SQL Assignment Parts 1 & 2

Lee Hembling

Contents

[Part 1 – create-databases.sql 2](#_Toc165459314)

[Importing create-databases.sql 2](#_Toc165459315)

[Query 1 3](#_Toc165459316)

[Query 1 Continued 3](#_Toc165459317)

[Query 2 4](#_Toc165459318)

[Task 1a 5](#_Toc165459319)

[Task 1b 5](#_Toc165459320)

[Task 2 6](#_Toc165459321)

[Task 3 6](#_Toc165459322)

[Task 4 7](#_Toc165459323)

[Task 5 7](#_Toc165459324)

[Task 6 7](#_Toc165459325)

[EER Diagram 8](#_Toc165459326)

[Part 2 – World db.sql 10](#_Toc165459327)

[Import and execute 10](#_Toc165459328)

[Task 1 11](#_Toc165459329)

[Task 2 11](#_Toc165459330)

[Task 3 12](#_Toc165459331)

[Task 4 12](#_Toc165459332)

[Task 5 13](#_Toc165459333)

[Task 6 14](#_Toc165459334)

[Task 7 15](#_Toc165459335)

[Task 8 15](#_Toc165459336)

[Task 9 16](#_Toc165459337)

[Extra task 1 16](#_Toc165459338)

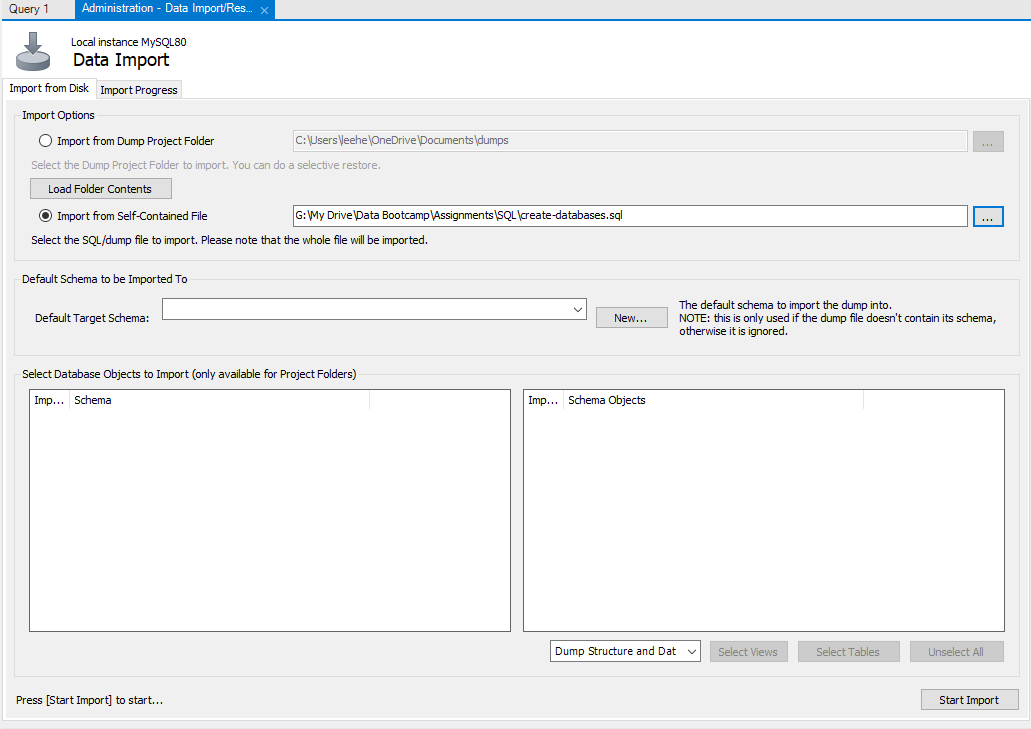
[Extra task 2 17](#_Toc165459339)

[Extra task 3 19](#_Toc165459340)

[EER Diagram 20](#_Toc165459341)

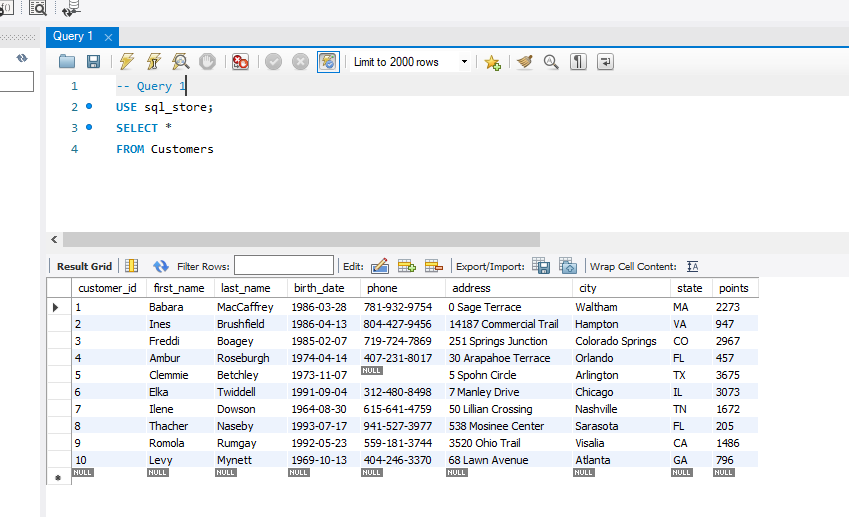
## Part 1 – create-databases.sql

## Importing create-databases.sql



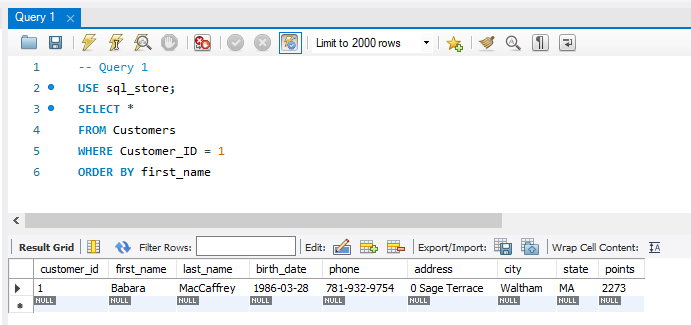
Importing the ‘Create-databases.sql’ script into MySQL Workbench.

## Query 1



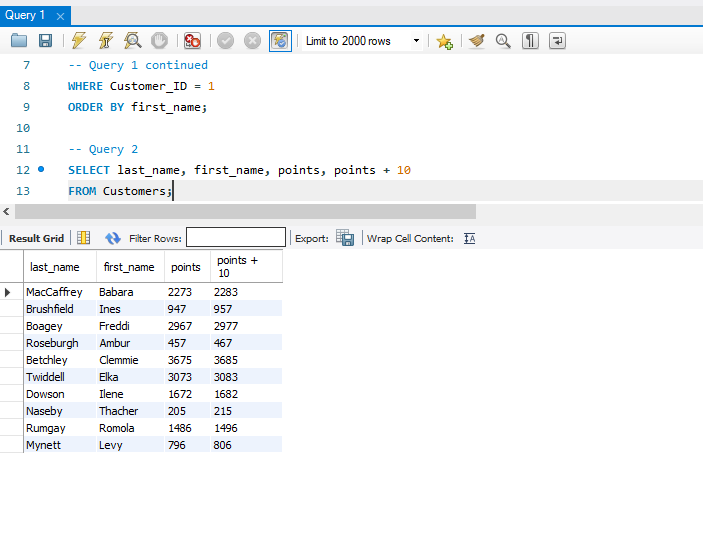
Using **SELECT \*** returns all results from the **Customers** table

## Query 1 Continued



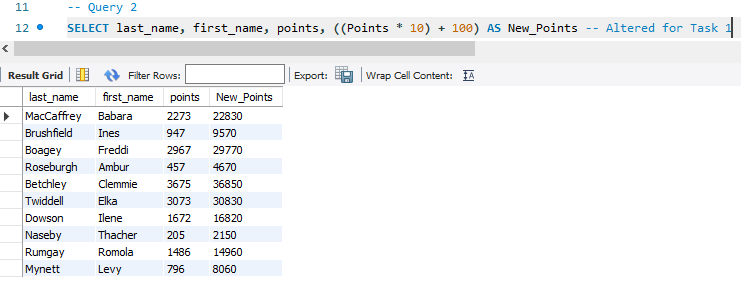
Lines 5 & 6 add a constraint and a sort. In this case the result will only be where the **Customer\_ID** is equal to 1 and the results are in order of **first\_name**. The result is only 1 row.

## Query 2



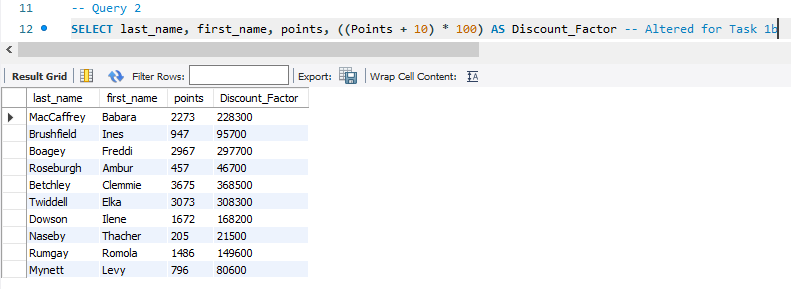
Query 2 displays **the last\_name, first\_name, points and point + 10 columns** from the **Customers** table.

## Task 1a



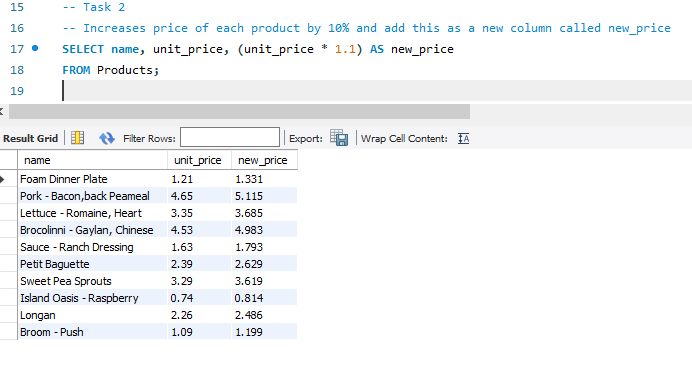
Task 1a asked to change the points to read points times by 10 and plus 100. I used an **alias (AS New\_Points)** to add this to the results.

## Task 1b



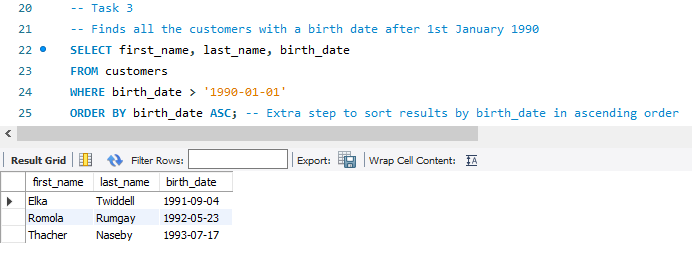
Query 2 has been altered to create a **discount\_factor** column.

## Task 2



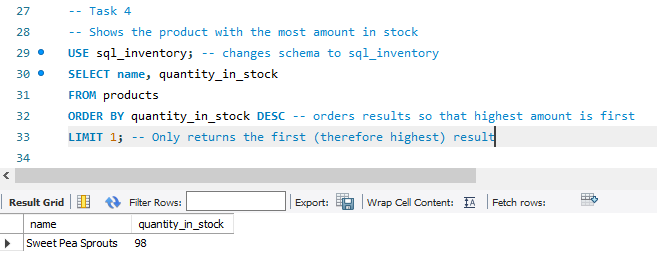
Task 2 Increases price of each product by 10% and add this as a new column called **new\_price**.

## Task 3



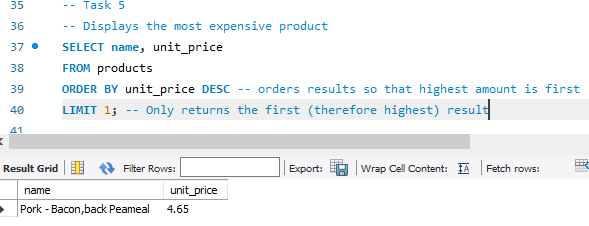
This query finds all customers born after the 1st January 1990. I also used **ORDER BY** to sort the results by **birth\_date** in **ascending** order

## Task 4



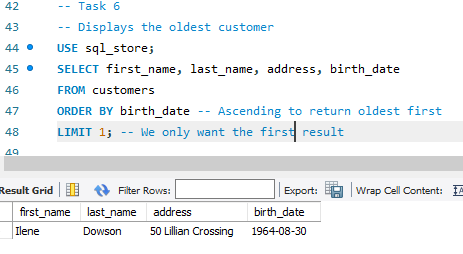
The comments in the screenshot above explain what is happening to display the product with the highest amount in stock.

## Task 5



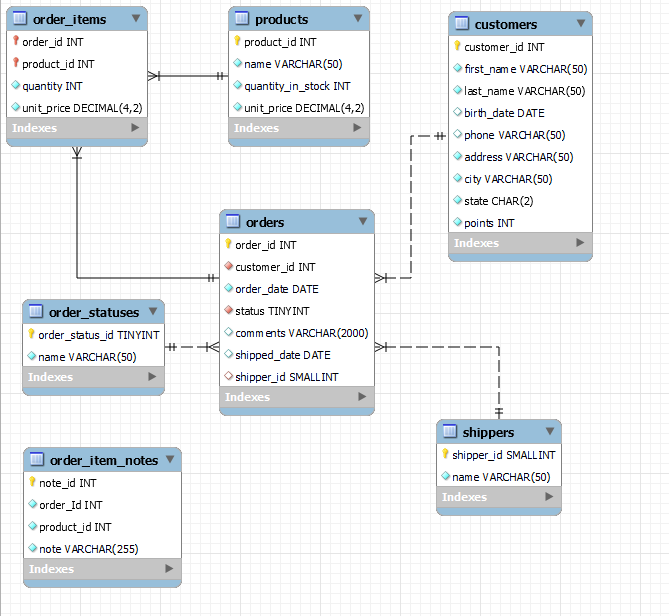
Similar to Task 4, but in this case the most expensive product is returned.

## Task 6



Another single result, this time the **oldest customer** from the **sql\_store** schema is returned.

## EER Diagram

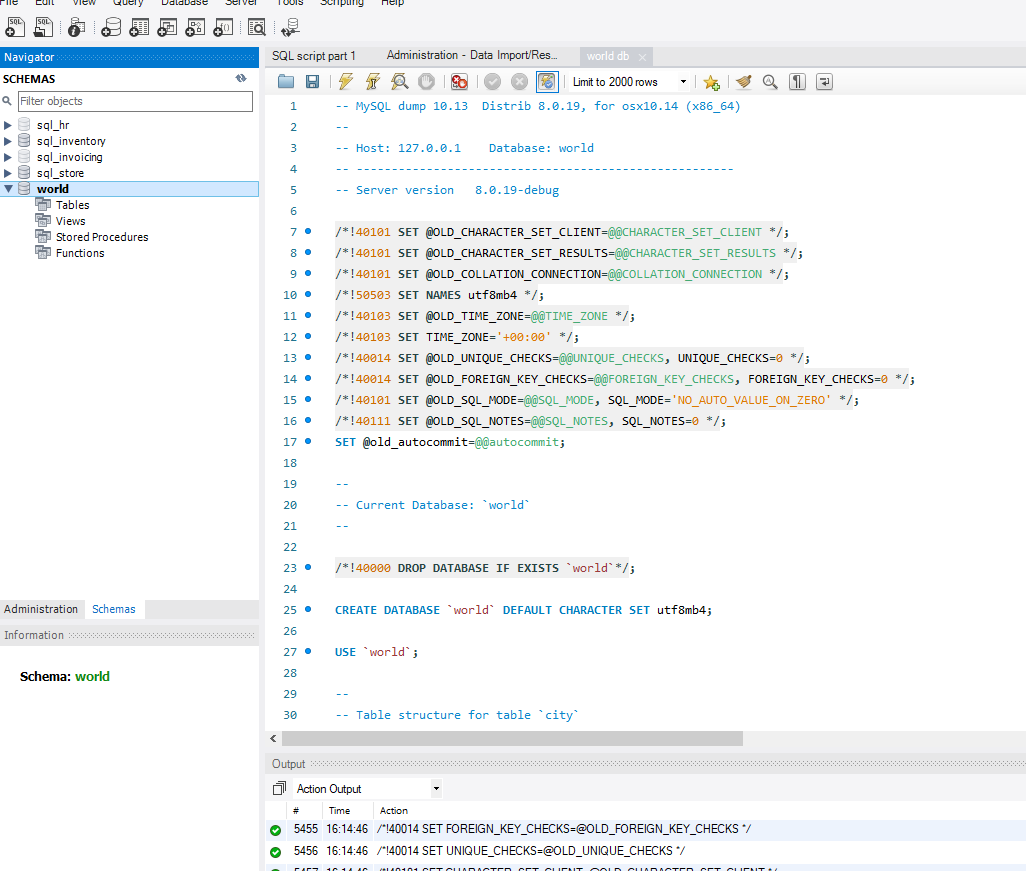


Explaining this EER Diagram:

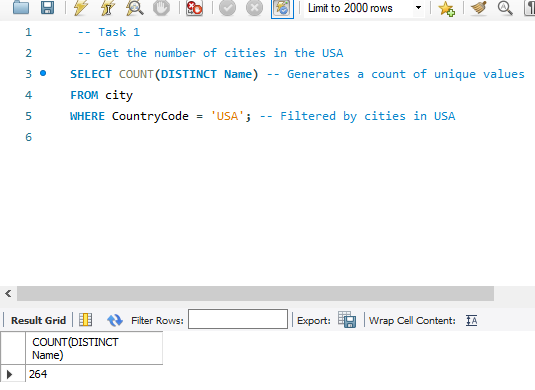
1. I have placed the orders table at the center of the model as it has the most relationships with other tables.
   1. A one or many relationship to order\_items. This is because each order could have one or many ordered items.
   2. A one (and only one) relationship to order\_statuses. This is because each order can only have one status.
   3. A one (and only one) relationship to shippers. This is because each order only has one shipper.
   4. A one (and only one) relationship to customers. Because each order only has one customer.
2. customers, shippers, order\_statuses all have a one or many relationship to orders. This is because:
   1. a customer can place one order, or as many as they want
   2. a shipper could have one or many orders
   3. an order status could apply to one or many orders
3. order\_items has a one (and only one) relationship to products. This is because an order can only have a product once. (Although if a product ordered was more than one quantity, this would need to change).
4. products has a one or many relationship to order\_items. Similar to above.
5. The relationships between orders and order\_statuses, customers and shippers is non-identifying. This means that each table has its own primary key. Therefore order\_items, which doesn’t have a primary key has an identifying relationship.
6. order\_items has two foreign keys, which are order\_id (primary key from orders) and product\_id (primary key from products).

# Part 2 – World db.sql

## Import and execute

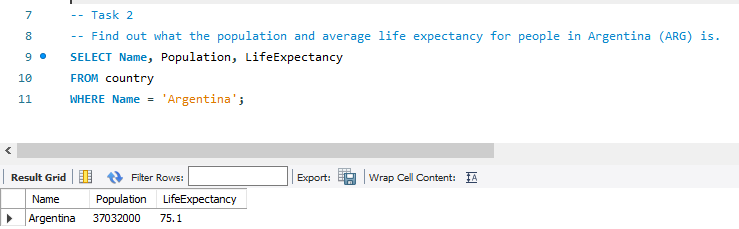


## Task 1



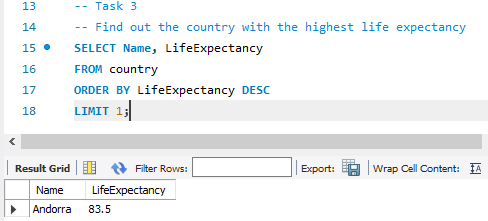
I used COUNT to get the number of cities in the USA. I added DISTINCT to make sure the count only returns unique rows.

## Task 2



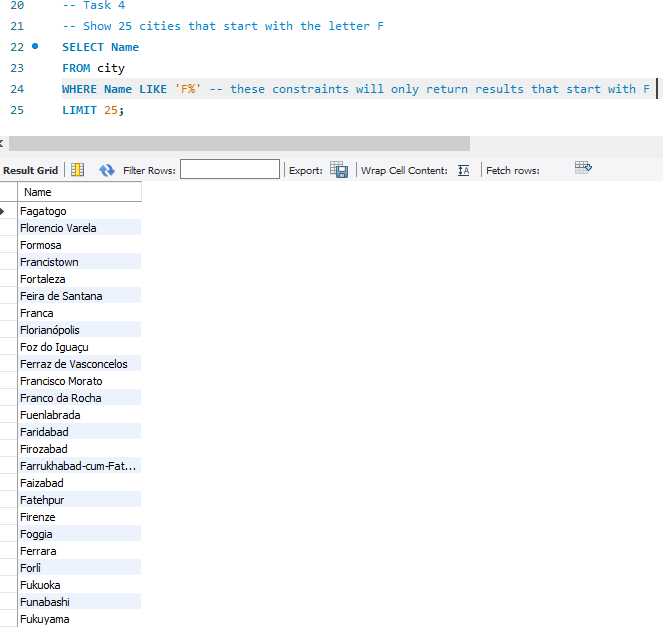
This query displays the population and average life expectancy for Argentina.

## Task 3



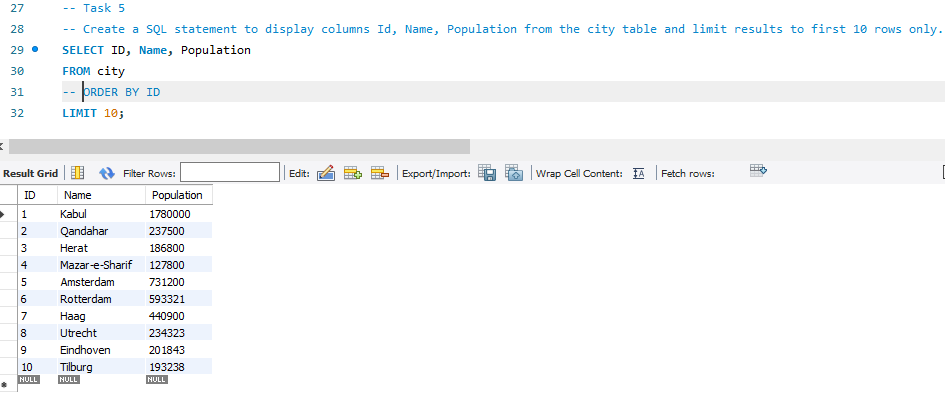
The above query shows that Andorra is the country with the highest life expectancy of 83.5 on average.

## Task 4



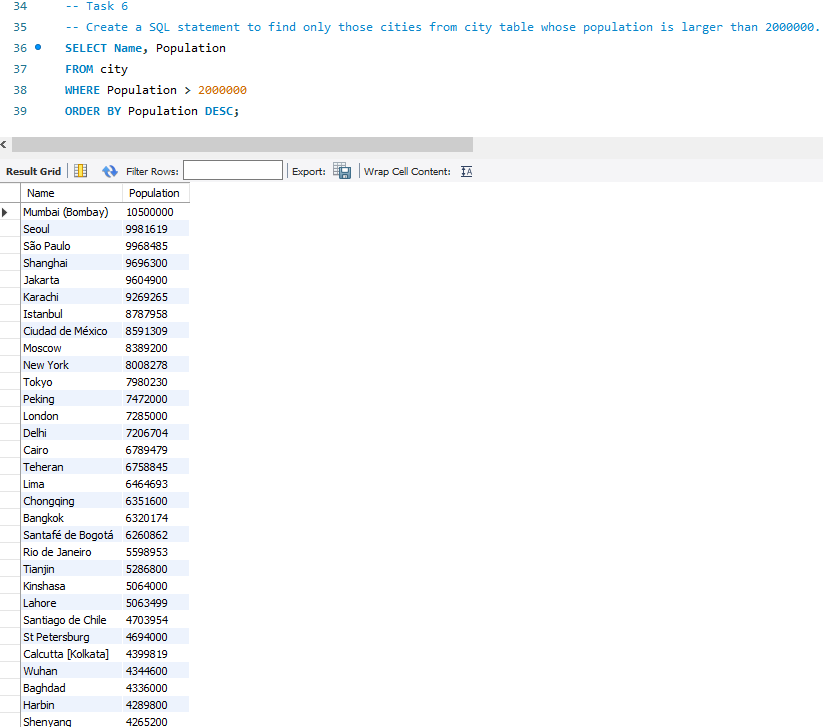
This query displays 25 cities starting with the letter F. If I wanted this to be the first 25 results, I could insert ‘ORDER BY city’ at line 24.

## Task 5



This query returns the first 10 entries in the city table. I noticed that because the **primary key (ID)** is included, that **ORDER BY ID is not required** to return the first 10 rows and makes no difference. I therefore commented it out.

## Task 6



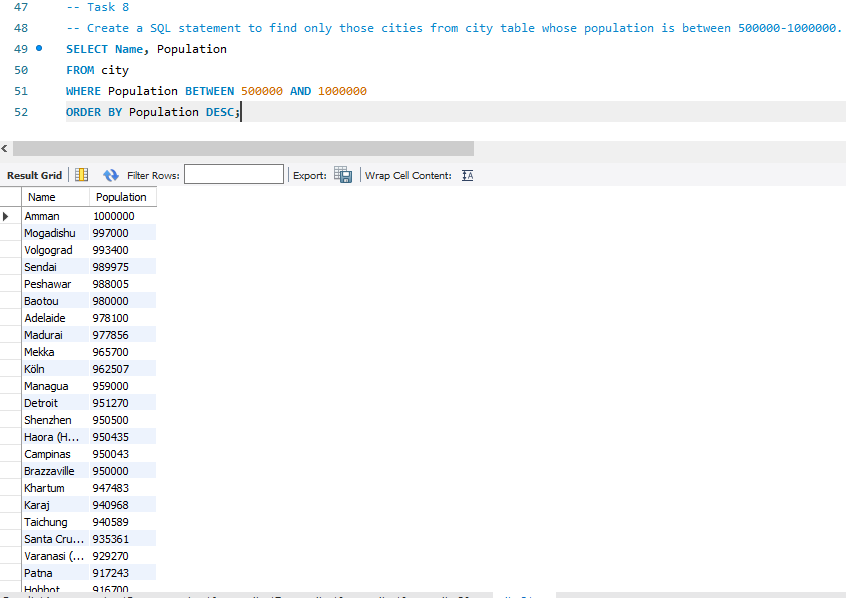
This query displays cities with a population larger than 2 million. I added ORDER BY with DESC to sort the results in descending order.

## Task 7



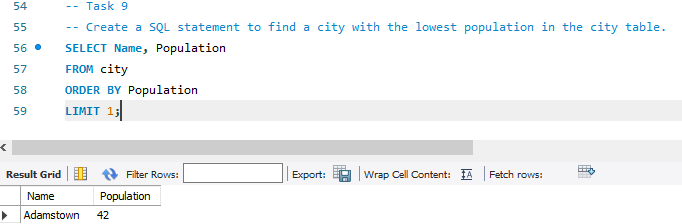
Using the **wildcard %** after the prefix ‘Be’ I was able to get all cities starting with ‘Be’.

## Task 8



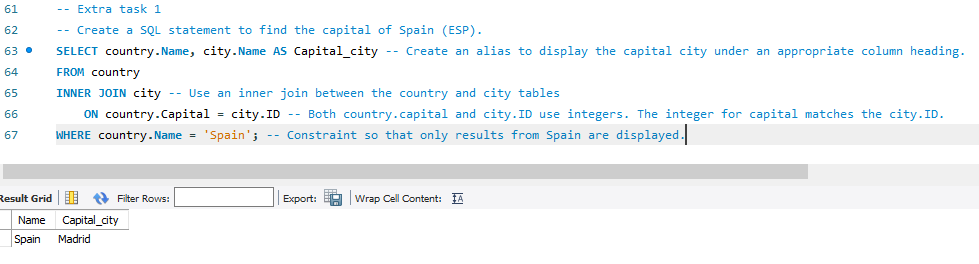
Using the constraint **BETWEEN** the two given numbers displayed the required results.

## Task 9



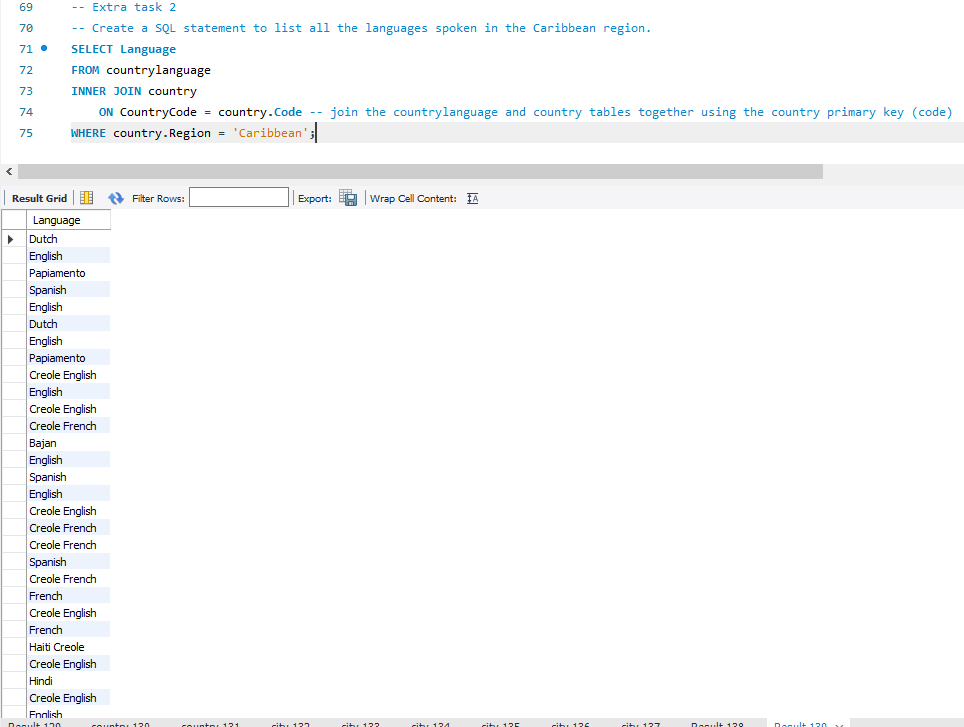
This straightforward query displays the city with the lowest population.

## Extra task 1

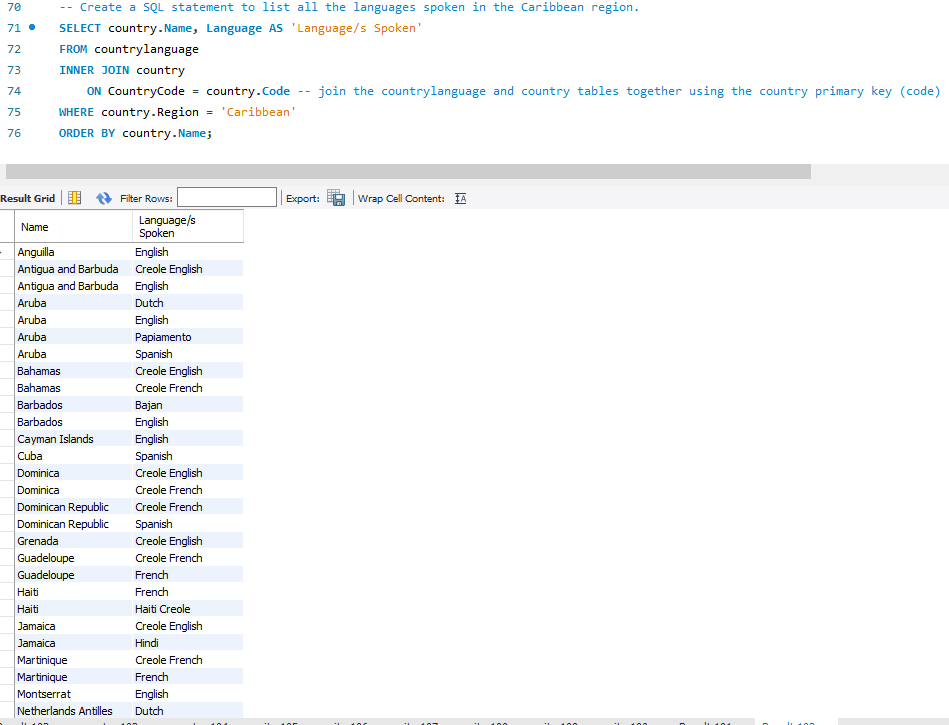


Here an INNER JOIN is used to link the capital column from the country table (an integer of 653) with the corresponding value (integer of 653) in the ID column (Primary Key) in the city table.

## Extra task 2

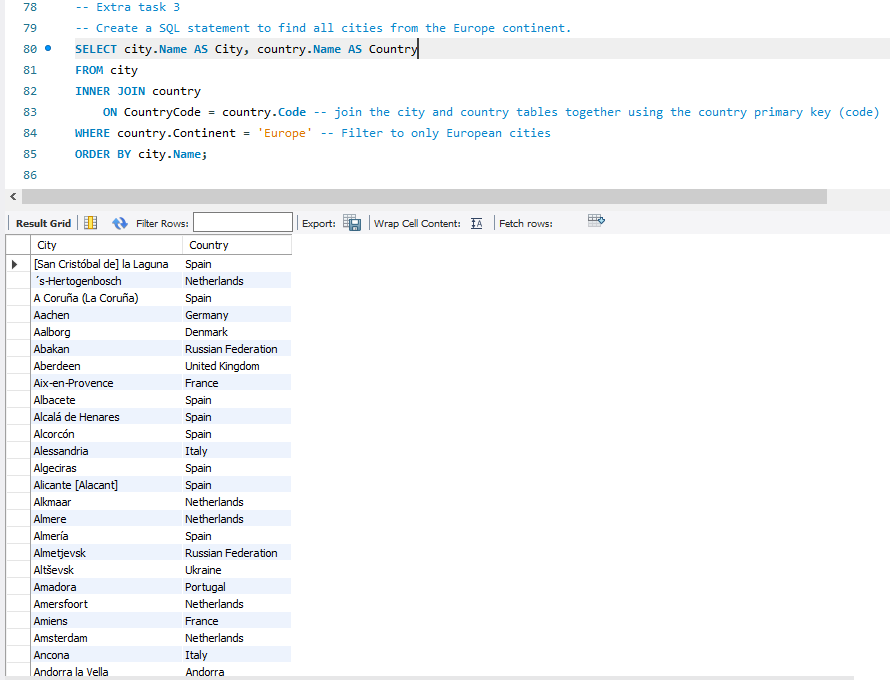


This query uses a join on the primary key for country (code) with countryCode from the country language table. Although the result is correct, it could be improved to make it easier to understand.



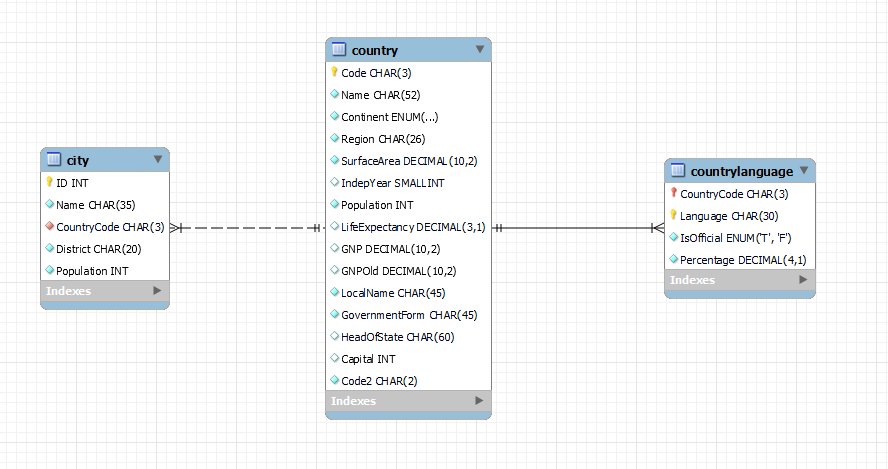
Adding the name of country (and sorting the results by this), as well as using an alias for the languages spoken, makes the result much easier to understand.

## Extra task 3



This displays cities from Europe, using a join and a constraint. I also added aliases to make the column headings meaningful, as they previously would have just said ‘Name’.

## EER Diagram



1. The primary key in the country table is Code.
2. The primary key in the city table is ID.
3. The primary key in the countrylanguage table is Language.
4. The foreign key in the city table is CountryCode.
5. The foreign key in the countrylanguage table is CountryCode.