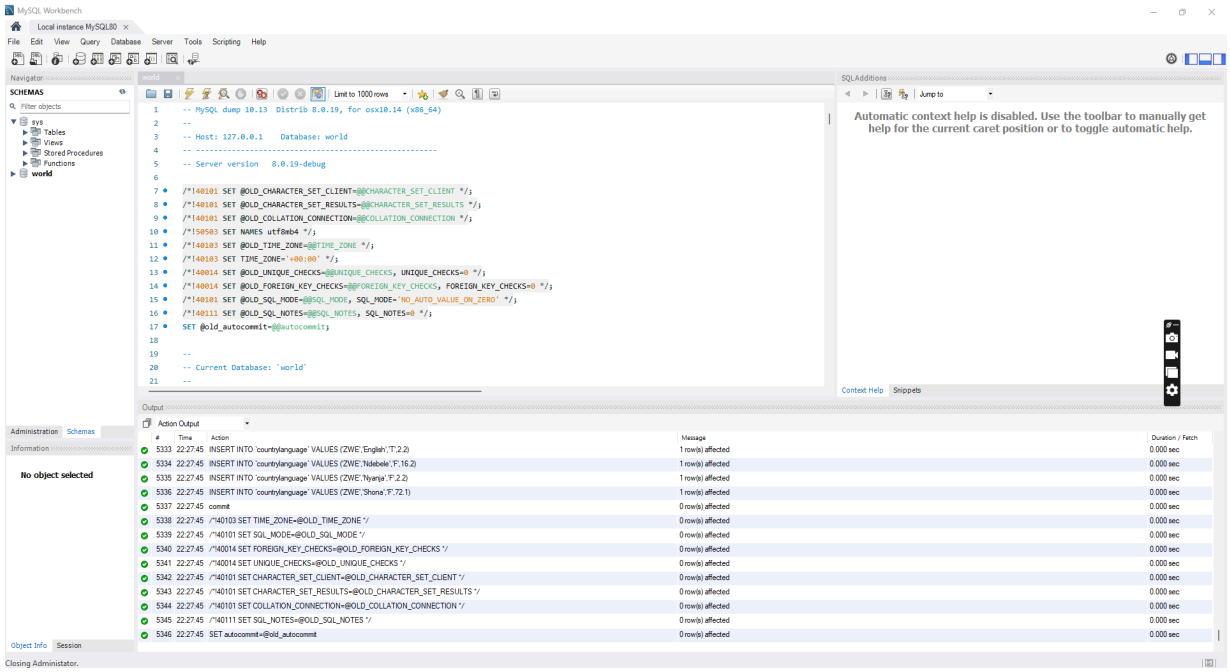
A. Import and Execute

1. For this assignment the “world” database from mysql website resources is used. The world database is a sample database included with MySQL.

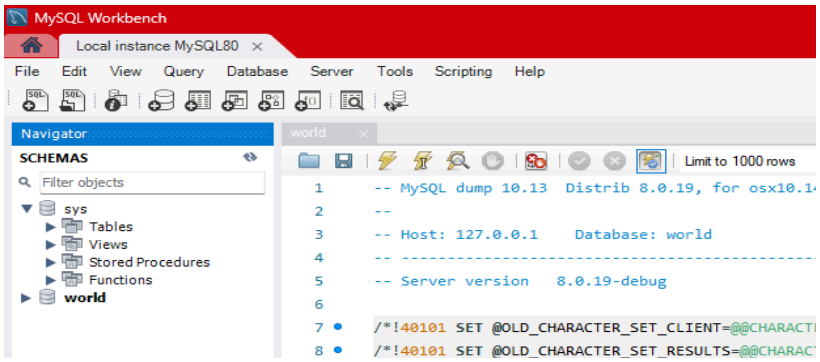
   It's often used as a learning and testing tool for those who are new to working with relational databases and MySQL in particular. Here's a breakdown of its key aspects:

   The primary purpose of the world database is to demonstrate the functionalities of relational databases and familiarize users with MySQL concepts like tables, columns, and querying data.

   Below the SQL Script “world db” is imported and executed.

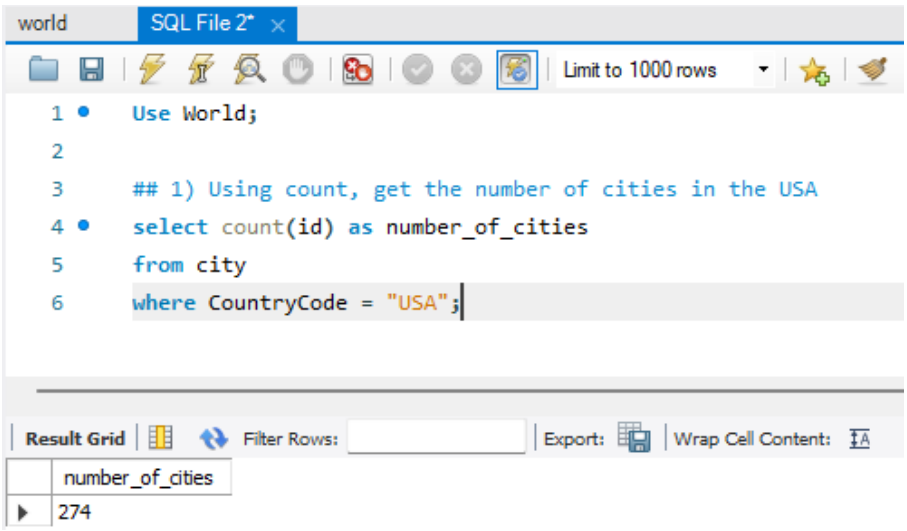


2. Refreshing to confirm if “world” exists in schema.



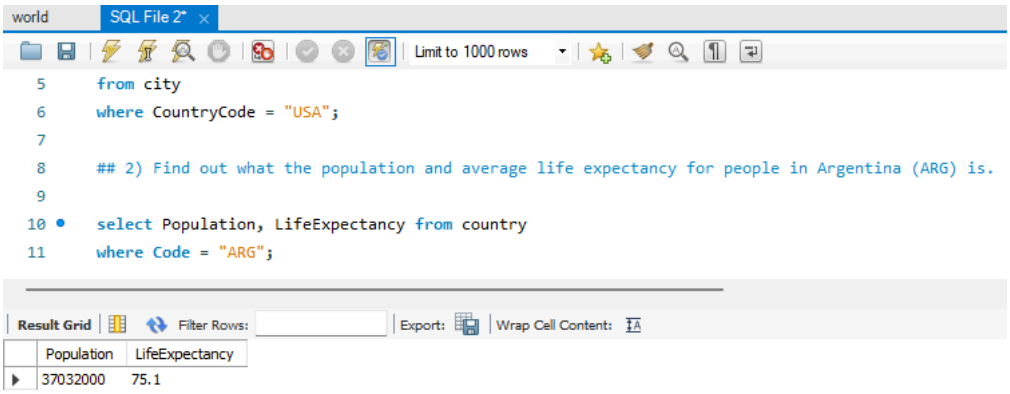
B. Querying Tasks

1. Task #1: Using count, get the number of cities in the USA



The code queries the city table, specifically counting the number of entries where the country code matches "USA", providing the total number of cities within the United States.

2. Task #2: Find out what the population and average life expectancy for people in Argentina (ARG) is.



This query retrieves information from a relational database schema, specifically targeting the "country" table. Its objective is to extract two data points for Argentina: population (total number of residents) and life expectancy (average lifespan in years). The code achieves this by selecting the desired columns ("Population" and "LifeExpectancy") and filtering the results to include only the entry where the "Code" column value matches "ARG", ensuring that information is retrieved exclusively for Argentina. In essence, this query acts as a formal request to the database, asking it to provide these specific data points for Argentina from the "country" table.

3. Task #3: Using ORDER BY, LIMIT, what country has the highest life expectancy?

```
13      ## 3) Using ORDER BY, LIMIT, what country has the highest life expectancy?
14 •    Select Name from country
15      ORDER BY LifeExpectancy desc limit 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

Name
Andorra

This SQL code snippet retrieves the name of the country with the highest life expectancy. It works by first selecting the "Name" column from the "country" table. Then, it sorts all countries in descending order based on their life expectancy values. Finally, it limits the result to just the top row, which represents the country with the longest life expectancy.

4. Task #4: Select 25 cities around the world that start with the letter 'F' in a single SQL query.

```
17      ## 4) Select 25 cities around the world that start with the letter 'F' in a single SQL query.
18 •    select Name from city
19      where Name like "F%" limit 25;
20
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

Name
Fagatogo
Florencio Varela
Formosa
Francistown
Fortaleza
Feira de Santana
Franca
Florianópolis
Foz do Iguaçu
Ferraz de Vasconcelos
Francisco Morato
Franco da Rocha
Fuenlabrada
Faridabad
Firozabad
Farrukhabad-cum-Fat...
Faizabad
Fatehpur
Firenze
Foggia
Ferrara
Forlì
Fukuoka
Funabashi
Fukuyama

This formal SQL code query retrieves a list of city names from a relational database schema. It focuses on the "city" table and aims to identify entries that begin with the letter "F" by employing a pattern matching operator ("like") to identify entries in the "Name" column where the initial character matches "F" ("F%"), and restricts the results to the top 25 matching rows, providing a list of 25 city names starting with "F".

5. Create a SQL statement to display columns `Id`, `Name`, `Population` from the `city` table and limit results to first `10` rows only.

```
21  ## 5) Create a SQL statement to display columns Id, Name, Population from the city table and limit results to first 10 rows only.
22  select ID, Name, Population from city
23  order by ID asc limit 10;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Fetch rows:

ID	Name	Population
1	Kabul	1780000
2	Qandahar	237500
3	Herat	186800
4	Mazar-e-Sharif	127800
5	Amsterdam	731200
6	Rotterdam	593321
7	Haag	440900
8	Utrecht	234323
9	Eindhoven	201843
10	Tilburg	193238
NULL	NULL	NULL

The report specifies the extraction of data from the "city" table, focusing on three specific columns: ID, Name, and Population. The query is structured to retrieve the first 10 entries while ensuring they are ordered by the ID column in ascending order, thus facilitating a sequential arrangement from the lowest to the highest ID numbers.

6. Task #6: Create a SQL statement to find only those cities from city table whose population is larger than 2000000.

```
25  ## 6) Create a SQL statement to find only those cities from city table whose population is larger than 2000000.
26  SELECT * FROM world.city
27  WHERE population > 2000000;
28
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

ID	Name	CountryCode	District	Population
35	Alger	DZA	Alger	2168000
56	Luanda	AGO	Luanda	2022000
69	Buenos Aires	ARG	Distrito Federal	2982146
130	Sydney	AUS	New South Wales	3276207
131	Melbourne	AUS	Victoria	2865329
150	Dhaka	BGD	Dhaka	3612850
206	São Paulo	BRA	São Paulo	9968485
207	Rio de Janeiro	BRA	Rio de Janeiro	5598953
208	Salvador	BRA	Bahia	2302832
209	Belo Horizonte	BRA	Minas Gerais	2139125
210	Fortaleza	BRA	Ceará	2097757
456	London	GBR	England	7285000
554	Santiago de ...	CHL	Santiago	4703954
593	Guayaquil	ECU	Guayas	2070040
608	Cairo	EGY	Kairo	6789479
609	Alexandria	EGY	Aleksandria	3328196
610	Giza	EGY	Giza	2221868
653	Madrid	ESP	Madrid	2879052
712	Cape Town	ZAF	Western Cape	2352121
756	Addis Abeba	ETH	Addis Abeba	2495000
765	Quezon	PHL	National Capital ...	2173831
939	Jakarta	IDN	Jakarta Raya	9604900
940	Surabaya	IDN	East Java	2663820
941	Bandung	IDN	West Java	2429000
1024	Mumbai (Bom...	IND	Maharashtra	10500000
1025	Delhi	IND	Delhi	7206704

city 6

The specified code operates as a query, directing the database to access the "city" table within the "world" database and retrieve all available data fields. Additionally, it includes a condition to filter the results, limiting them to cities with a population surpassing 2 million. In summary, this query essentially asks the database to scour through the "city" table, extracting all information, but exclusively returning entries where the population exceeds the 2 million mark.

7. Task #7: Create a SQL statement to find all city names from city table whose name begins with “Be” prefix.

```
29      ## 7) create a SQL statement to find all city names from city table whose name begins with “Be” prefix.
30 •    SELECT Name FROM world.city
31      WHERE Name LIKE 'Be%';
32
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Name
Béjaïa
Béchar
Benguela
Berazategui
Belize City
Belmopan
Belo Horizonte
Belém
Belford Roxo
Betim
Bento Gonçal...
Belfast
Benoni
Bekasi
Bengkulu
Belgaum
Bellary
Berhampore (...
Beawar
Bettiah
Beerseba
Bene Beraq
Bergamo
Beppu
Beograd
Benxi

The SQL code provided serves as a query to extract data from a relational database schema, with a specific focus on entries from the "city" table. It begins by instructing the database to retrieve solely the "Name" column from the designated table within the "world" database. Further, a filtering condition is applied using the LIKE operator, which allows for pattern matching within the "Name" column. The pattern specified, 'Be%', instructs the database to include city names starting with "Be" followed by any sequence of characters (or no characters at all). In summary, this query prompts the database to search through the "city" table, extracting only the names of cities that begin with the letters "Be," disregarding the subsequent characters.

8. Task #8: •Create a SQL statement to find only those cities from city table whose population is between 500000-1000000.

```
33    ## 8) Create a SQL statement to find only those cities from city table whose population is between 500000-1000000.
34 •  SELECT * FROM world.city
35    WHERE population BETWEEN 500000 AND 1000000;
36
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

ID	Name	CountryCode	District	Population
5	Amsterdam	NLD	Noord-Holland	731200
6	Rotterdam	NLD	Zuid-Holland	593321
36	Oran	DZA	Oran	609823
64	Dubai	ARE	Dubai	669181
72	Rosario	ARG	Santa Fé	907718
73	Lomas de Zamora	ARG	Buenos Aires	622013
74	Quilmes	ARG	Buenos Aires	559249
75	Almirante Brown	ARG	Buenos Aires	538918
76	La Plata	ARG	Buenos Aires	521936
77	Mar del Plata	ARG	Buenos Aires	512880
134	Adelaide	AUS	South Australia	978100
152	Khulna	BGD	Khulna	663340
186	Cotonou	BEN	Atlantique	536827
193	Santa Cruz de la...	BOL	Santa Cruz	935361
194	La Paz	BOL	La Paz	758141
195	El Alto	BOL	La Paz	534466
219	Campinas	BRA	São Paulo	950043
220	São Gonçalo	BRA	Rio de Janeiro	869254
221	Nova Iguaçu	BRA	Rio de Janeiro	862225
222	São Luís	BRA	Maranhão	837588
223	Maceió	BRA	Alagoas	786288
224	Duque de Caxias	BRA	Rio de Janeiro	746758
225	São Bernardo do...	BRA	São Paulo	723132
226	Teresina	BRA	Piauí	691942
227	Natal	BRA	Rio Grande d...	688955
228	Osasco	BRA	São Paulo	659604

The SQL code provided serves as a query to extract data from a relational database schema, specifically targeting entries from the "city" table. It begins by instructing the database to retrieve all columns from the designated table within the "world" database. Further, a filtering condition is applied using the BETWEEN operator along with specified population thresholds (500,000 and 1,000,000). This condition limits the retrieved data to include only cities where the population falls within the defined range, inclusive of the provided boundaries. In summary, this query prompts the database to search through the "city" table, extracting all available information for cities whose population lies between 500,000 and 1,000,000 inhabitants.

9. Task #9: Create a SQL statement to find a city with the lowest population in the city table.

```
37    ## 9) Create a SQL statement to find a city with the lowest population in the city table.
38    SELECT * FROM world.city
39    ORDER BY population ASC
40    LIMIT 1;
41
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Fetch r

ID	Name	CountryCode	District	Population
2912	Adamstown	PCN	–	42
NULL	NULL	NULL	NULL	NULL

The SQL code provided serves as a query to extract data from the "city" table within the "world" database schema. It begins by instructing the database to retrieve all columns from the specified table. Furthermore, the data is sorted in ascending order based on the population column, ensuring that cities with the smallest populations appear first. The LIMIT 1 clause restricts the result set to only one row, effectively returning information about the city with the lowest population. In summary, this query prompts the database to search through the "city" table, sorting cities by population in ascending order and returning details about the city with the smallest population.

10. Task #10: Create a SQL statement to find the capital of Spain (ESP).

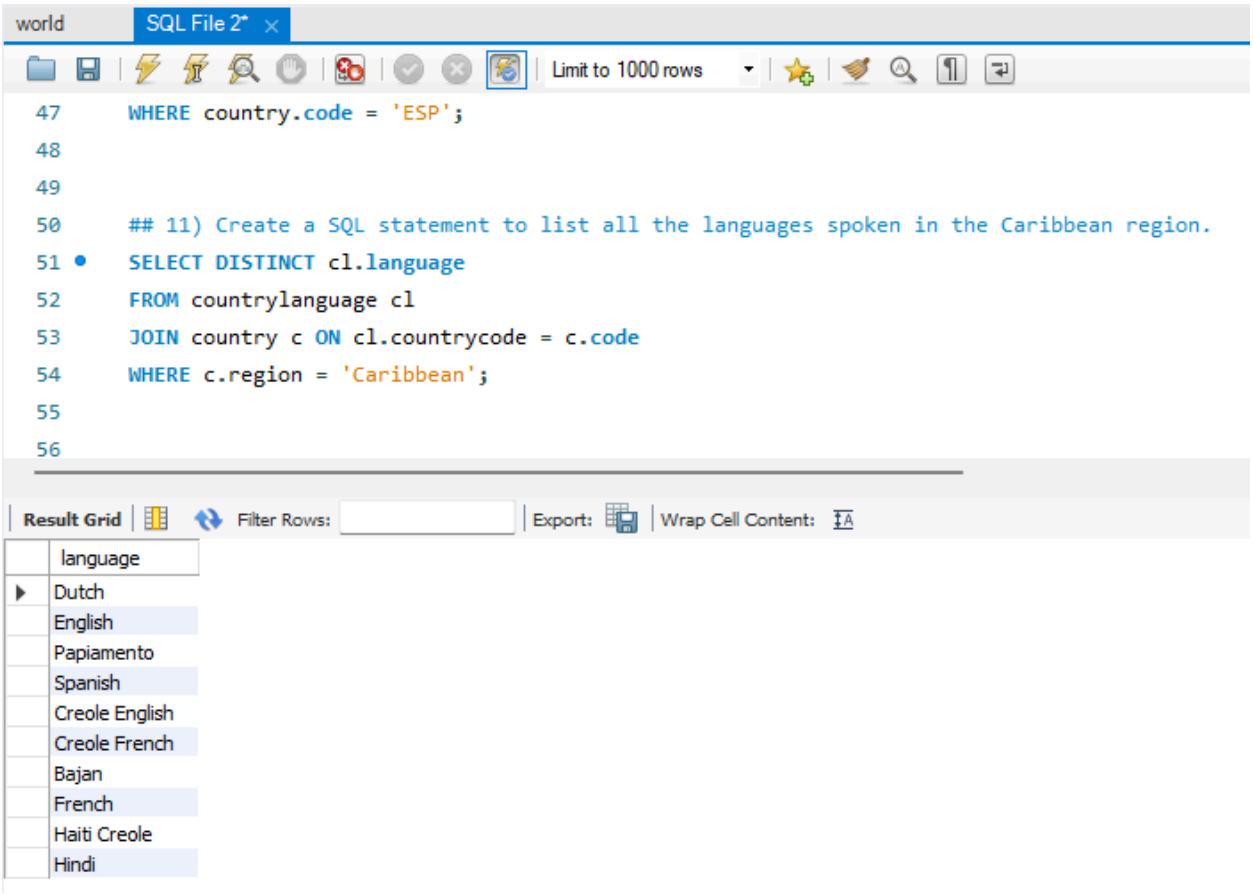
```
42    ## 10) Create a SQL statement to find the capital of Spain (ESP).
43    USE world;
44    SELECT city.name
45    FROM city
46    JOIN country ON city.id = country.capital
47    WHERE country.code = 'ESP';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name
Madrid

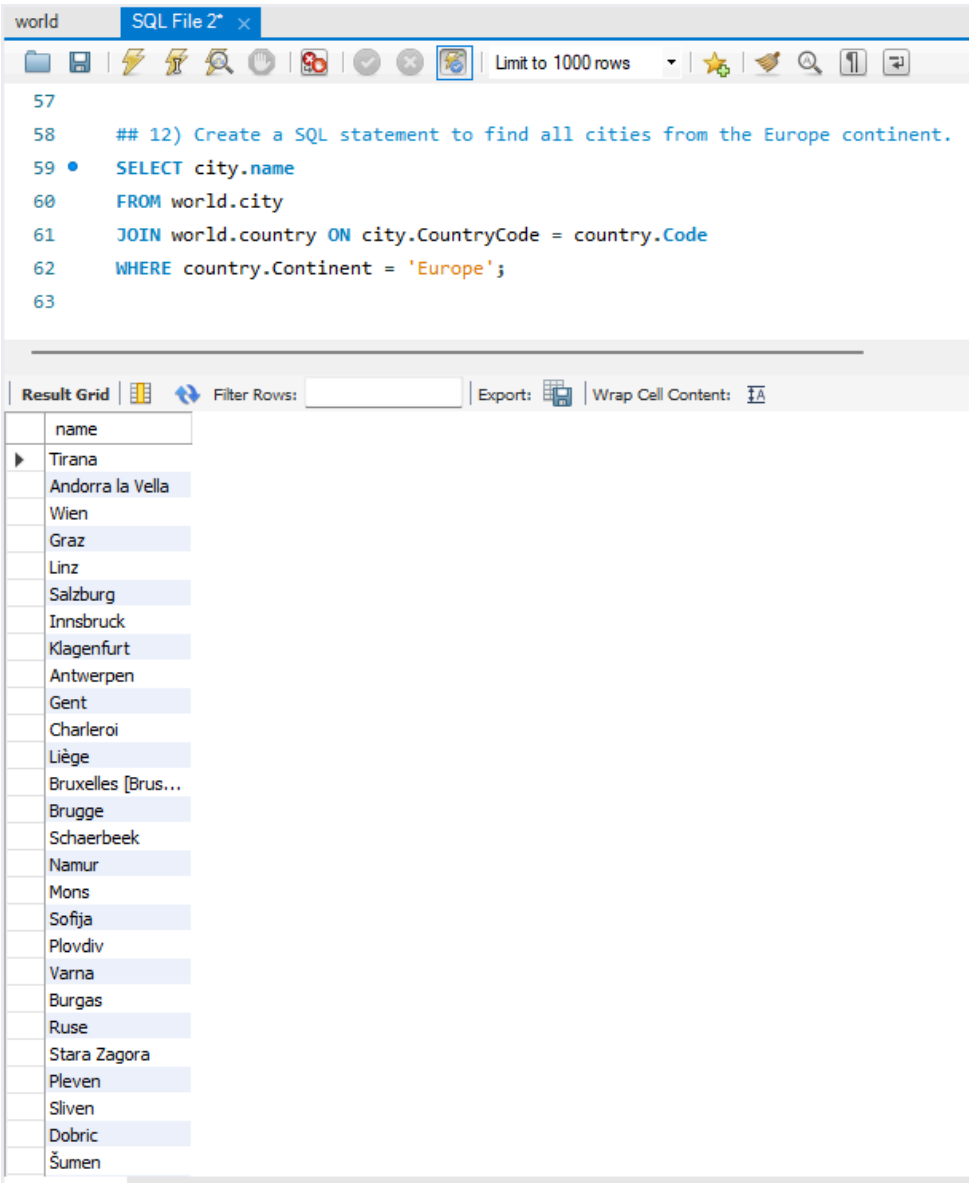
The provided SQL code retrieves data from a relational database schema involving two tables: "city" and "country." It begins by specifying the use of the "world" database for subsequent queries. Then, it selects only the "name" column from the "city" table. The JOIN operation is employed to combine data from the "city" and "country" tables based on a specific condition - where the ID of the city matches the capital value in the country table, suggesting that the "capital" column in the "country" table likely holds city IDs.
Additionally, the query filters the results to include only entries from the "country" table where the "code" column is 'ESP', which presumably represents Spain. In summary, the query aims to retrieve the name of the city that serves as the capital in Spain, utilizing data from both the "city" and "country" tables in the "world" database schema.

11. Task #11: Create a SQL statement to list all the languages spoken in the Caribbean region.



This SQL query is structured to extract data from a relational database schema involving three tables: "countrylanguage", "country", and "language". It starts by selecting the "name" column from the "language" table and then proceeds to join the "countrylanguage" table with the "country" table based on matching country codes, and subsequently joins the "language" table with the "countrylanguage" table based on matching language IDs. The WHERE clause further refines the results by filtering entries from the "country" table to include only those where the "region" column is identified as 'Caribbean'. Ultimately, this query aims to retrieve the names of languages spoken in countries situated within the Caribbean region, utilizing relationships between the tables to gather the necessary information.

12. Task #12: ·Create a SQL statement to find all cities from the Europe continent.



This SQL query orchestrates a data retrieval operation from a relational database schema, focusing on the "city" and "country" tables within the "world" database. It commences by selecting only the "name" column from the "city" table. Following this, a join operation is conducted between the "city" and "country" tables, linking them based on the equivalence of the country code in the "city" table and the code in the "country" table. The WHERE clause is then employed to refine the results, restricting data from the "country" table to entries where the continent is identified as 'Europe'. Ultimately, this query aims to retrieve the names of cities located within countries classified under the European continent, utilizing relational links and filtering conditions to precisely define the dataset of interest.

C. EER Diagram

An Enhanced Entity-Relationship (EER) diagram is a formal graphical representation used in database design to depict the entities (data objects), their attributes (properties), relationships between those entities, and the cardinalities (occurrences) of those relationships. It serves as a high-level blueprint for the overall structure of a database, fostering a clear understanding of the data model and its relationships. EER diagrams extend the capabilities of basic Entity-Relationship (ER) diagrams by incorporating additional features like specialization/generalization hierarchies, allowing for the modeling of complex relationships and inheritance between entities.

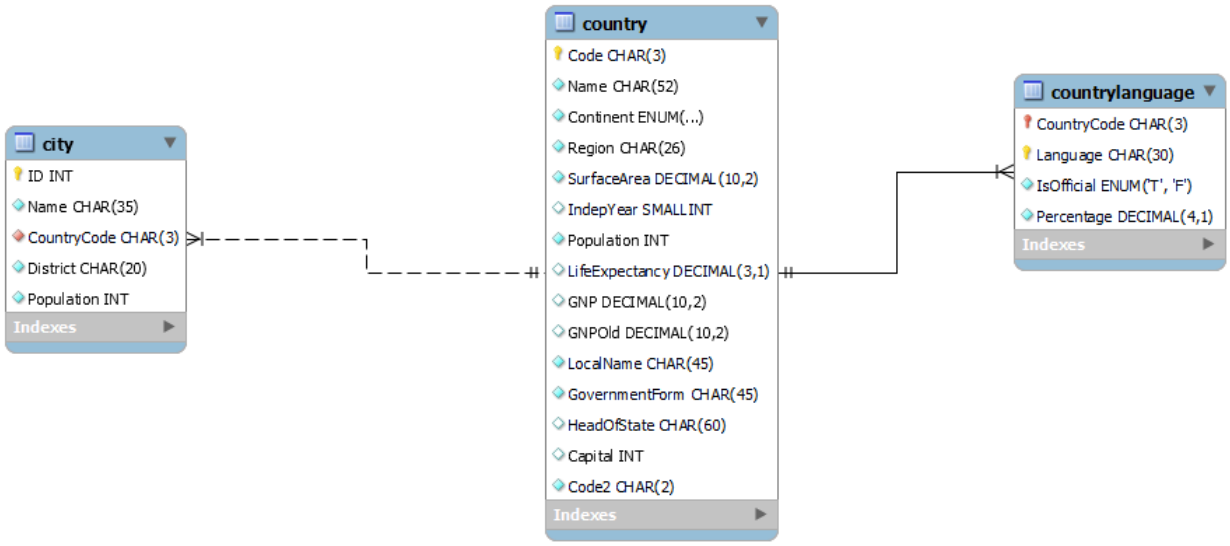
The diagram presents a single primary relationship between the city, country and countrylanguage tables:

A one-to-many relationship means in this schema, one country can have many cities, but a city can only belong to one country:

- The city table likely has a column named Country_Code (or similar).
- This Country_Code would be a foreign key, referencing the Code column in the country table.
- Every city record in the city table would have a Country_Code value that matches an existing Code value in the country table, ensuring accurate linking.

A many-to-many relationship means in this schema, multiple countries can be connected to multiple languages. This table would have two foreign keys:

- One foreign key referencing the Code column from the country table.
- Another foreign key referencing a column Language the countryLanguage table that stores details about individual languages.



D. Primary and Foreign Keys

In the "world" database schema:

- In the "country" table, the primary key is usually the "Code" column. This column contains unique country codes, serving as the main identifier for each country.
- For the "city" table, the primary key is often the "ID" column. Each city is assigned a unique ID, allowing for easy identification and distinction between different cities.
- In the "countrylanguage" table, the primary key typically consists of a combination of the "CountryCode" and "Language" columns. Together, these columns represent unique language entries associated with each country.
- Within the "city" table, the foreign key is usually the "CountryCode" column. This column links each city to its respective country by referencing the primary key "Code" column in the "country" table.
- Similarly, in the "countrylanguage" table, the foreign key is typically the "CountryCode" column. This column establishes a connection to the "country" table by referencing its primary key "Code" column, thereby indicating which countries correspond to the listed languages.