Databases and SQL Assignment 2

MYSQL AND DBMS

Rayan Arshed

JUST IT | DATA COHORT 1 – DFEW4 |16/10/2023

Contents

Written Tasks

Task 1 – List the different types of relationships in relational databases and provide examples

Relational Databases typically contain multiple relationships, which are links made between data tables that relate their relationships to each other. These relationships are typically linked using Primary and Foreign Keys, where Primary Keys are an attribute that is uniquely identifying for each record within the dataset, and the Foreign Keys are the Primary Keys of other tables, located within another table, allowing for the relationship to be established.

Of these relationships between tables, there are typically 3 forms they can take. Firstly and the most common is a One-to-Many relationship, where the relationship established between two tables takes the form where 1 value in Table A can be related to many values in Table B, but each value in Table B is only linked to a single value in Table A. An example of this could be Customers and Orders, where 1 Customer can purchase and make many Orders in their name, however each recorded Order is only made by a single Customer; there are never 2 Customers per order. This relationship helps establish a hierarchy between tables, which can aid in logging and recording order history by each user, and thus allows for extensive querying of the database to look for specific information related to a single person.

Another form of relationships between tables is a One-to-One relationship. This typically happens when a single record in Table A is linked to a single record in Table B. These typically do not need to be established as separate tables, as if the two tables are linked in this relationship, then an argument can be made to combine the two tables into one, reducing relational complexity which can speed up querying of the database. However, there are specific cases where these relationships are useful, for example, Patients and their Medical Records, as each Patient has a single Medical Record, which technically could be included under the Patient table, however the advantage of Relational Databases is that access to specific tables can be restricted to require certain clearances, so by maintaining the One-to-One link, someone such as a front desk secretary for a hospital could have access to the Patient's contact information, but not be able to access their personal Medical Records for privacy reasons.

Finally, the last form of relationships between tables is a Many-to-Many relationship. This is where a single value in Table A is linked to multiple in Table B, meanwhile a single value in Table B is linked to multiple values in Table A. This relationship is the least common as it causes ambiguity between the relationships, and makes things more complex, so typically it is avoided or worked around by implementing a bridging table. An example of this scenario could be Students and Subjects, where 1 Student can study many Subjects, and each Subject has multiple Students enrolled on the course. Due to the ambiguity, a bridging table can be implemented, say Enrolment, where One student can be in multiple Enrolments, and a Subject can be in many Enrolments, however each Enrolment is linked to a single Subject, turning this Many-to-Many relationship into a One-to-Many relationship on both sides of Enrolment, with the Many relationships being linked to Enrolment, clarifying the data and creating a clearer link between the tables.

Task 2 – What is Normalization and why is it important for database development?

Data Normalization is the process of taking a database and splitting each entity within the database into its own table, while establishing relationships between the tables. This is extremely important as without normalization, every single database would be a Flat File Database, which is a single file that allows for duplicates, and aggregated data that can be harder to interpret. Furthermore, Flat File Databases are much harder to query due to unspecified relationships between fields, and no way of efficiently searching through the data without sorting it first. As such, Normalization is applied to such databases, splitting each identified entity into a separate table, and creating Primary and Foreign keys to create relational links between tables.

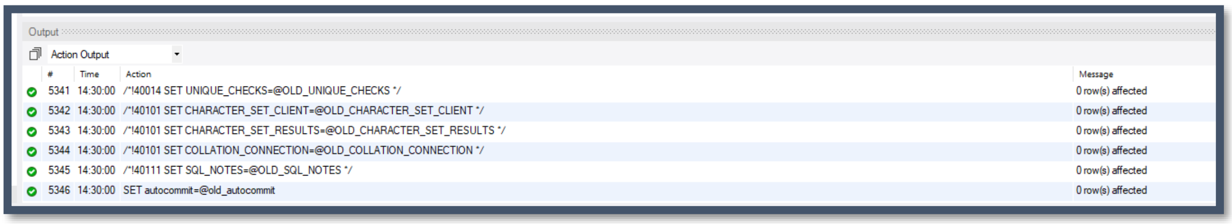
By Normalizing data and creating relationships, querying is much easier to perform, due to the links between data allowing for sorting and searching, and due to the normalized nature of the database, redundant data can be kept at a minimum, saving file space, and allowing for more efficient storage and maintenance of data. For example, updating someone's name in a Flat File Database is a lot more work; you would need to search through the data for every instance of the person's name, and update it manually. Meanwhile, in a relational database that has been normalized, there would only be one entry for someone's name, say a Student's personal details, so only that single record would need to be updated, and the change would be universal and instant across the database.

Furthermore, Normalizing a database allows for the relations to be made between tables, which offers a method to delegate access to specific parts of the data to people with the relevant authority. For example, with the Medical Record example I used in the previous task, this would not be possible with an unnormalized database, but with a normalized database, restrictions can be placed on data access, and clearances can be set for different profiles, aiding in end user security, allowing for data procedures, protections, and laws to be followed. In conclusion, normalization is important for database development as it aids in query systems, efficient storage of data (especially at the largest scale by removing redundant data), and aids in security and access concerns to private or confidential data.

MySQL Tasks

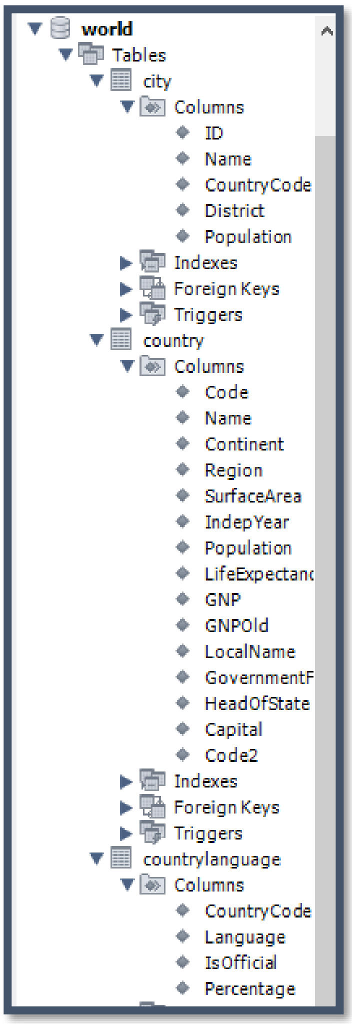
Initial Setup

The provided script for the World database has been run, prompting this output:

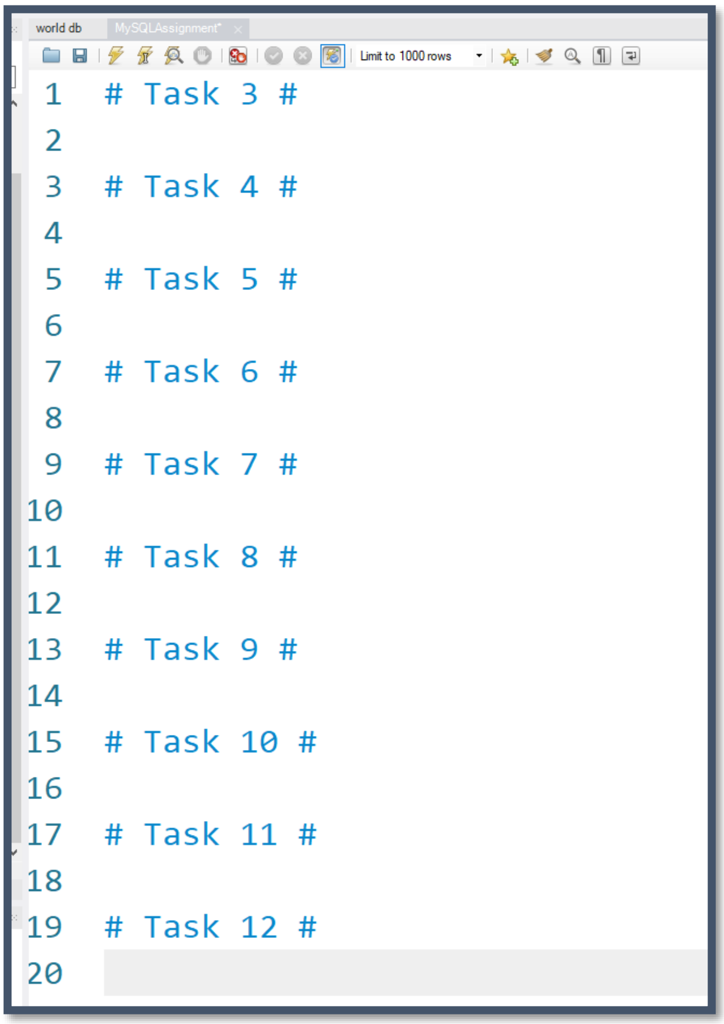


```
Output
Action Output
#     Time      Action                                                         Message
5341  14:30:00  /*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */                0 row(s) affected
5342  14:30:00  /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */  0 row(s) affected
5343  14:30:00  /*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */ 0 row(s) affected
5344  14:30:00  /*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */  0 row(s) affected
5345  14:30:00  /*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */                        0 row(s) affected
5346  14:30:00  SET autocommit=@old_autocommit                                  0 row(s) affected
```

With this done, the schemas sidebar can be checked to ensure that the World database has the correct columns required for the future task, which can be seen here, alongside which a new SQL Query page that has been created, which will be used to document and save the tasks ahead:



```
world
  Tables
    city
      Columns
        ID
        Name
        CountryCode
        District
        Population
      Indexes
      Foreign Keys
      Triggers
    country
      Columns
        Code
        Name
        Continent
        Region
        SurfaceArea
        IndepYear
        Population
        LifeExpectanc
        GNP
        GNPOld
        LocalName
        GovernmentF
        HeadOfState
        Capital
        Code2
      Indexes
      Foreign Keys
      Triggers
    countrylanguage
      Columns
        CountryCode
        Language
        IsOfficial
        Percentage
```

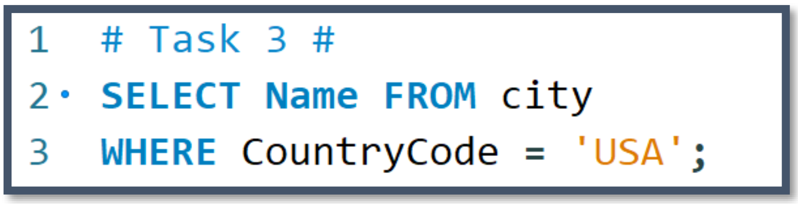


```
# Task 3 #

# Task 4 #

# Task 5 #

# Task 6 #

# Task 7 #

# Task 8 #

# Task 9 #

# Task 10 #

# Task 11 #

# Task 12 #

```

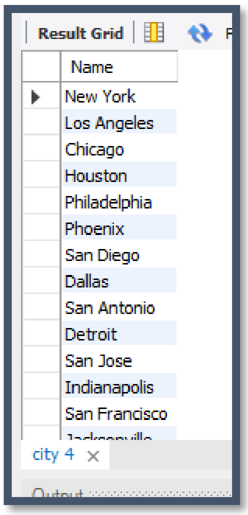
With this setup done, work can officially begin on the tasks ahead.

Task 3 – Using count() to get the Number of Cities in the USA

To begin with, before implementing the COUNT, a simple SQL output for the cities in the USA can be done to verify the query, before adding in the COUNT attribute. This simple query and its output can be seen below:

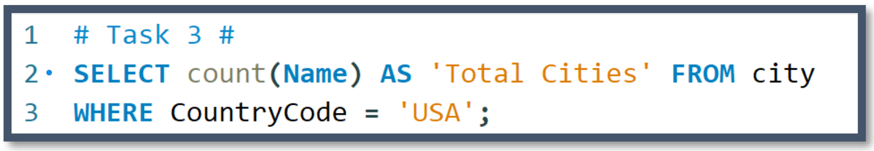


```sql
# Task 3 #
SELECT Name FROM city
WHERE CountryCode = 'USA';
```

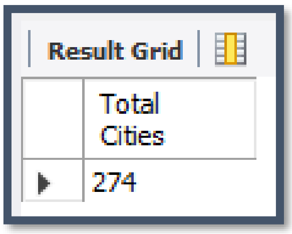


```
Result Grid
Name
New York
Los Angeles
Chicago
Houston
Philadelphia
Phoenix
San Diego
Dallas
San Antonio
Detroit
San Jose
Indianapolis
San Francisco
city 4
```

This outputs a list of cities within the USA, verifying the query works. Then, the count modifier can be used to tally up the total of the cities, and an alias can be given to the value too, which looks like so below:



```sql
# Task 3 #
SELECT count(Name) AS 'Total Cities' FROM city
WHERE CountryCode = 'USA';
```

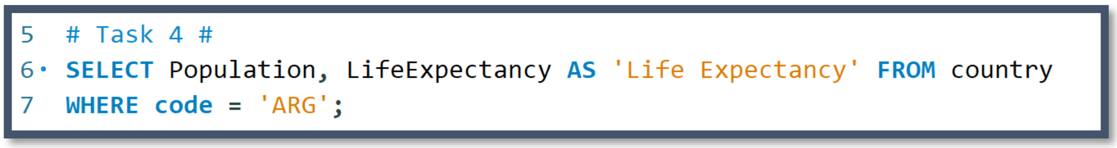


```
Result Grid
Total
Cities
274
```

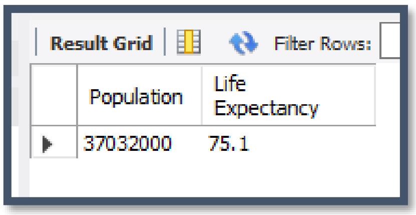
Therefore from this query, it can be gathered that there are 274 recorded cities within the USA.

Task 4 – Finding the Population and Life Expectancy of Argentina

Similar to the previous task, except with two fields to be selected, a simple query can be created to find the population and life expectancy for Argentina:



```sql
# Task 4 #
SELECT Population, LifeExpectancy AS 'Life Expectancy' FROM country
WHERE code = 'ARG';
```

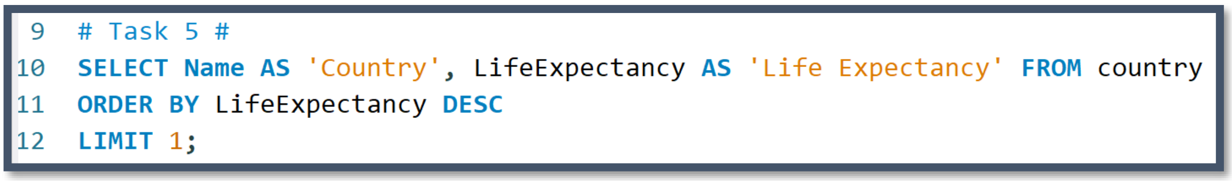


```
Result Grid    Filter Rows:
Population   Life
             Expectancy
37032000     75.1
```

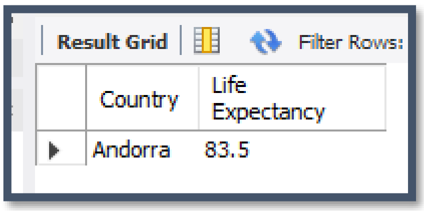
From this, it can be seen that the population of Argentina is 37,032,000 people, and the Life Expectancy of the population is 75.1 Years.

Task 5 – Using ORDER BY, LIMIT, to find the Country with the Highest Life Expectancy

By using ORDER BY and LIMIT, the returned values from the query can be ordered from highest to lowest, and the total returned values can be constrained, allowing for only a single result to be returned. Implementing these into a query looks like so:



```sql
# Task 5 #
SELECT Name AS 'Country', LifeExpectancy AS 'Life Expectancy' FROM country
ORDER BY LifeExpectancy DESC
LIMIT 1;
```

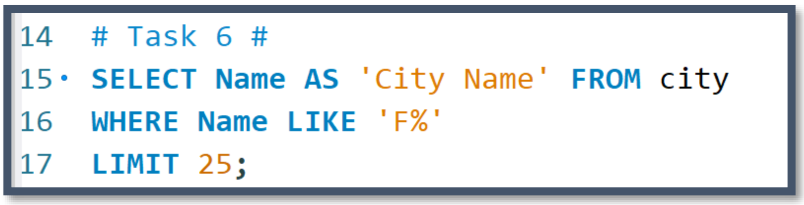


```
Country    Life Expectancy
Andorra    83.5
```

From this result, it can be gathered that Andorra is the country with the highest Life Expectancy, at 83.5 Years.

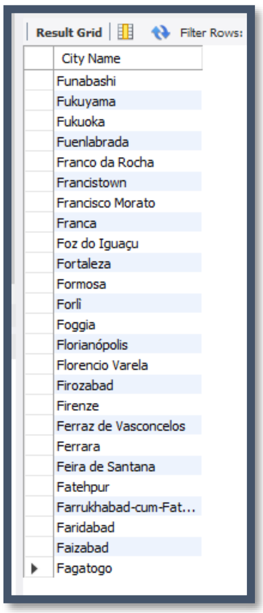
Task 6 – Selecting 25 Cities that start with the letter ‘F’

Similar to the previous query, Limit will be used for this question, to constrain the result to 25 values, and a wildcard value will be used in the Where clause, to find names that start with the letter ‘F’. This query will look like so:



```sql
# Task 6 #
SELECT Name AS 'City Name' FROM city
WHERE Name LIKE 'F%'
LIMIT 25;
```

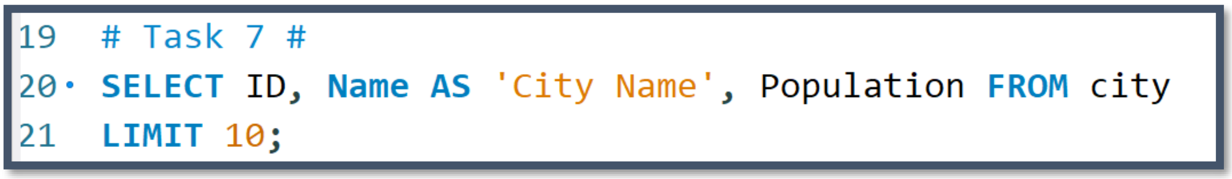
This returns a list of 25 cities that begin with F as seen below:



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

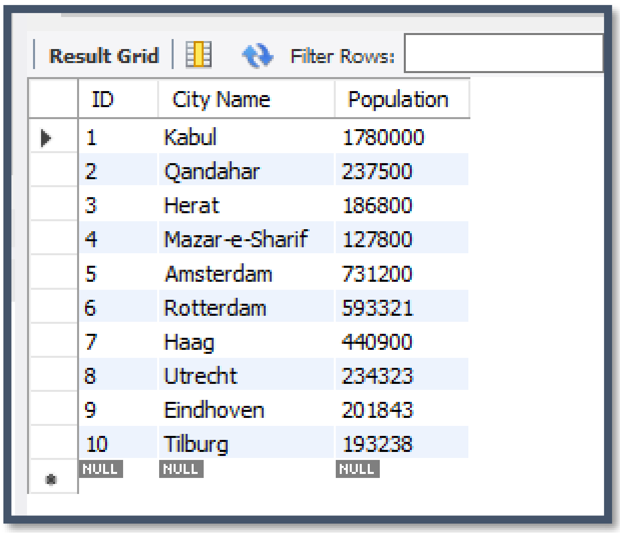
Task 7 – SQL Statement that displays ID, Name, Population, limited to the first 10 rows

Again similar to previous queries, this is a simple query that requires multiple selected values, and a limit of 10 values to be outputted. This query will look like so:



```sql
# Task 7 #
SELECT ID, Name AS 'City Name', Population FROM city
LIMIT 10;
```

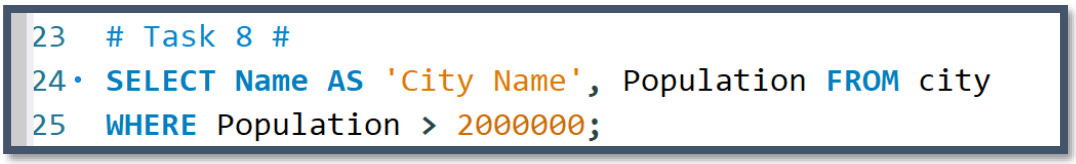
This returns a list of 10 cities as seen below:



ID	City 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

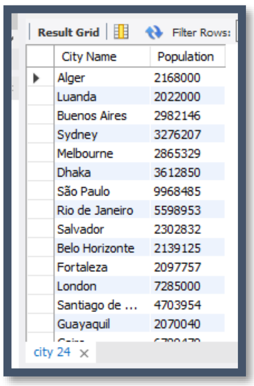
Task 8 – Finding Cities whose population is Larger than 2,000,000

This query will require comparison operators, specifically greater than, which will be used within the Where clause to find the specified values. This query will appear as so below:



```sql
# Task 8 #
SELECT Name AS 'City Name', Population FROM city
WHERE Population > 2000000;
```

This returns a lengthy list as seen below:

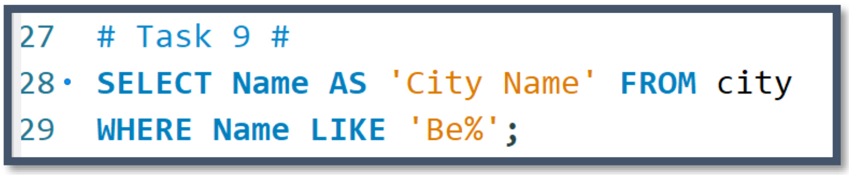


City Name	Population
Alger	2168000
Luanda	2022000
Buenos Aires	2982146
Sydney	3276207
Melbourne	2865329
Dhaka	3612850
São Paulo	9968485
Rio de Janeiro	5598953
Salvador	2302832
Belo Horizonte	2139125
Fortaleza	2097757
London	7285000
Santiago de ...	4703954
Guayaquil	2070040

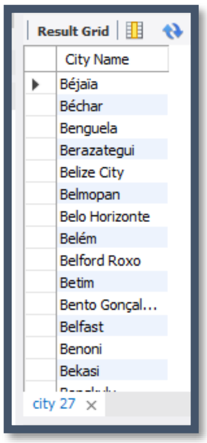
city 24

Task 9 – Finding all City names that begin with 'Be'

Similar to a previous query, this one will require a wildcard in the Where clause to find all cities that begin with Be. This query will be like so, and returns a lengthy list:



```
# Task 9 #
SELECT Name AS 'City Name' FROM city
WHERE Name LIKE 'Be%';
```



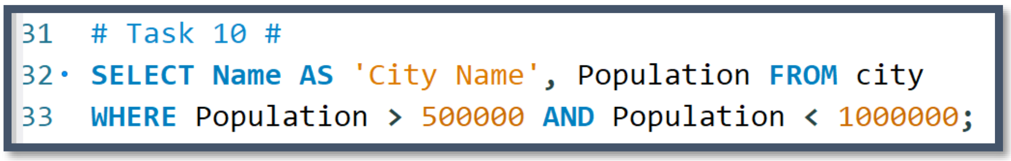
Result Grid

City Name
Béjaïa
Béchar
Benguela
Berazategui
Belize City
Belmopan
Belo Horizonte
Belém
Belford Roxo
Betim
Bento Gonçal...
Belfast
Benoni
Bekasi

city 27

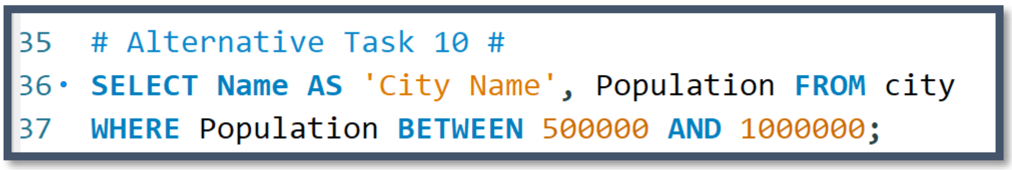
Task 10 – Finding the Cities with a Population Between 500,000 and 1,000,000

Again, similar to previous queries, this will need a comparison operator in the Where clause. The query will appear like so below:



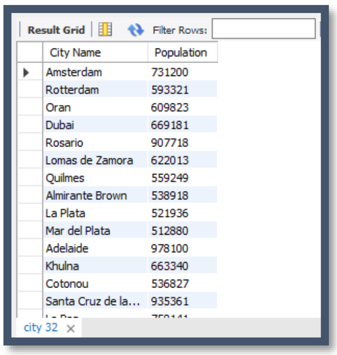
```
# Task 10 #
SELECT Name AS 'City Name', Population FROM city
WHERE Population > 500000 AND Population < 1000000;
```

This query could also be written as:



```
# Alternative Task 10 #
SELECT Name AS 'City Name', Population FROM city
WHERE Population BETWEEN 500000 AND 1000000;
```

Either method would work, I personally prefer to use mathematical signs in my work to prevent any ambiguity in the comparisons, however the Between operator here may make more sense for the request, considering that the inclusion of 500,000 and 1,000,000 has not been specified, so it may be more relevant towards the request, as Between includes the bound in the returned result. Both of these queries result in a lengthy list as seen below:



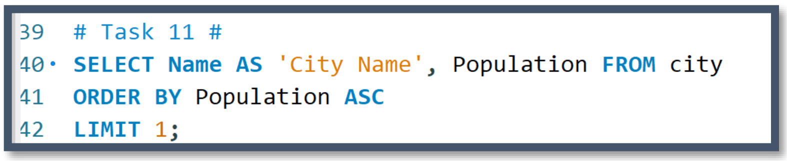
Result Grid Filter Rows:

City Name	Population
Amsterdam	731200
Rotterdam	593321
Oran	609823
Dubai	669181
Rosario	907718
Lomas de Zamora	622013
Quilmes	559249
Almirante Brown	538918
La Plata	521936
Mar del Plata	512880
Adelaide	978100
Khulna	663340
Cotonou	536827
Santa Cruz de la...	935361

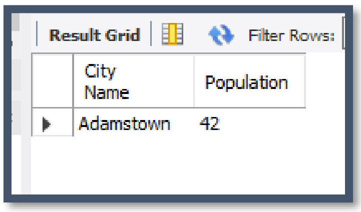
city 32

Task 11 – Finding the City with the Lowest Population

This statement will require an Order By clause, which would allow for ranking the populations from lowest to highest. After doing so, a Limit clause can be used to find the first value returned, which will be the lowest population value in the dataset. This query and outputted result will look like below:



```sql
# Task 11 #
SELECT Name AS 'City Name', Population FROM city
ORDER BY Population ASC
LIMIT 1;
```

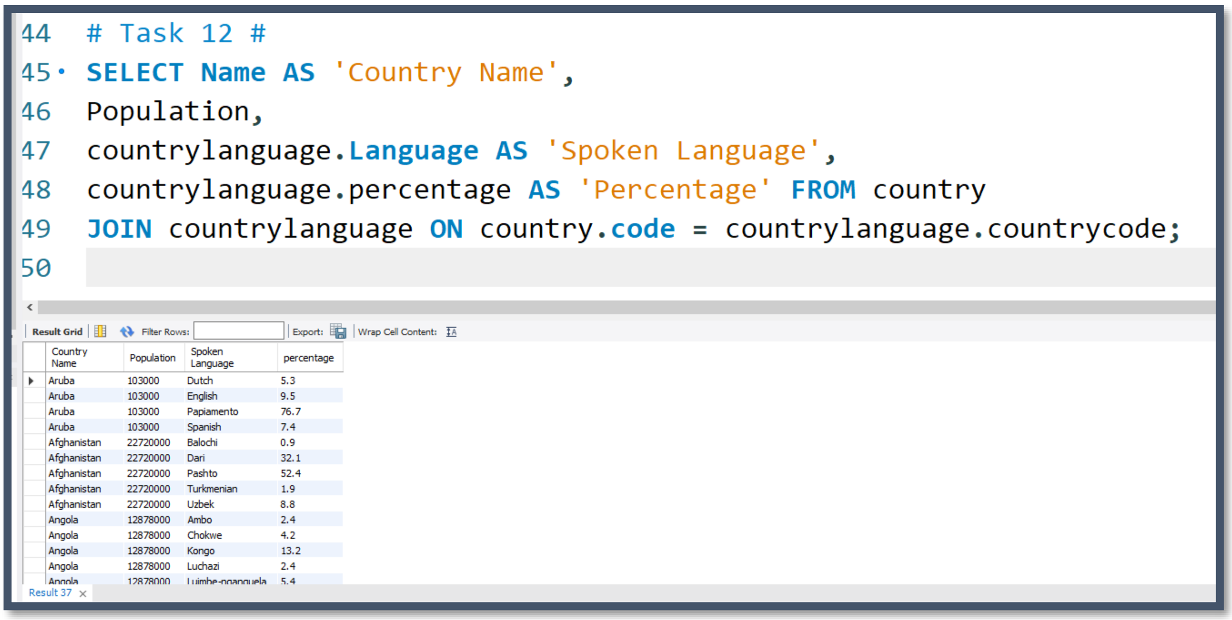


City Name	Population
Adamstown	42

From this it can be gathered that Adamstown in the Pitcairn Islands has the lowest population of 42 people.

Task 12 – Finding the Population of Switzerland and all the Languages spoken there

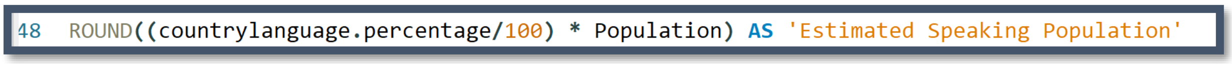
This is a more complex query, which will require a join between the Country and CountryLanguage tables in order to perform. Furthermore, a calculation can be used to find out the estimated population that speaks each unique language within Switzerland, by using the Percentage field and the Population field with some math. With a basic SQL function to check the fields, we can see that Percentage is given as a %, so if the value is divided by 100, and then multiplied against the population, an estimate for the Population that speaks a certain language could be found. The basic Query used to deduce this by checking the available data fields can be seen below:



```sql
# Task 12 #
SELECT Name AS 'Country Name',
Population,
countrylanguage.Language AS 'Spoken Language',
countrylanguage.percentage AS 'Percentage' FROM country
JOIN countrylanguage ON country.code = countrylanguage.countrycode;

```

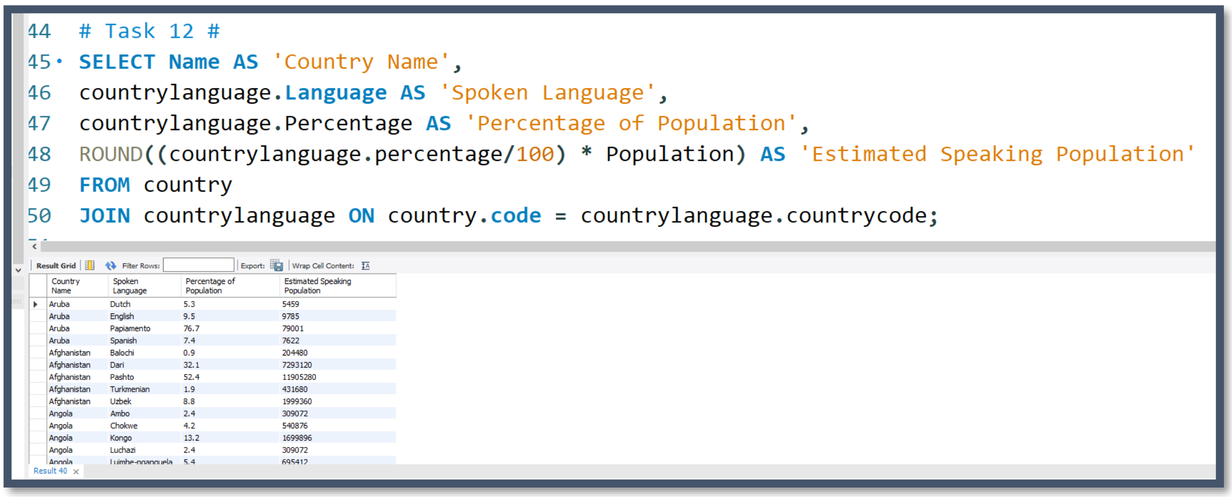
Country Name	Population	Spoken Language	percentage
Aruba	103000	Dutch	5.3
Aruba	103000	English	9.5
Aruba	103000	Papiamento	76.7
Aruba	103000	Spanish	7.4
Afghanistan	22720000	Balochi	0.9
Afghanistan	22720000	Dari	32.1
Afghanistan	22720000	Pashto	52.4
Afghanistan	22720000	Turkmenian	1.9
Afghanistan	22720000	Uzbek	8.8
Angola	12878000	Ambo	2.4
Angola	12878000	Chokwe	4.2
Angola	12878000	Kongo	13.2
Angola	12878000	Luchazi	2.4
Angola	12878000	Luimbe-nganguela	5.4

Now knowing this, a calculation can be implemented as I detailed above. This line of calculation will look like below:



```sql
ROUND((countrylanguage.percentage/100) * Population) AS 'Estimated Speaking Population'
```

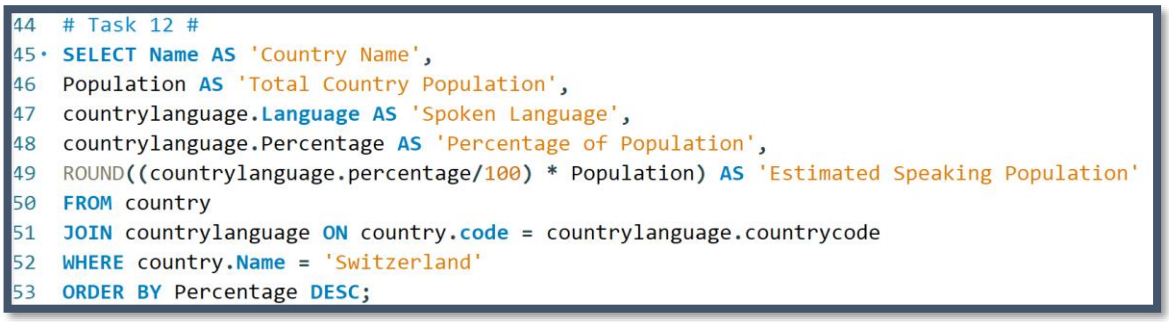
This line divides the percentage value by 100, and then multiplies it by Population, before rounding it to the nearest integer as it is a population value. Implementing this back into the query to test how the results are formatted looks like so:



```
# Task 12 #
SELECT Name AS 'Country Name',
countrylanguage.Language AS 'Spoken Language',
countrylanguage.Percentage AS 'Percentage of Population',
ROUND((countrylanguage.percentage/100) * Population) AS 'Estimated Speaking Population'
FROM country
JOIN countrylanguage ON country.code = countrylanguage.countrycode;
```

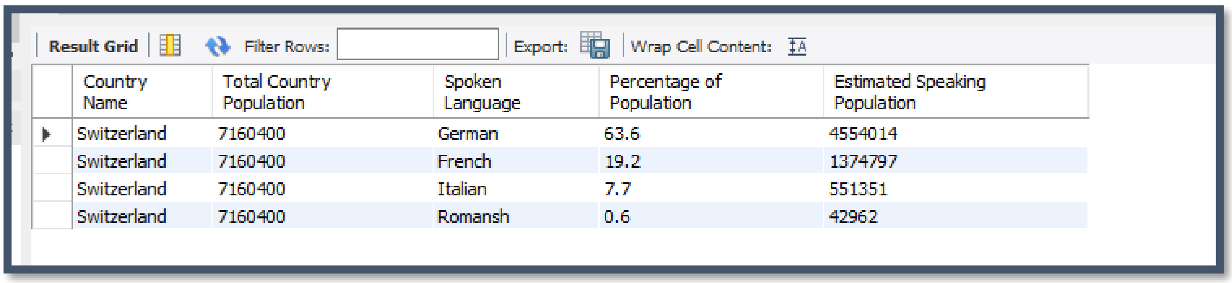
Country Name	Spoken Language	Percentage of Population	Estimated Speaking Population
Aruba	Dutch	5.3	5459
Aruba	English	9.5	9785
Aruba	Papiamento	76.7	79001
Aruba	Spanish	7.4	7622
Afghanistan	Balochi	0.9	204480
Afghanistan	Dari	32.1	7293120
Afghanistan	Pashto	52.4	11905280
Afghanistan	Turkmenian	1.9	431680
Afghanistan	Uzbek	8.8	1999360
Angola	Ambo	2.4	309072
Angola	Chokwe	4.2	540876
Angola	Kongo	13.2	1699896
Angola	Luchazi	2.4	309072
Angola	Luimbe-nganguela	5.4	695412

And now from here, the Where clause and an Order By can be added to filter the results to only include Switzerland and order by most spoken languages. The final detailed query will look like so:



```
# Task 12 #
SELECT Name AS 'Country Name',
Population AS 'Total Country Population',
countrylanguage.Language AS 'Spoken Language',
countrylanguage.Percentage AS 'Percentage of Population',
ROUND((countrylanguage.percentage/100) * Population) AS 'Estimated Speaking Population'
FROM country
JOIN countrylanguage ON country.code = countrylanguage.countrycode
WHERE country.Name = 'Switzerland'
ORDER BY Percentage DESC;
```

This query returns 4 rows, which are the languages spoken in Switzerland, the percentage of the population that speaks those languages, and the estimated population value for each language as seen below:



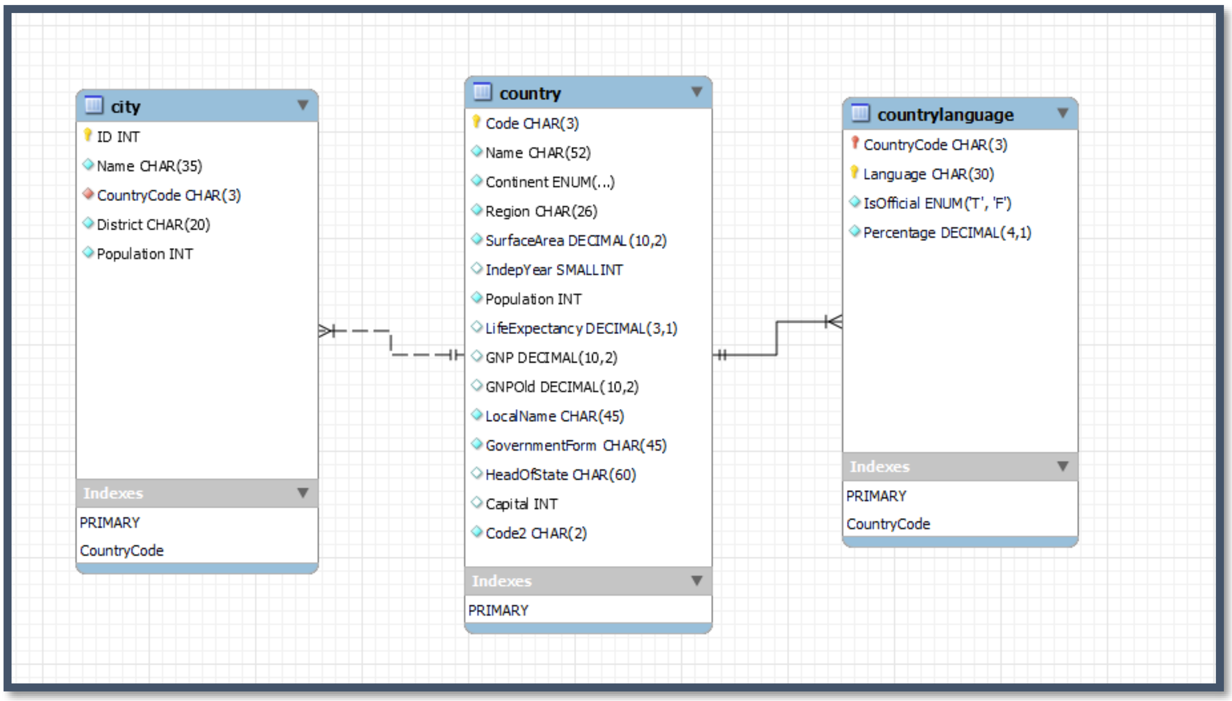
Country Name	Total Country Population	Spoken Language	Percentage of Population	Estimated Speaking Population
Switzerland	7160400	German	63.6	4554014
Switzerland	7160400	French	19.2	1374797
Switzerland	7160400	Italian	7.7	551351
Switzerland	7160400	Romansh	0.6	42962

These results from the query show that German, French, Italian and Romansh are the most frequently spoken languages within Switzerland, and also offers an estimated speaking population for each language. Doing the math, the Percentage values add up to 91.1%, and the Estimated field adds up to 6,523,124 people, which when divided by the Total Population, leads to 91.09%, meaning that about 8.9% of the population is unaccounted for, or about 637,992 people have not been counted in the data. Most likely these people speak a language not listed here, and as there is no Other category for languages, which is what I will assume in this case.

Entity Relationship Diagram Tasks

Task 13 – Creating an Entity Relationship Diagram

Following the steps in the instructions, by using the Reverse Engineer option, an Entity Relationship Diagram has been created for this database, which can be seen below:



The three tables are shown to have relational links to each other, with Country having a One-to-Many relationship to both City and CountryLanguage. This mean that for every country stored in the Country table, there are multiple cities that reference it in the City table, and multiple CountryLanguage that reference it in the CountryLanguage table. However in each of these two tables, each individual record refers to only a single Country.

Task 14 – Use the Entity Relationship Diagram to answer the following:

a. Identify the Primary Key in the Country Table

The Primary Key of the Country Table is the Code attribute. This is the country code that corresponds to each country saved within the table, as it is a unique code per country, and no two countries share or use the same code, meaning that it can be used as a unique identifier. This code was used near the start of the assignment, in Task 3 and 4, where it was used to identify the Country that the Query was to gather data for, such as Cities in the USA, or Population and Life Expectancy for Argentina.

b. Identify the Primary Key in the City Table

The Primary Key of the City Table is the ID attribute, which is a unique ID attributed to every single City registered within the database. It is a unique integer attributed to each city, and could be seen outputted in Task 7, where for example, Kabul was assigned an ID of 1 as it is the first value in the table.

c. Identify the Primary Key in the Country Language Table

CountryLanguage is an interesting table, as there are actually two Keys that make up the Primary Key, being the CountryCode and Language, together creating a Composite Primary Key. This makes sense when considering that multiple countries may speak the same languages, and languages are not unique to a specific country, so by using a Composite Primary Key like this, it prevents a potential Many-to-Many relationship from occurring, enabling unique identification of each language per country.

d. Identify the Foreign Key in the City Table

The Foreign Key in the City Table is the Country Code attribute, which is the Primary Key of the Country table. This is included in the City table in order to create and establish a relationship between the two values, providing the basis for their One-to-Many relationship.

e. Identify the Foreign Key in the Country Language Table

As previously established, the CountryLanguage table uses a Composite Primary Key. Of this Composite Key, the Language was unique to the CountryLanguage table, meanwhile the CountryCode is the Foreign Key used to establish the relational link between this table and the Country table, while also still being a part of the Composite Primary Key as it aids in unique identification alongside the Language attribute.

Reflective

Reflecting on this project, I feel that this has been a great success for me, as not only do I believe that I met the requirements of the SQL tasks, but I feel that I may have gone above and beyond especially when considering Task 12. The question may have asked for a simple statement to show the languages spoken in Switzerland, and the population, but calculating an estimate for each language successfully is evidence of me testing the waters and trying to be creative with my queries, to push my limits and see what I can do with the data and tasks provided to me. I believe that I answered the first two tasks adequately and did what was required of me with the Entity Relationship Diagram section. I feel that my formatting throughout this document has also been consistent, and meets the neatness required for a strong presentation of my work.

In terms of potential improvements, maybe I could have toyed with some other SQL queries and experimented with what I could find, however nothing came to mind as I worked through the tasks. It was only Task 12 that struck a chord in me, making me realise that I could take this further, as most of the other tasks seemed fairly straightforward in their requirements. I did consider potentially making my own Entity Relationship Diagram before generating the ERD from MySQL Workbench, so that I could compare and see the differences and see if I was correct, but as the ERD is also provided in the list of tasks in the presentation, I did not see it worthwhile to try that as I had already seen what the ERD is supposed to look like.

In conclusion, considering my own personal experience and previous work with SQL, as I studied the language both during my education and during my degree, I still believe that I have learnt something from this assignment, as I had no prior experience with using the MySQL software; the majority of my previous SQL experience has been from in-line coding in other languages such as with Python or JavaScript. Now though, I feel familiar enough and confident with MySQL that I feel like most query requests I could solve on my own, and for anything more complex, my fundamental knowledge paired with researching the internet would allow me to answer those requests. As such, I see this assignment as a great success, and it has taught me a lot about SQL, database structure, and querying for results.

SQL Code Archive

My whole SQL code file recorded below for clarity's sake:

```
# Task 3 #

SELECT count(Name) AS 'Total Cities' FROM city

WHERE CountryCode = 'USA';


# Task 4 #

SELECT Population, LifeExpectancy AS 'Life Expectancy' FROM country

WHERE code = 'ARG';


# Task 5 #

SELECT Name AS 'Country', LifeExpectancy AS 'Life Expectancy' FROM country

ORDER BY LifeExpectancy DESC

LIMIT 1;


# Task 6 #

SELECT Name AS 'City Name' FROM city

WHERE Name LIKE 'F%'

LIMIT 25;


# Task 7 #

SELECT ID, Name AS 'City Name', Population FROM city

LIMIT 10;


# Task 8 #

SELECT Name AS 'City Name', Population FROM city

WHERE Population > 2000000;


# Task 9 #

SELECT Name AS 'City Name' FROM city

WHERE Name LIKE 'Be%';


# Task 10 #

SELECT Name AS 'City Name', Population FROM city

WHERE Population > 500000 AND Population < 1000000;
```

```
# Alternative Task 10 #

SELECT Name AS 'City Name', Population FROM city

WHERE Population BETWEEN 500000 AND 1000000;


# Task 11 #

SELECT Name AS 'City Name', Population FROM city

ORDER BY Population ASC

LIMIT 1;


# Task 12 #

SELECT Name AS 'Country Name',

Population AS 'Total Country Population',

countrylanguage.Language AS 'Spoken Language',

countrylanguage.Percentage AS 'Percentage of Population',

ROUND((countrylanguage.percentage/100) * Population) AS 'Estimated Speaking Population'

FROM country

JOIN countrylanguage ON country.code = countrylanguage.countrycode

WHERE country.Name = 'Switzerland'

ORDER BY Percentage DESC;
```