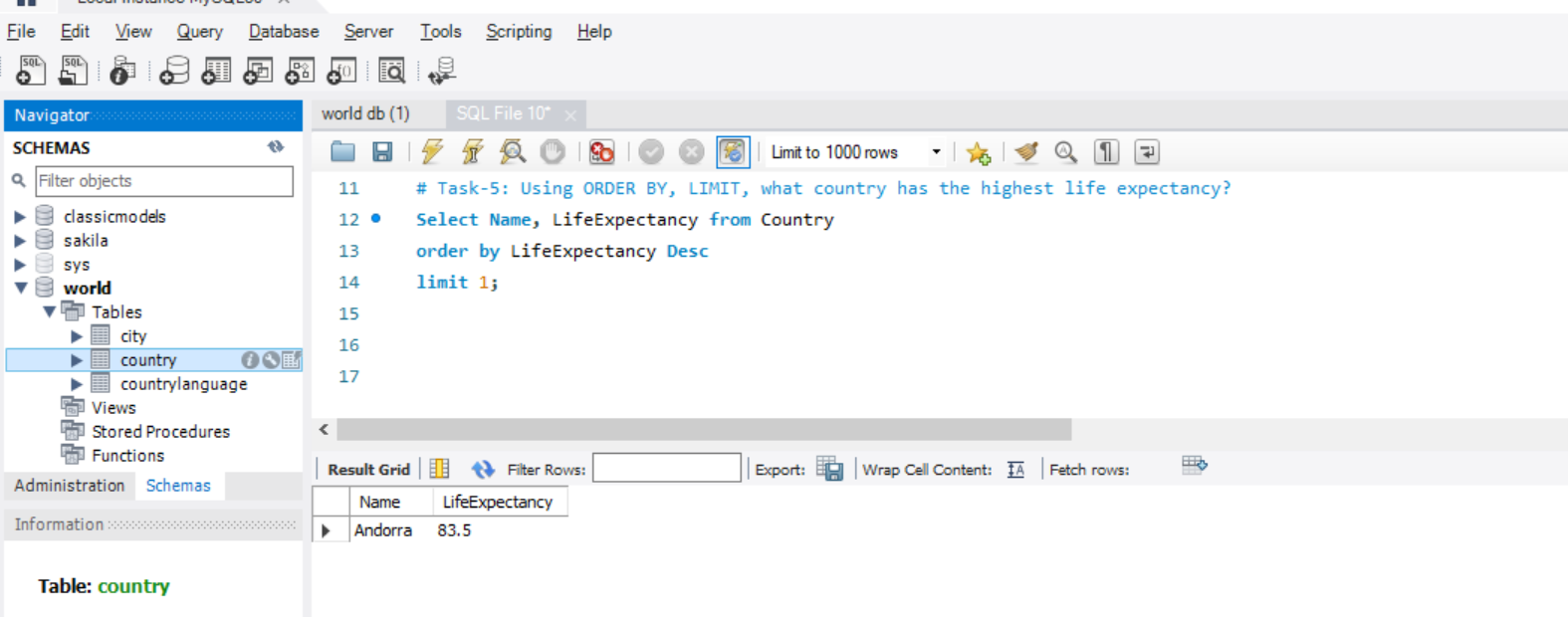
**Task-3: Using count, get the number of cities in the USA.**



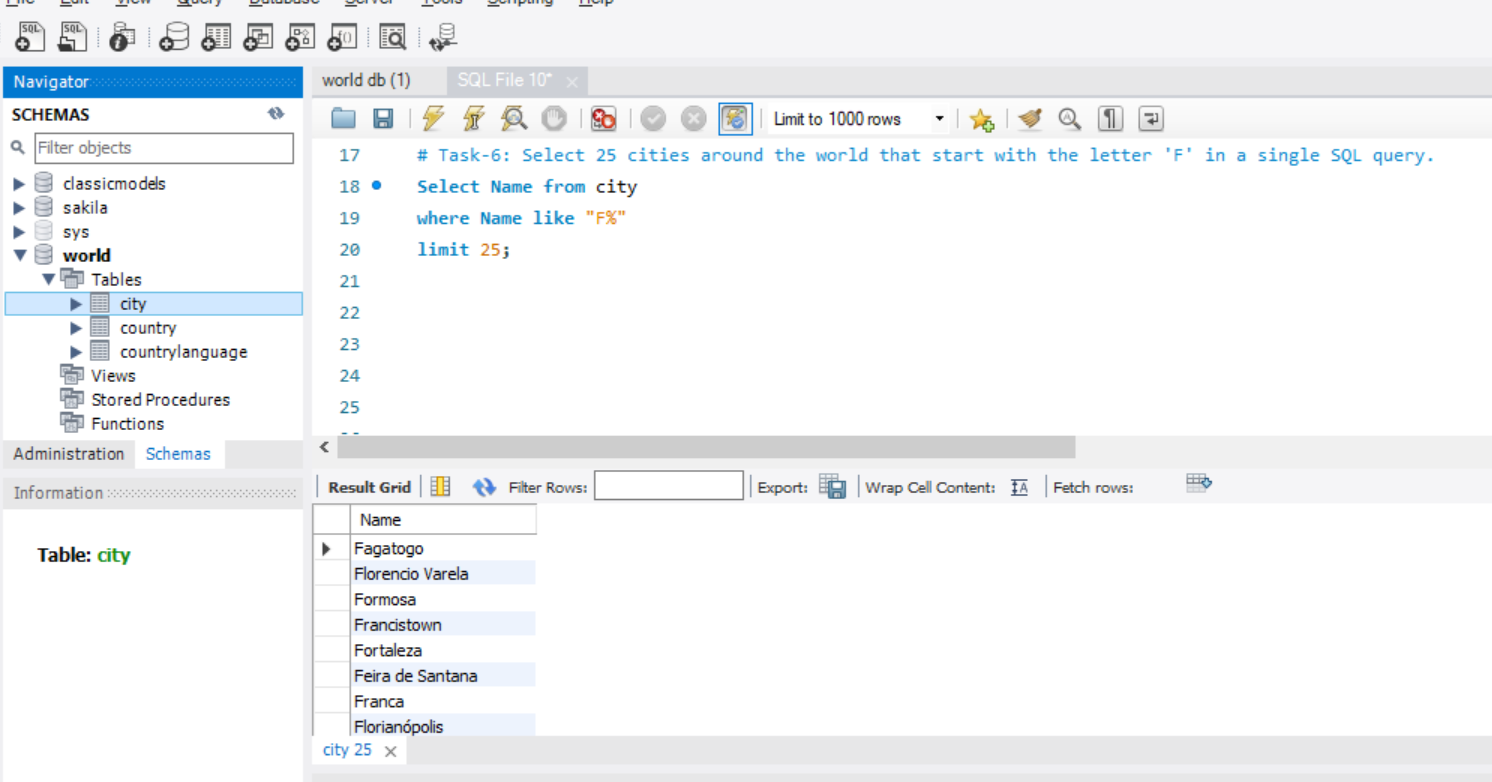
**Task-4: Find out what the population and life expectancy for people in Argentina (ARG) is**



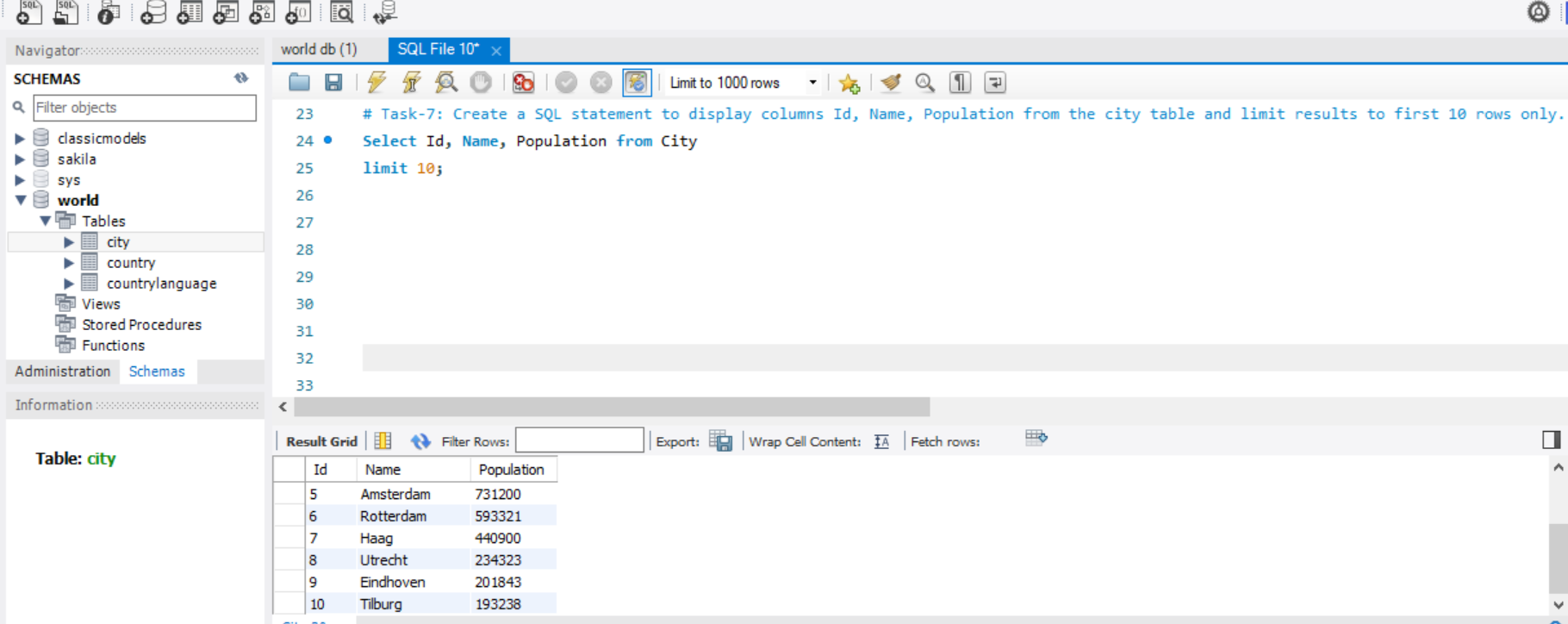
**Task-5: Using ORDER BY, LIMIT, what country has the highest life expectancy?**



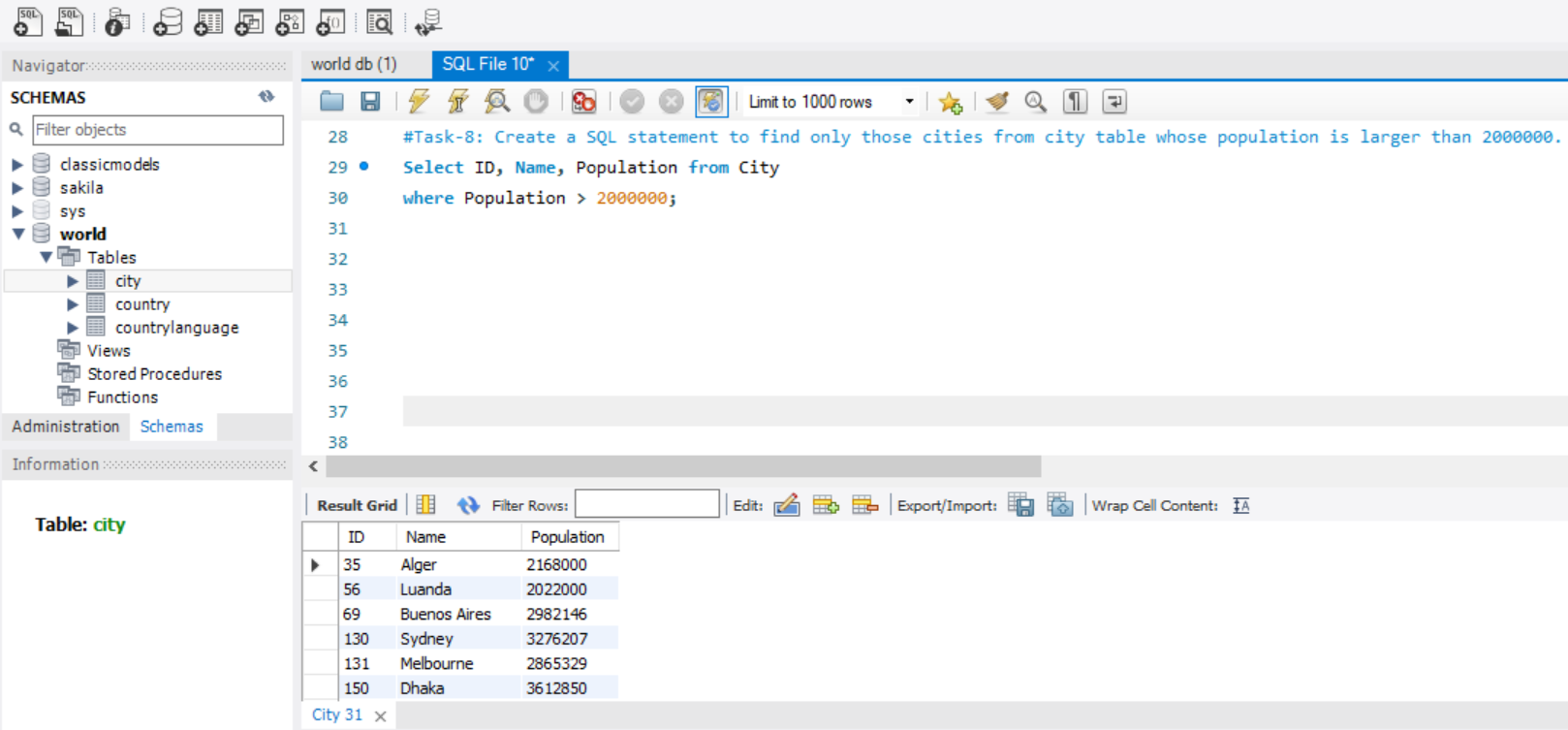
**Task-6: Select 25 cities around the world that start with the letter 'F' in a single SQL query.**



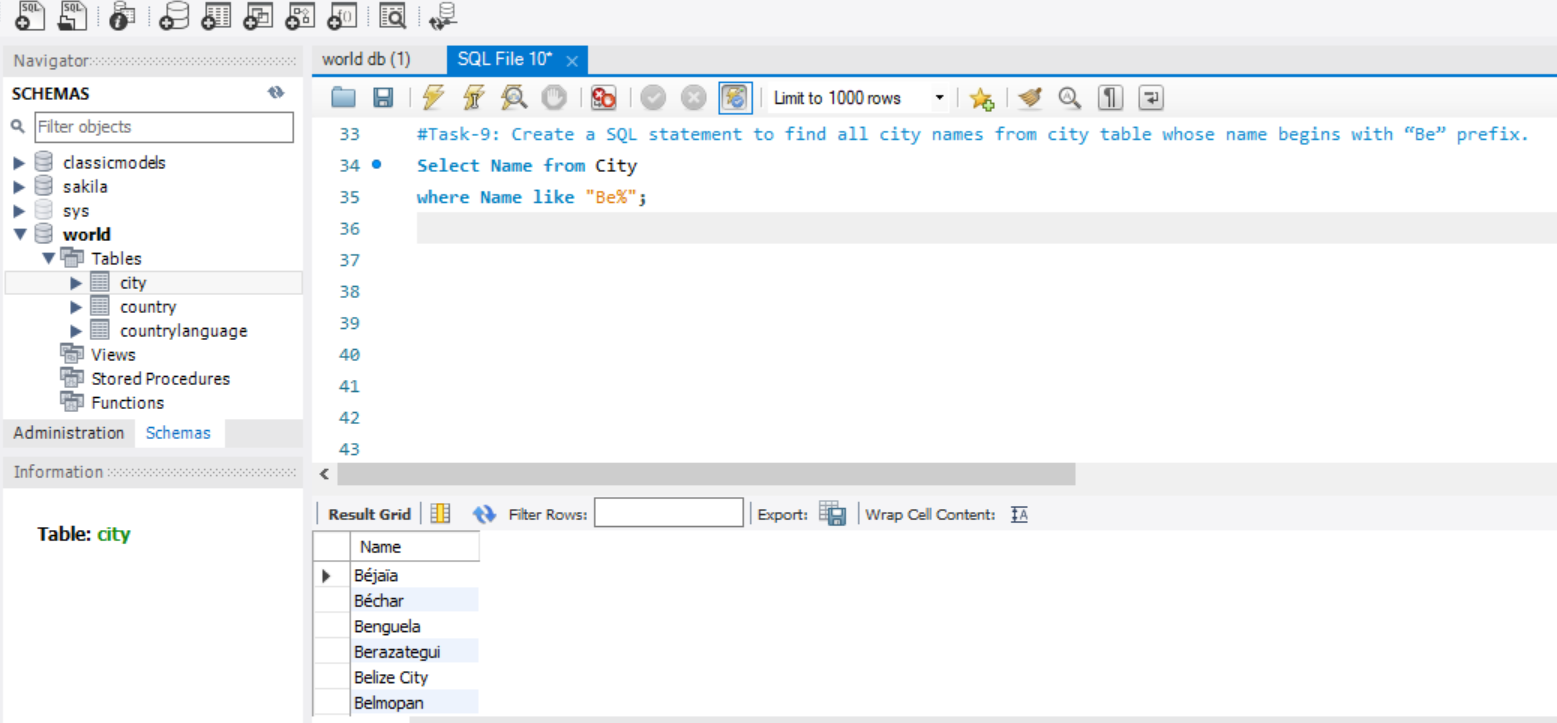
**Task-7: Create a SQL statement to display columns Id, Name, Population from the city table and limit results to first 10 rows only.**



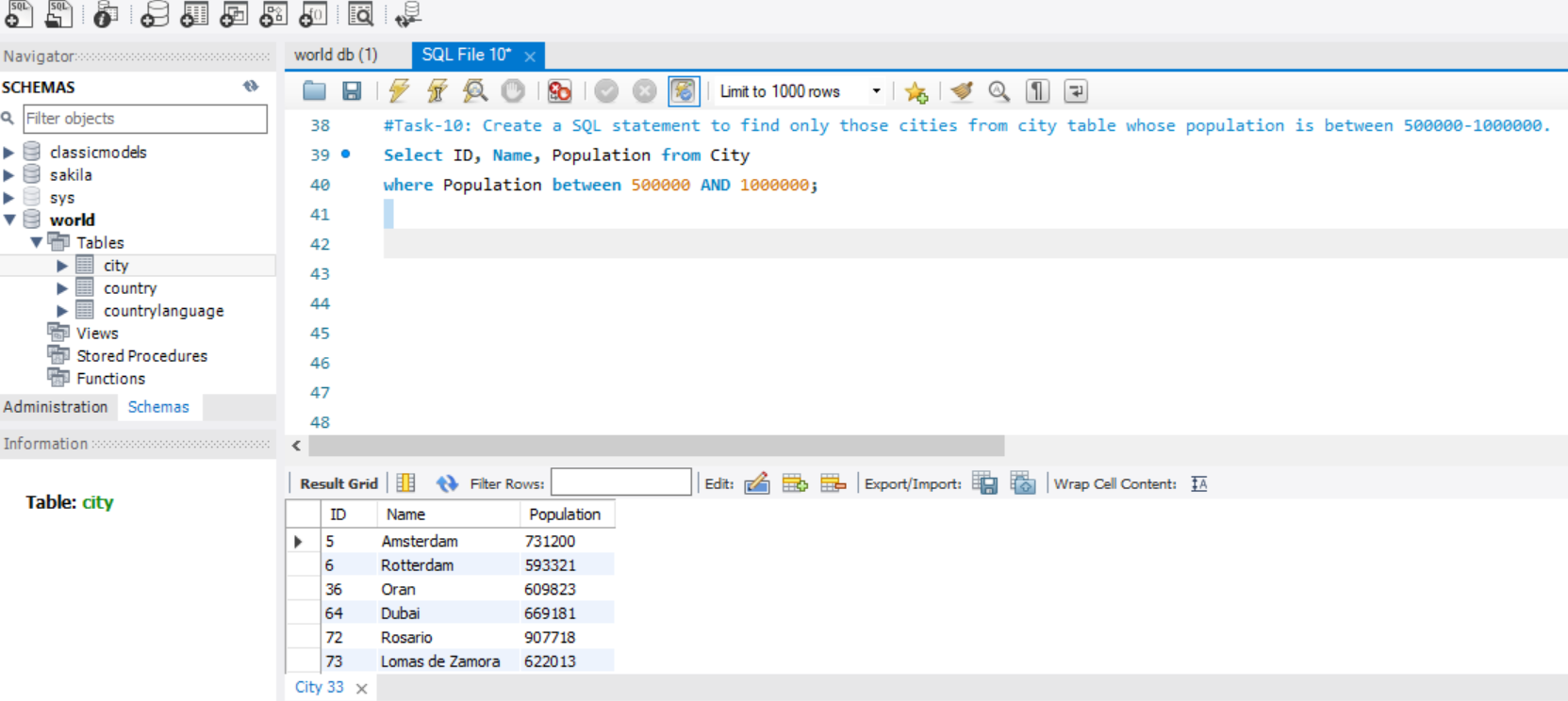
**Task-8: Create a SQL statement to find only those cities from city table whose population is larger than 2000000.**



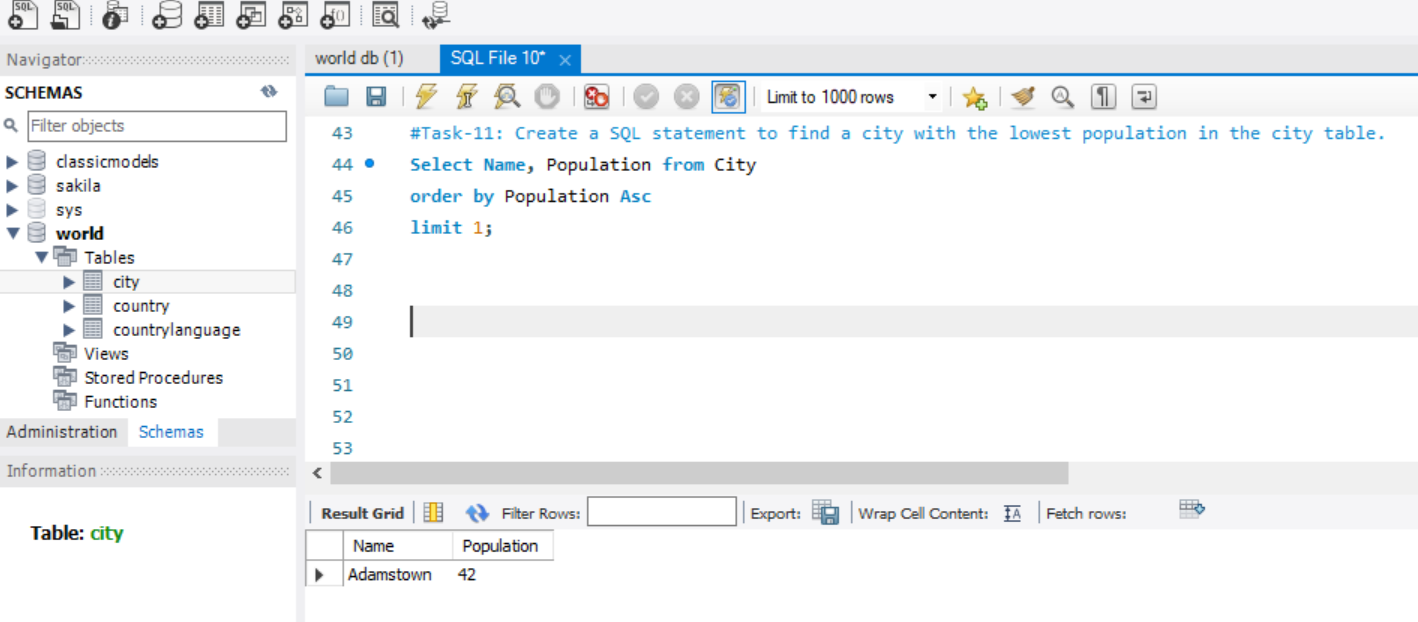
**Task-9: Create a SQL statement to find all city names from a city table whose name begins with “Be” prefix.**



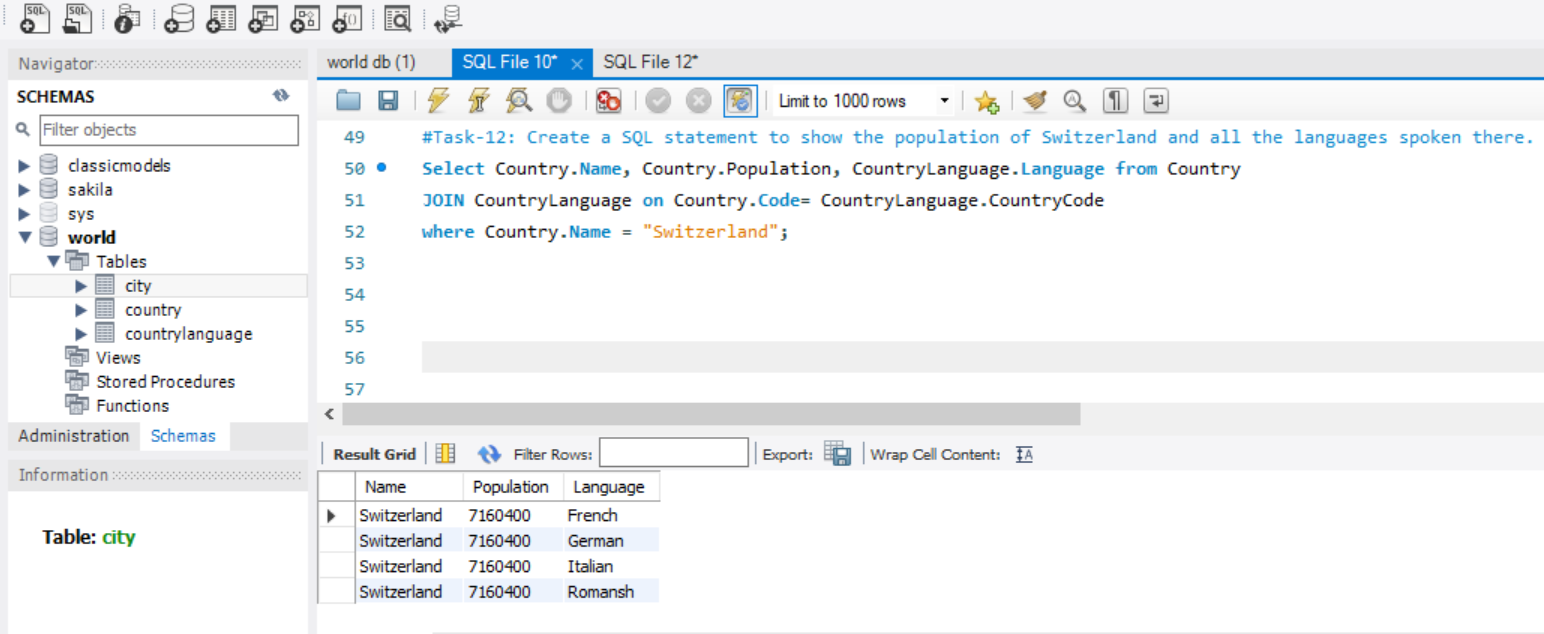
**Task-10: Create a SQL statement to find only those cities from city table whose population is between 500000-1000000.**



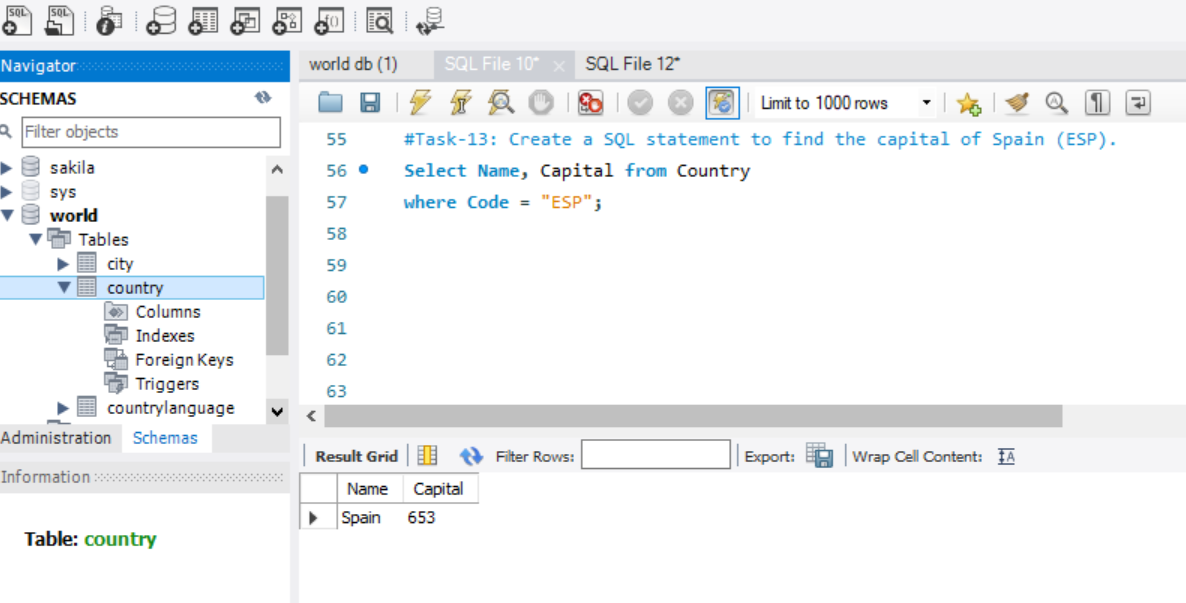
**Task-11: Create a SQL statement to find a city with the lowest population in the city table.**



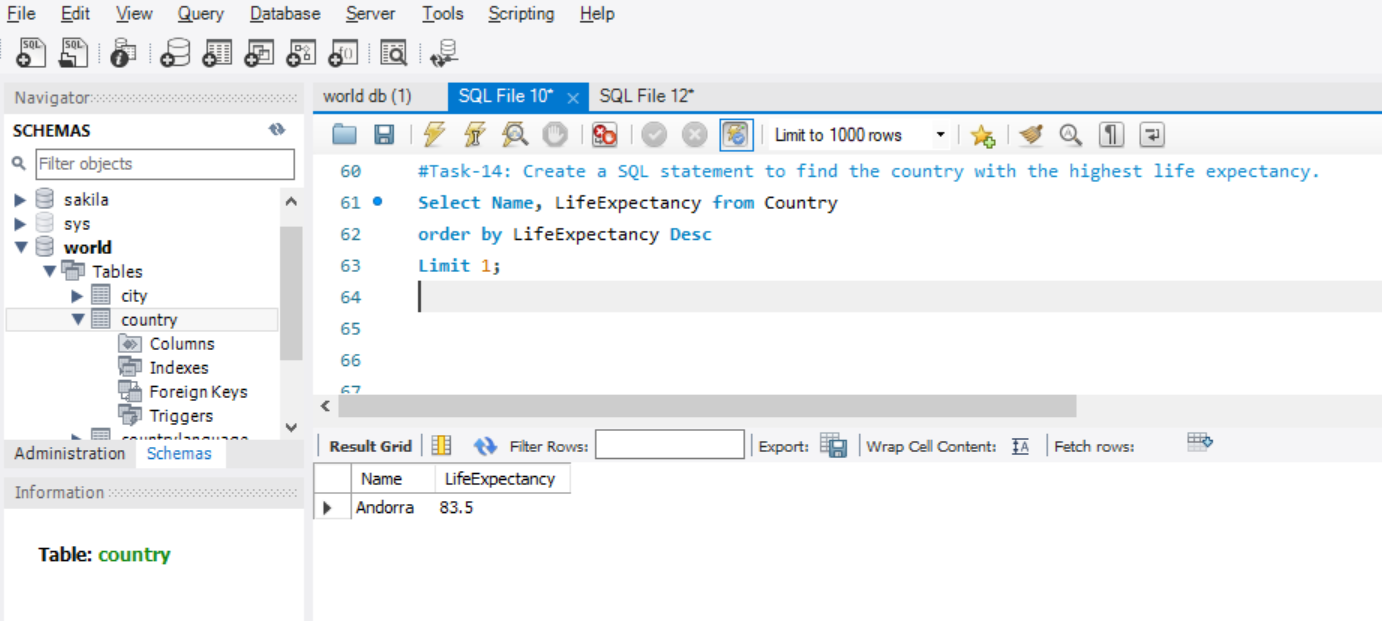
**Task-12: Create a SQL statement to show the population of Switzerland and all the languages spoken there.**



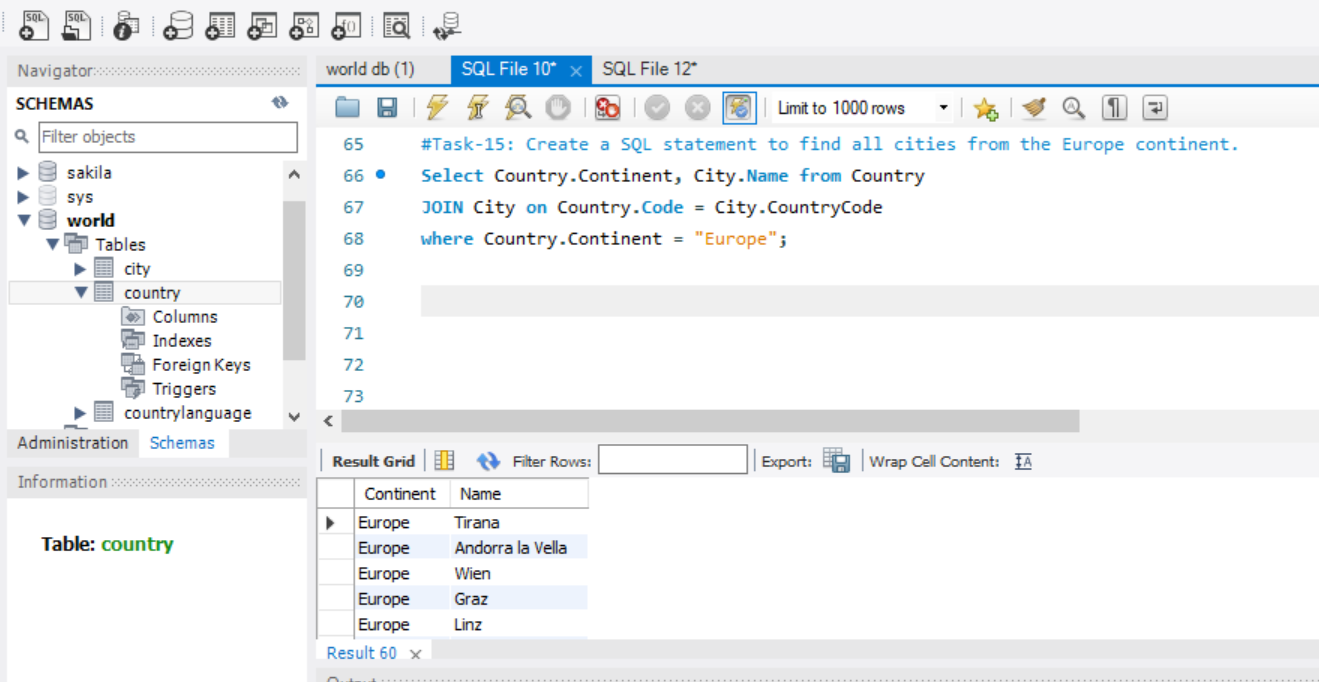
**Task-13: Create a SQL statement to find the capital of Spain (ESP).**



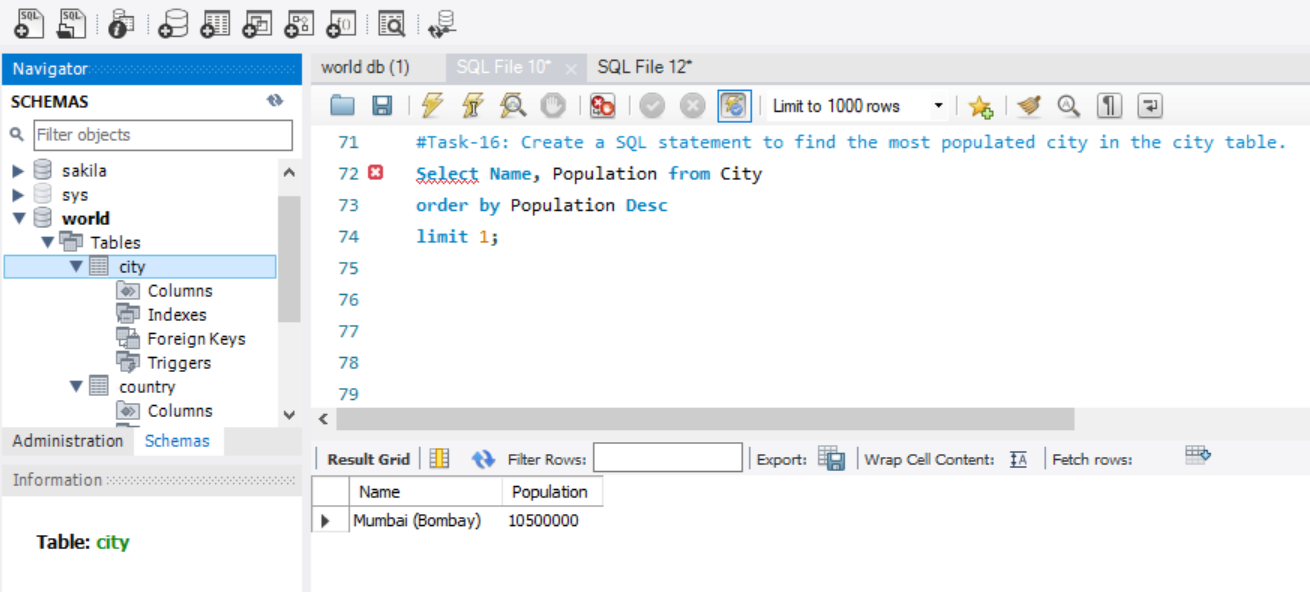
**Task-14: Create a SQL statement to find the country with the highest life expectancy.**



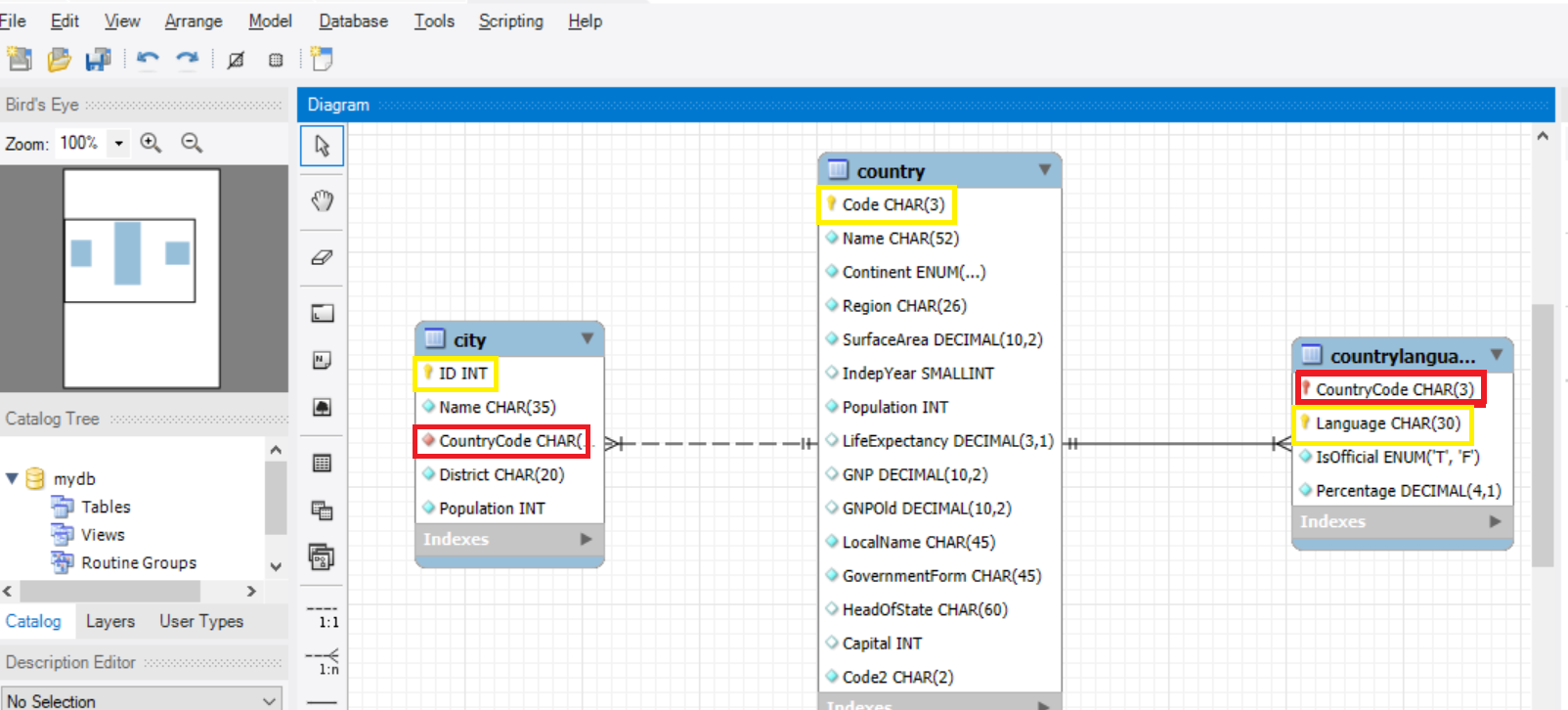
**Task-15: Create a SQL statement to find all cities from the Europe continent.**



**Task-16: Create a SQL statement to find the most populated city in the city table.**



**Task-17:**



•Identify the primary key in the country table.

--> Code CHAR(3)

•Identify the primary key in the city table.

--> ID INT

•Identify the primary key in countrylanguage table.

--> Language CHAR(30)

•Identify the foreign key in the city table.

--> CountryCode CHAR(3)

•Identify the foreign key in countrylanguage table.

--> CountryCode CHAR(3)