Databases and SQL

Contents

Different types of relationships in a relation database 2

Normalisation..... 4

De-normalisation 4

Database 5

  Import and Execute..... 5

  Saving Task and Query 6

SQL Code 6

  AS 6

  Task 3- COUNT() 6

  Task 4- Population and Life expectancy in Argentina 6

  Task 5- ORDER BY & LIMIT 7

  Task 6- 25 cities around the world starting with ‘F’..... 7

  Task 7- Display columns 8

  Task 8- Cities with more than 2,000,000 population..... 8

Optional Tasks..... 9

  Task 9- City names starting off with ‘Be’ 9

  Task 10- Cities with 500,000- 1,000,000..... 9

  Task 11- City with lowest population..... 10

  Task 12- Population and language spoken in Switzerland..... 10

Extra SQL Queries..... 11

  Official languages above 50% which end with ‘i’..... 11

  Countries in the 'Federal Republic' government form 11

  Count the unique number of continent with a null value in GNPold 12

  ROUND () Average life expectancy in the whole world 12

  Common cities in 'city' and 'countrylanguage' tables which are 80% and above 13

  HAVING function- IndepYears with values which has the LifeExpectancy above 70 on average..... 13

EER Diagram..... 14

  Identifying the primary keys in the country, city and countrylanguage tables 14

  Identifying the foreign keys in the city, country and countrylanguage tables 14

Reflection 15

References 16

Different types of relationships in a relation database

Relational database contains tables with related data stored in rows and columns. Each table has consistent columns and each column is assigned to a specific data type.

There are various relationships in relational databases which defines how data from one table relates to data in another table.

Below are the main relation types of a relational database:

1. One-to-One Relationship- occurs when a row in one table relates to only one row in another table. For example, a person and their passport information.
2. One- to- Many Relationship- most commonly used type in a relational database. This relationship means one record in table1 relates to one or more records in table2. For example, A customer and their Amazon orders.
3. Many- to-Many Relationship- used when many records in table1 are related to many records in table2. For example, many students can enrol in many courses. Unfortunately, Many-to-Many relationships are uncommon in relational databases due to its complexities and challenges. When a Many-to-Many Relationship is identified, a bridging table is created to break the Many-to-Many relationship.
4. Many-to One Relationship- where multiple records from table1 are linked to a single record in table2. For example, in a Student and Class table, many students can be enrolled in a single class and one class can have many students

These relationships help smooth complicated data models that accurately reflect real-world situations within a relational database system. Relationships in databases are vital as they establish connections between various entities, enabling data retrieval, manipulation, and analysis. They safeguard data integrity, consistency, and prevent redundancy. For example, in a customer and amazon_order tables, the relationship assures that each order is linked with a specific customer, so it stops the possibility of wrong and inaccurate data that might appear without this connection.

Although these relations help solve future problems, these relationships can encounter challenges as well. There are three challenges:

1. Data Redundancy- this happens when the same part of data exists in multiple tables. Therefore, this can lead complex data maintenance and inconsistencies. Additional, this can lead to unreliable and meaningless data.
2. Update Anomalies- this occurs when updating data in a database, however, unintentionally, this update affects numerous rows or columns. Moreover, inconsistent updates on related tables can lead data discrepancies.
3. Deletion Anomalies- when data from one table is removed, it might unintentionally remove data which is linked from other tables.

However, the challenges can be overcome by the following three:

1. Normalisation- this process breaks down data into multiple tables and arranges them efficiently. This process helps reduces redundancy.
2. Foreign Key Constraints- implementing these constraints ensures data integrity and enforces referential integrity.
3. Use of Joins: utilising SQL joins properly smooths efficient data retrieval from multiple related tables.

Data redundancy and anomalies can be minimised by executing these strategies. This ensures data accuracy, integrity, and consistency within relational databases.

Normalisation

Normalisation is a process of organising data in a database by diving large tables into smaller, more manageable ones and establishing connections between them. The main goal includes reducing data redundancy, upholding data integrity, and enhancing data consistency. By doing so, normalisation supports in the elimination of anomalies like update, insert, and delete inconsistencies that can compromise the database's efficiency and integrity.

Normalisation is important to database development and the reasons are:

1. Data Integrity- it supports data accuracy and consistency by reducing duplications and irregularities. This results in lowering the risks of data corruption or inconsistencies.
2. Space Efficiency- by cutting data redundancy, normalisation promotes the development of more compact and efficient databases which leads to significant storage savings.
3. Update Anomalies Prevention- it ensures updates to data are implemented regularly and consistently across the database which prevents inconsistencies that could arise if the data were duplicated or spread across multiple tables.
4. Simplified Data Maintenance- structuring data in a normalised manner simplifies data maintenance by structuring data in a way that makes it easier to update, insert, and delete records without affecting the entire database.
5. Increased Query Performance: Well-normalized databases often demonstrate improved query performance, as they allow faster retrieval of data and more efficient execution of queries.

To summarise, normalisation plays a fundamental role in ensuring that databases are well-structured, efficient, and reliable. This is critical for managing and processing data in various applications and systems.

De-normalisation

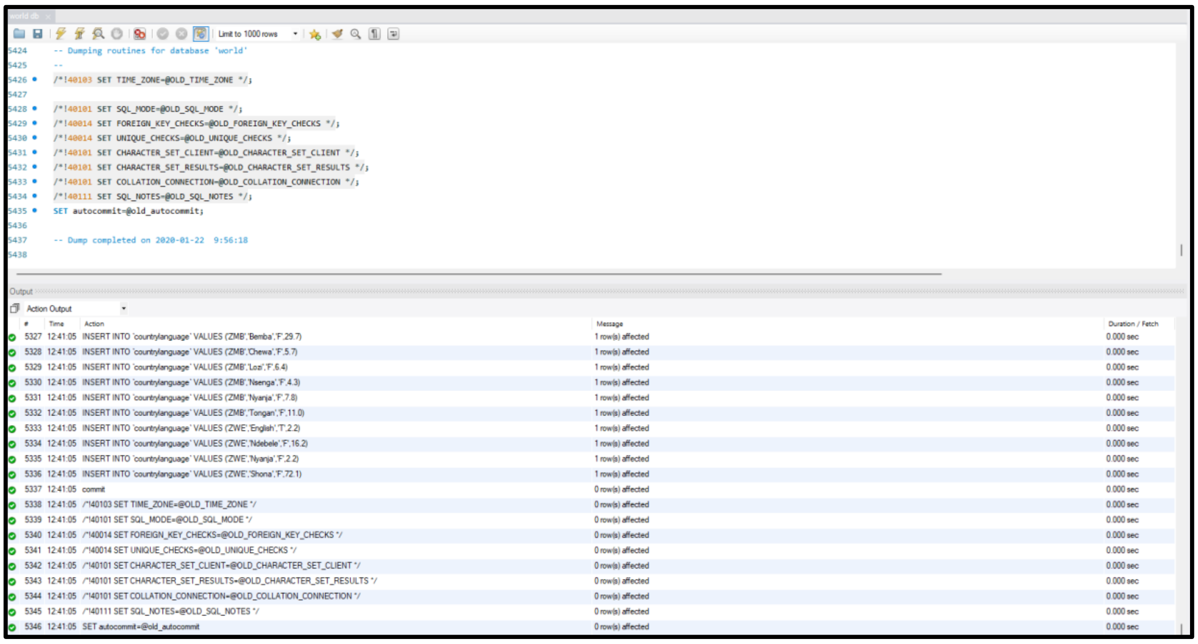
De-normalisation is the process of introducing redundancy into previously normalised databases with the goal of optimising database query performance. It intentionally introduces redundancy to a normalised database using different techniques such as table splitting, adding derived and redundant columns and mirrored tables to solve issues in normalised data. It is commonly applied in scenarios where improved query speed and expedited data retrieval are necessary. It can improve performance however, it can also complicate data maintenance and potentially result in inconsistencies if it is not managed accurately.

While de-normalisation can enhance performance, it is vital to strike the right balance between the advantages and potential disadvantage which includes increased storage demands, data redundancy, and potential inconsistencies.

Database

Import and Execute

The World db file was opened on MySQL Workbench. Unfortunately, the code was not executing as the server was not connected. Following research, the server was re-connected by selecting 'Query' and then 'Reconnect to Server'. Subsequently, the imported code was highlighted and executed using the symbol. The presence of successful ticks in the output indicated the successful execution of the code.



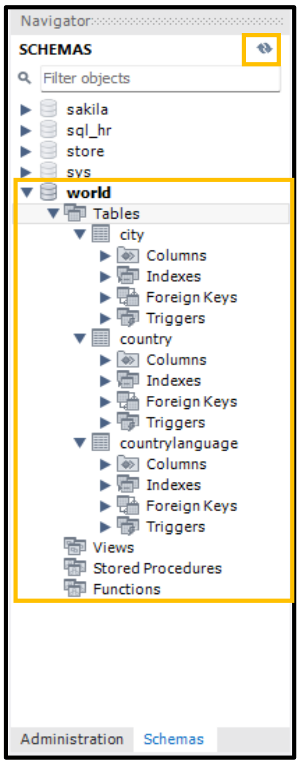
```
-- Dumping routines for database 'world'
--
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
SET autocommit=@old_autocommit;

-- Dump completed on 2020-01-22  9:56:18
```

```
#     Time      Action                                                                Message              Duration / Fetch
5327  12:41:05  INSERT INTO `countrylanguage` VALUES ('ZMB','Bemba','F',29.7)         1 row(s) affected    0.000 sec
5328  12:41:05  INSERT INTO `countrylanguage` VALUES ('ZMB','Chewa','F',5.7)          1 row(s) affected    0.000 sec
5329  12:41:05  INSERT INTO `countrylanguage` VALUES ('ZMB','Lozi','F',6.4)           1 row(s) affected    0.000 sec
5330  12:41:05  INSERT INTO `countrylanguage` VALUES ('ZMB','Nsenga','F',4.3)         1 row(s) affected    0.000 sec
5331  12:41:05  INSERT INTO `countrylanguage` VALUES ('ZMB','Nyanja','F',7.8)         1 row(s) affected    0.000 sec
5332  12:41:05  INSERT INTO `countrylanguage` VALUES ('ZMB','Tongan','F',11.0)        1 row(s) affected    0.000 sec
5333  12:41:05  INSERT INTO `countrylanguage` VALUES ('ZWE','English','T',2.2)        1 row(s) affected    0.000 sec
5334  12:41:05  INSERT INTO `countrylanguage` VALUES ('ZWE','Ndebele','F',16.2)       1 row(s) affected    0.000 sec
5335  12:41:05  INSERT INTO `countrylanguage` VALUES ('ZWE','Nyanja','F',2.2)         1 row(s) affected    0.000 sec
5336  12:41:05  INSERT INTO `countrylanguage` VALUES ('ZWE','Shona','F',72.1)         1 row(s) affected    0.000 sec
5337  12:41:05  commit                                                                0 row(s) affected    0.000 sec
5338  12:41:05  /*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */                              0 row(s) affected    0.000 sec
5339  12:41:05  /*!40101 SET SQL_MODE=@OLD_SQL_MODE */                                0 row(s) affected    0.000 sec
5340  12:41:05  /*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */            0 row(s) affected    0.000 sec
5341  12:41:05  /*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */                      0 row(s) affected    0.000 sec
5342  12:41:05  /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */        0 row(s) affected    0.000 sec
5343  12:41:05  /*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */      0 row(s) affected    0.000 sec
5344  12:41:05  /*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */        0 row(s) affected    0.000 sec
5345  12:41:05  /*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */                              0 row(s) affected    0.000 sec
5346  12:41:05  SET autocommit=@old_autocommit                                        0 row(s) affected    0.000 sec
```

After that, on the left side, the ‘Schemas’ in the ‘Navigator’ tab was refreshed to check the existence of the World DB.



Saving Task and Query

To generate a new query tab, the keyboard shortcuts 'Ctrl' + 'T' was used. Subsequently, the new query was saved by executing the key combination 'Ctrl', 'Shift', and 'S'. The query was then saved and named as 'MySQL Project.

SQL Code

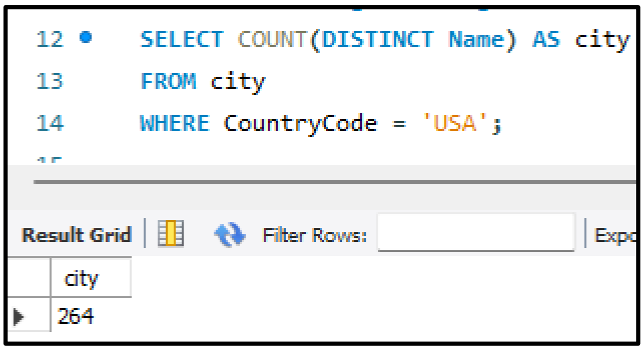
AS

AS is a keyword used for aliasing. It is used to temporarily rename a table or a column in our result set. By using aliasing, it does not change the actual dataset's table or column names, it only applies to the results of the query. AS is used for clarity or brevity and can be very helpful.

Task 3- COUNT()

The COUNT() function is an aggregate function that is used to counts the number of records that match a condition of the query and COUNT(DISTINCT) function is used to count the number of distinct(unique) and non-null values in a dataset.

The code below is counting the unique names where country code is USA, from the city table.



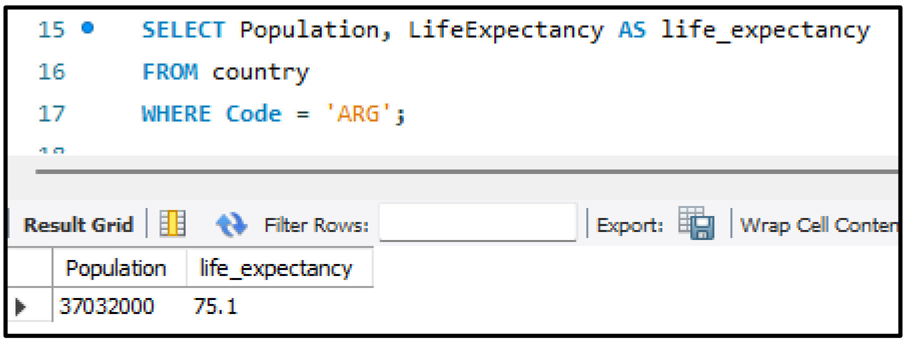
```sql
SELECT COUNT(DISTINCT Name) AS city
FROM city
WHERE CountryCode = 'USA';
```

```
city
264
```

As a result, the total number of cities in the USA is 264.

Task 4- Population and Life expectancy in Argentina

Argentina's population is 37,032,000 and the life expectancy is 75.1. These results were retrieved by selecting the population and life expectancy data from the country table, with a specific condition added using a WHERE clause to filter entries with the country code 'ARG'.



```sql
SELECT Population, LifeExpectancy AS life_expectancy
FROM country
WHERE Code = 'ARG';
```

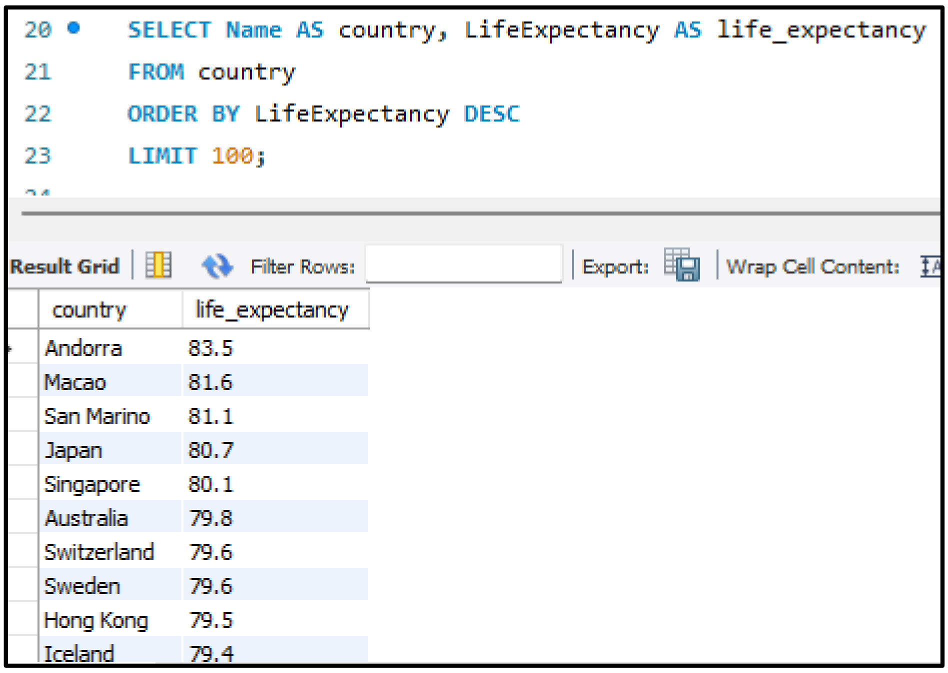
```
Population   life_expectancy
37032000     75.1
```

Task 5- ORDER BY & LIMIT

The 'ORDER BY' clause is used to sort the results set of a query of one or more fields. It allows the results to be arranged in either ascending or descending order.

The LIMIT function is used to constrain the number of records to output.

The code below retrieves the 'Name' and 'LifeExpectancy' columns from the 'country' table and using the keyword DESC, the results are arranged in descending order based on the 'LifeExpectancy' column. The output is then restricted to the initial 100 results.



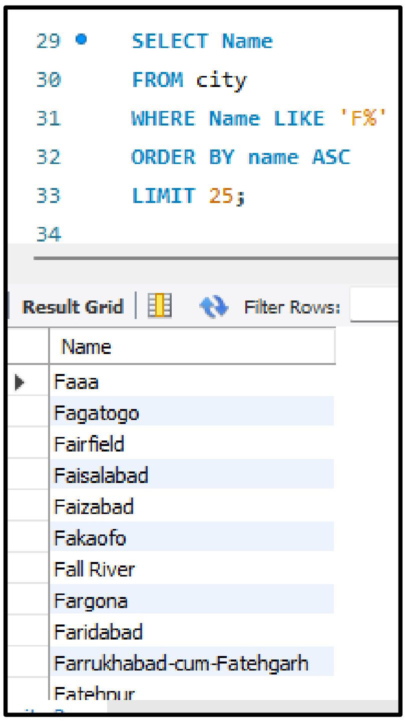
```sql
SELECT Name AS country, LifeExpectancy AS life_expectancy
FROM country
ORDER BY LifeExpectancy DESC
LIMIT 100;
```

country	life_expectancy
Andorra	83.5
Macao	81.6
San Marino	81.1
Japan	80.7
Singapore	80.1
Australia	79.8
Switzerland	79.6
Sweden	79.6
Hong Kong	79.5
Iceland	79.4

As shown on the image, the country with the highest life expectancy is Andorra. Andorra's life expectancy is 83.5

Task 6- 25 cities around the world starting with 'F'

To retrieve a list of 25 cities starting with the letter 'F', the 'Name' column from the 'city' table was selected and a WHERE clause with the LIKE keyword was used. The '%' symbol was utilized to match any sequence of characters. Since the aim was to find cities beginning with 'F', the '%' symbol was positioned after 'F'. Conversely, if the objective had been to fetch cities ending with 'F', the '%' symbol would need to be placed before 'F', as in '%F'.

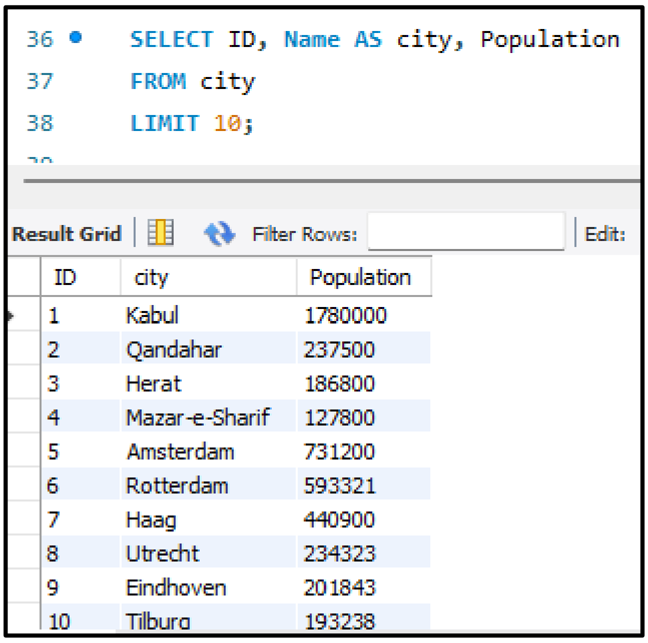


```sql
SELECT Name
FROM city
WHERE Name LIKE 'F%'
ORDER BY name ASC
LIMIT 25;
```

Name
Faaa
Fagatogo
Fairfield
Faisalabad
Faizabad
Fakaofo
Fall River
Fargona
Faridabad
Farrukhabad-cum-Fatehgarh
Fatehpur

Task 7- Display columns

In the following snippet, the ‘ID’, ‘Name’ and ‘Population’ columns were selected from the ‘city’ table. Additionally, the query is restricted to display only the initial 10 results.

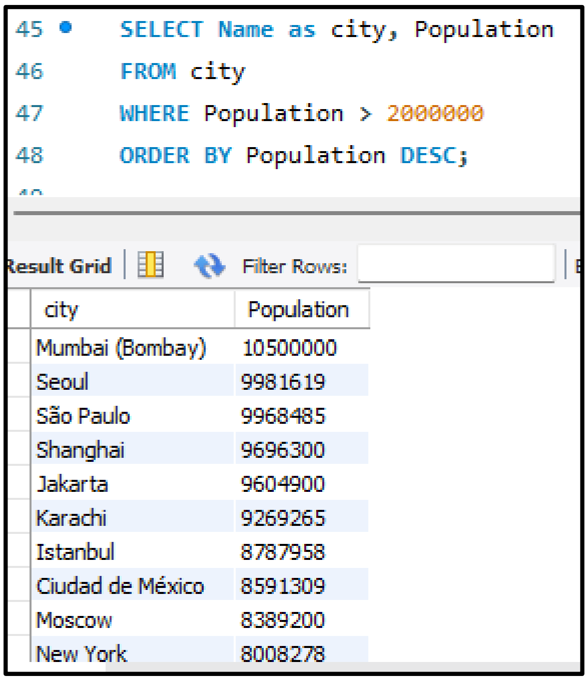


```sql
SELECT ID, Name AS city, Population
FROM city
LIMIT 10;
```

ID	city	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

Task 8- Cities with more than 2,000,000 population

The query below retrieves the ‘name’ and ‘population’ from the city table, filtering cities with population exceeding 2,000,000. This is achieved using a WHERE clause with the comparison operation ‘>’ to filter the desired population range. Furthermore, the results are ordered in descending order based on the ‘population’ column using the ORBER BY function



```sql
SELECT Name as city, Population
FROM city
WHERE Population > 2000000
ORDER BY Population DESC;
```

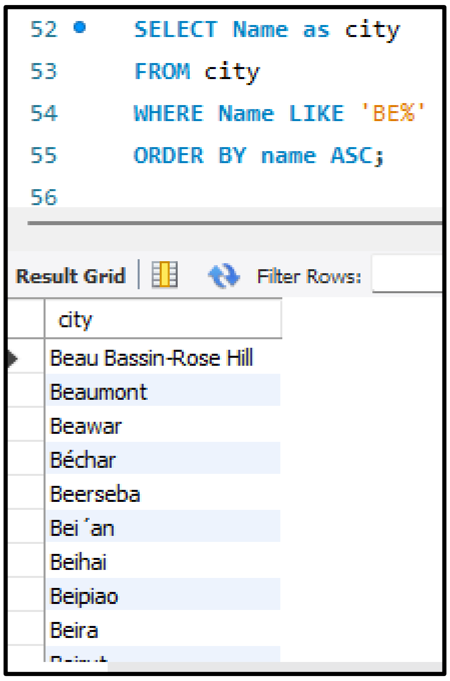
city	Population
Mumbai (Bombay)	10500000
Seoul	9981619
São Paulo	9968485
Shanghai	9696300
Jakarta	9604900
Karachi	9269265
Istanbul	8787958
Ciudad de México	8591309
Moscow	8389200
New York	8008278

As shown in the image, Mumbai has the highest population, totalling 10,500,000.

Optional Tasks

Task 9- City names starting off with ‘Be’

To extract city names commencing with 'Be', the ‘Name’ column from the city table was selected and a WHERE clause along with the keyword LIKE was used to search for a pattern in a field and the icon ‘%’ was also used to match characters. Considering the aim was to retrieve city names starting with 'Be', the '%' symbol was positioned after 'Be', as in '%Be'.



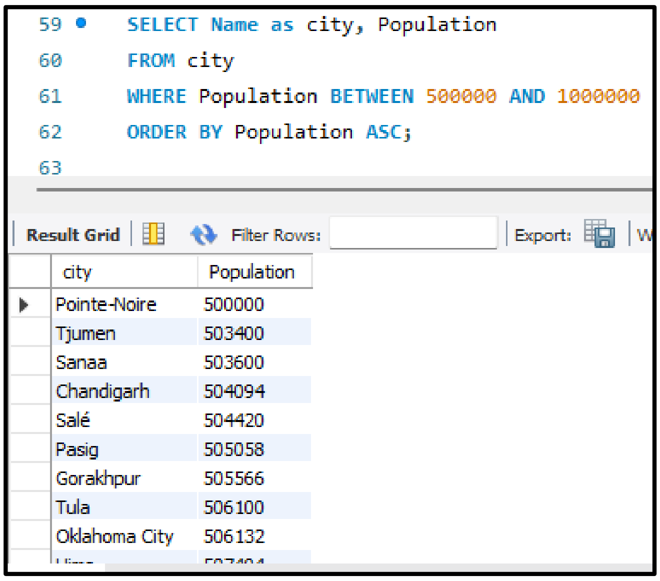
```sql
SELECT Name as city
FROM city
WHERE Name LIKE 'BE%'
ORDER BY name ASC;
```

city
Beau Bassin-Rose Hill
Beaumont
Beawar
Béchar
Beerseba
Bei´an
Beihai
Beipiao
Beira

In this picture, the results show cities that start with ‘Be’. The results have been ordered ascendingly.

Task 10- Cities with 500,000- 1,000,000

The ‘name’ and ‘population’ columns was selected in the city table and a WHERE clause with the BETWEEN and AND keyboards was used to meet the specified criteria. Furthermore, the ORDER BY function was utilised to arrange the data in ascending order based on the ‘Population’ attribute.

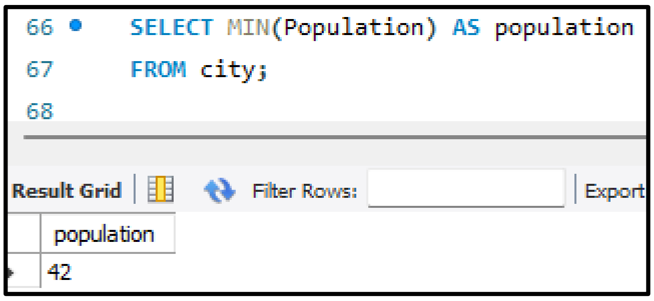


```sql
SELECT Name as city, Population
FROM city
WHERE Population BETWEEN 500000 AND 1000000
ORDER BY Population ASC;
```

city	Population
Pointe-Noire	500000
Tjumen	503400
Sanaa	503600
Chandigarh	504094
Salé	504420
Pasig	505058
Gorakhpur	505566
Tula	506100
Oklahoma City	506132

Task 11- City with lowest population

In order to identify the city with the smallest population, an aggregate function, MIN() was used. This function was used to retrieve the lowest population value from the city table.



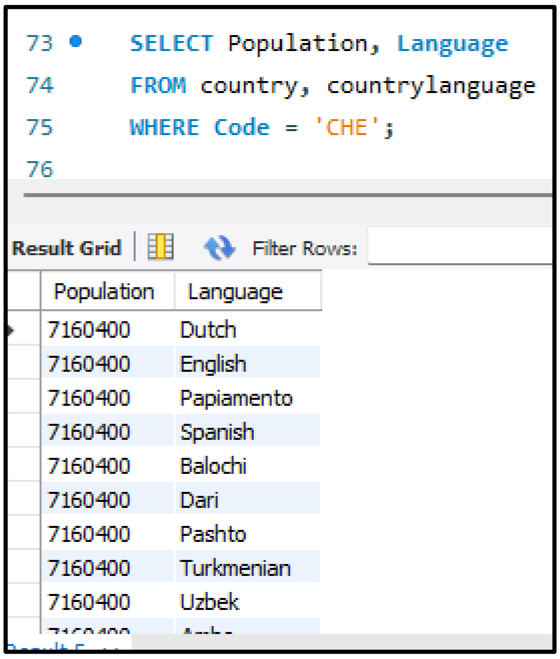
```
SELECT MIN(Population) AS population
FROM city;
```

Result Grid | Filter Rows: | Export

```
population
42
```

Task 12- Population and language spoken in Switzerland

The snippet below provides the population and languagues which are spoken in Switzerland. The 'population' column from the 'country' table and the 'language' column from the 'countrylanguage' were selected. A WHERE clause was used to set the 'Code' to 'CHE' to filter the data for Switzerland.



```
SELECT Population, Language
FROM country, countrylanguage
WHERE Code = 'CHE';
```

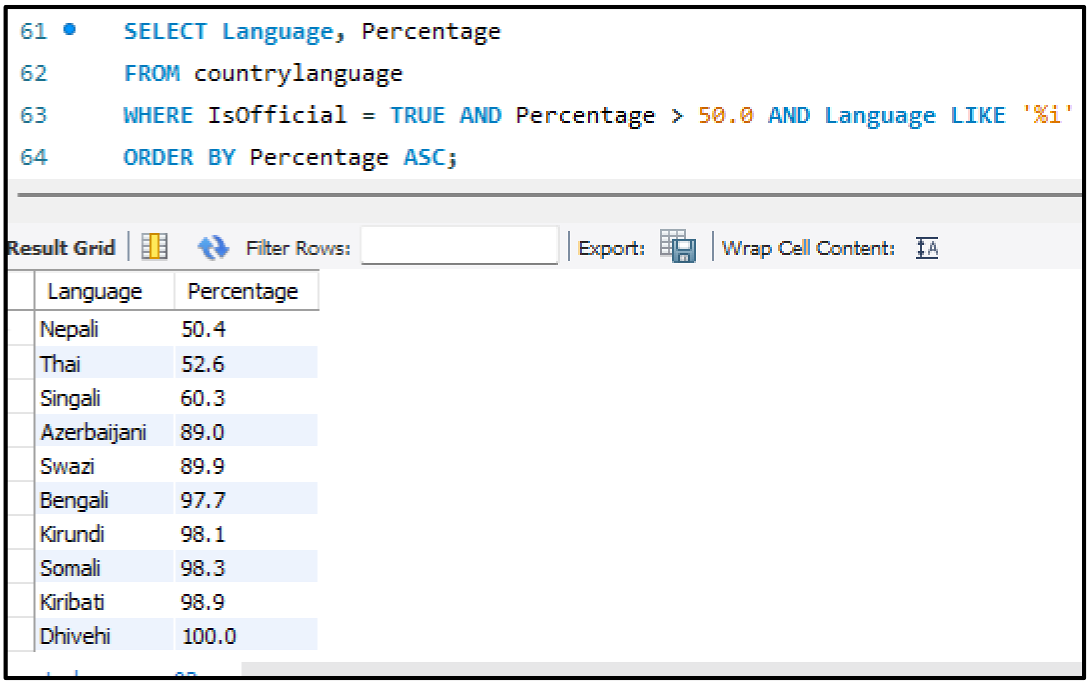
Result Grid | Filter Rows:

```
Population  Language
7160400     Dutch
7160400     English
7160400     Papiamento
7160400     Spanish
7160400     Balochi
7160400     Dari
7160400     Pashto
7160400     Turkmenian
7160400     Uzbek
```

Extra SQL Queries

Official languages above 50% which end with 'i'

The query in the snippet below selects the 'Language', 'IsOfficial' and 'Percentage' columns from the countrylanguage table. To collect the data for official languages which end with 'i' and is above 50%, the WHERE clause was used. As the 'IsOfficial' column has the Boolean datatype, the keyword 'TRUE' was used to obtain results which are official. Additionally, to filter the data that is above 50%, the '>' symbol was used and to obtain only results for languages ending with 'i' the '%' symbol was utilised. Lastly, to order the percentage from the lowest to highest, the function ORDER BY and the keyword ASC was used.

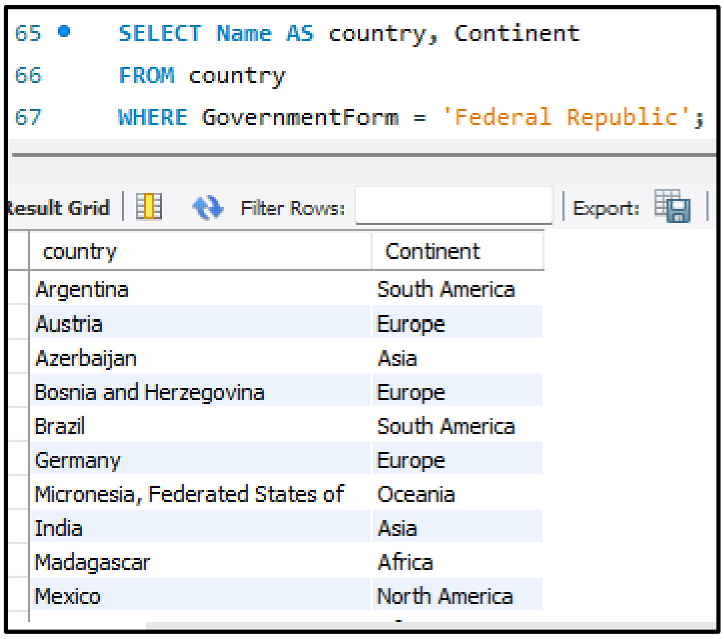


```sql
SELECT Language, Percentage
FROM countrylanguage
WHERE IsOfficial = TRUE AND Percentage > 50.0 AND Language LIKE '%i'
ORDER BY Percentage ASC;
```

Language	Percentage
Nepali	50.4
Thai	52.6
Singali	60.3
Azerbaijani	89.0
Swazi	89.9
Bengali	97.7
Kirundi	98.1
Somali	98.3
Kiribati	98.9
Dhivehi	100.0

Countries in the 'Federal Republic' government form

To obtain the country and continent names which is governed by 'Federal Republic', the 'Name' and 'Continent' columns were selected from the country table. Additionally, the WHERE clause was utilised to filter only countries that was governed by 'Federal Republic'.



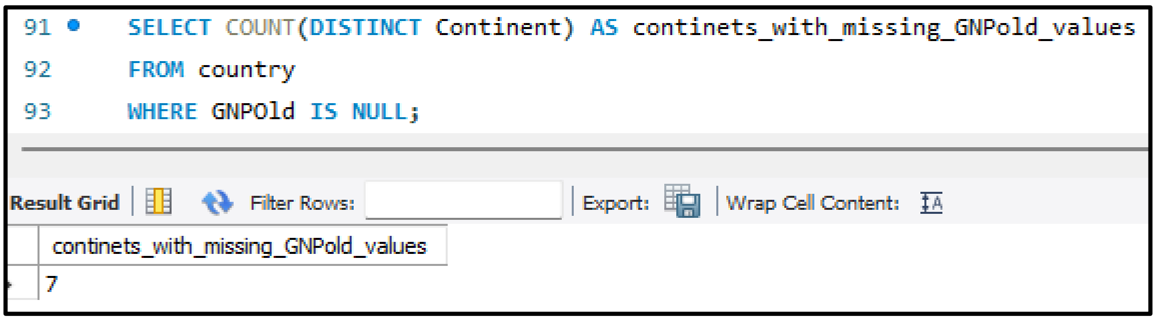
```sql
SELECT Name AS country, Continent
FROM country
WHERE GovernmentForm = 'Federal Republic';
```

country	Continent
Argentina	South America
Austria	Europe
Azerbaijan	Asia
Bosnia and Herzegovina	Europe
Brazil	South America
Germany	Europe
Micronesia, Federated States of	Oceania
India	Asia
Madagascar	Africa
Mexico	North America

Count the unique number of continent with a null value in GNPold

In the image below, the Continent column was uniquely counted by the COUNT(DISTINCT) function from the country table where the GNPold had an empty value. To filter the GNPold column, ‘IS NULL’ keyboard was utilised in a WHERE clause.

As presented in the snippet, there are 7 countries that had an empty value for the old gross national income.



```sql
SELECT COUNT(DISTINCT Continent) AS continets_with_missing_GNPold_values
FROM country
WHERE GNPOld IS NULL;
```

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

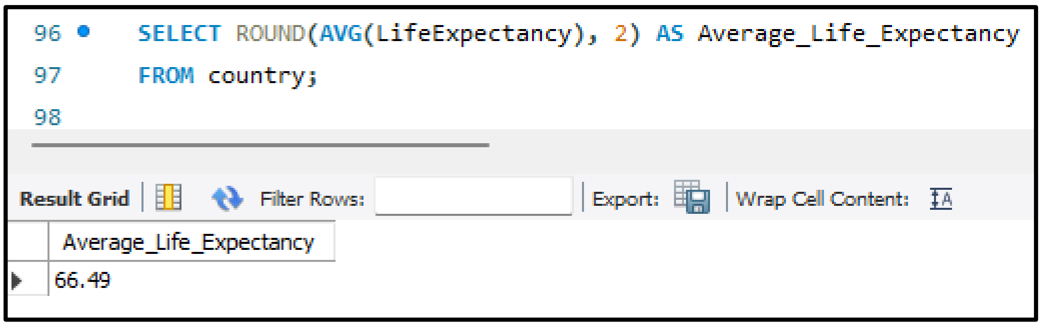
```
continets_with_missing_GNPold_values
7
```

ROUND () Average life expectancy in the whole world

To obtain the average life expectancy in the country table, the ‘LifeExpectancy’ column was selected and the aggregate function AVG() was used from the country table.

When the AVG() function was used the results was 66.48604 therefore, to round the results to two decimal places, the ROUND () function was utilised.

As shown on the image below, the average life expectancy in the country table, rounded to two decimal places is 66.49.



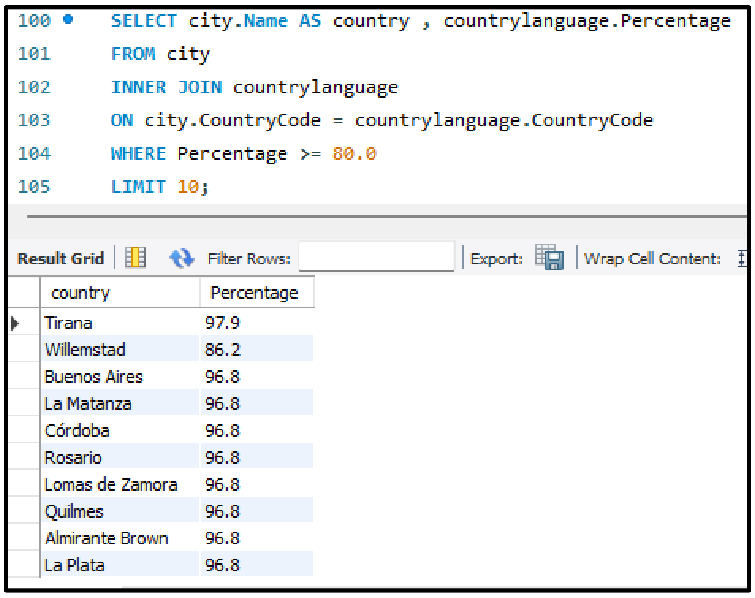
```sql
SELECT ROUND(AVG(LifeExpectancy), 2) AS Average_Life_Expectancy
FROM country;
```

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

```
Average_Life_Expectancy
66.49
```

Common cities in 'city' and 'countrylanguage' tables which are 80% and above

To retrieve the common cities in the city and countrylangage table, the 'Name' column from the city table and the 'Percentage' column from the countrylanguage table was selected. Then both city and countrylanguage tabled were joined using the INNER JOIN keyboard, this selects the records that have matching values in both tables. After that using the 'ON' keyboard and '=' symbol the CountryCode column from both city and countrylanguage tables were linked. As we would like the results for only countries which is 80% and above, a WHERE clause along with '>' and '=' symbols were utilised.



```
SELECT city.Name AS country , countrylanguage.Percentage
FROM city
INNER JOIN countrylanguage
ON city.CountryCode = countrylanguage.CountryCode
WHERE Percentage >= 80.0
LIMIT 10;
```

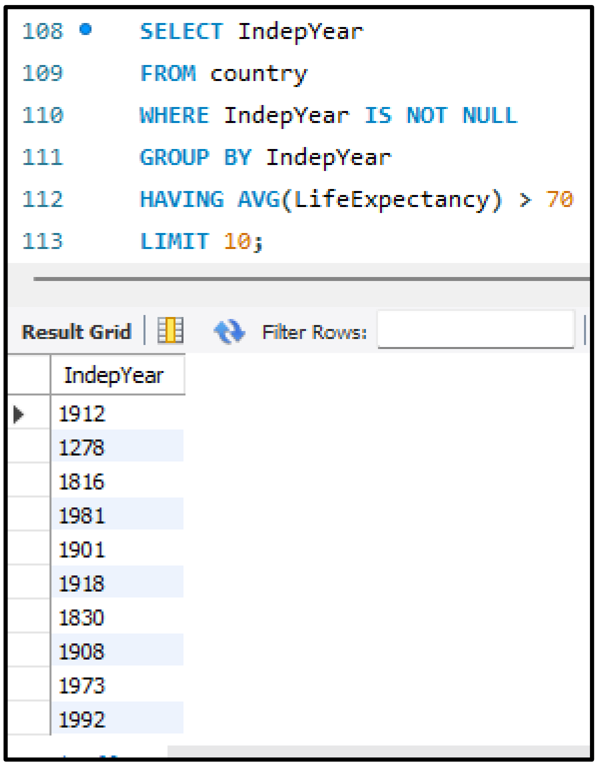
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

country	Percentage
Tirana	97.9
Willemstad	86.2
Buenos Aires	96.8
La Matanza	96.8
Córdoba	96.8
Rosario	96.8
Lomas de Zamora	96.8
Quilmes	96.8
Almirante Brown	96.8
La Plata	96.8

HAVING function- IndepYears with values which has the LifeExpectancy above 70 on average

The SQL query in the snippet selects the non-empty values from the IndepYear column in the country table.

As an aggregate function is used, the GROUP BY keyword is utilised to summarise the data. Additionally, a WHERE clause cannot be used to filter aggregate functions as groups have their own filtering keyword – HAVING. Therefore, HAVING keyboard is used to find the average life expectancy which is over 70.



```
SELECT IndepYear
FROM country
WHERE IndepYear IS NOT NULL
GROUP BY IndepYear
HAVING AVG(LifeExpectancy) > 70
LIMIT 10;
```

Result Grid | Filter Rows:

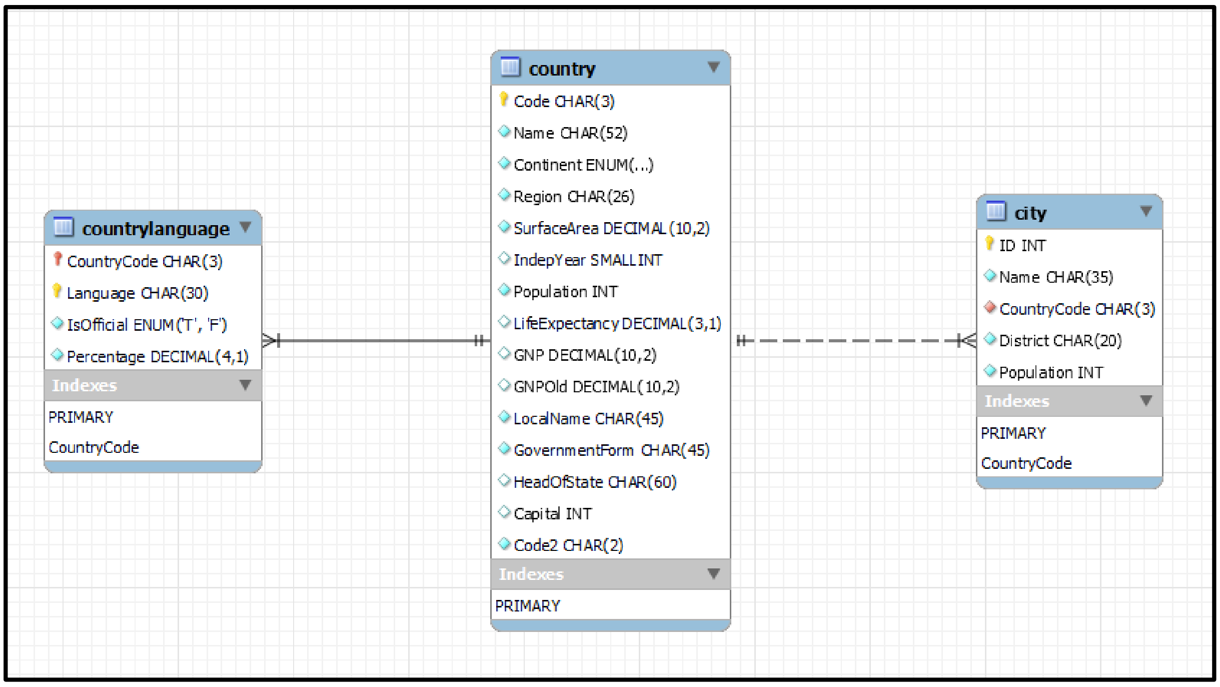
IndepYear
1912
1278
1816
1981
1901
1918
1830
1908
1973
1992

EER Diagram

The abbreviation EER stands for Enhanced Entity- Relationship and it is an advanced and evolved version of an Entity relationship (ER). It which helps create and maintain a detailed database through high levels models and tools. The EER diagrams are developed on the basic ER diagrams.

The advantage of EER models is that it is simple to develop and maintain and it is easy to understand. Additionally, due to how it is visual represented, it is easy to understand and maintain as well as helps display the relationships between the entities.

However, the drawbacks are that the EER diagrams have many limitations.



As you can see in the image above, there is a one-to-many relationship between the country and countrylanguage tables and there is also another one-to-many relationship between the country and city tables.

Identifying the primary keys in the country, city and countrylanguage tables

In the country table, the 'Code' column is the primary key, in the city table, the 'ID' column is the primary key and in the countrylanguage table, the 'Language' and 'CountryCode' columns form a composite primary key.

Identifying the foreign keys in the city, country and countrylanguage tables

There is no foreign key in the country and countrylanguage tables however in the city table the 'CountryCode' attribute is a foreign key.

Reflection

I dedicated time to researching many relationships within a relational database. Initially, I found de-normalisation challenging, utilising the available resources such as YouTube and study materials and by actively engaging with these resources, I was able to increase my understanding of de-normalisation. As a result of investing time, effort and dedication, I was able to overcome the challenges presented by complexities of de-normalisation and was able to understand its principles and applications in the context of relational databases.

In regards to the SQL queries, I've gone above and beyond the mandatory tasks as well as undertook the additional tasks. I took the initiative to go the extra mile by creating six additional SQL queries, utilising keywords that were not initially covered during the lesson. With a positive and enthusiastic approach, showcasing my passion for SQL and my eagerness to always expand my knowledge. This initiative enabled me to learn new techniques in SQL as well as demonstrated my ability to apply my knowledge in a practical setting, proving my adaptability and willingness to go beyond the expected requirements.

A challenge which I came across revolved around identifying the foreign key in the EER diagram, primarily due to my lack of knowledge with the coloured diamond shapes. Recognising the importance of addressing the knowledge gap, I invested extra time and effort into understanding the significance of these coloured diamond shapes within the context of the EER diagram. This effort gave me positive results as I gained a thorough understanding of the shapes and colour which enabled me to confidently revisit the EER diagram and accurately detect both the primary and foreign keys.

In conclusion, I consider this project to have been a boost in my learning journey, with the SQL tasks coming out as a particularly challenging and enjoyable part of the process. From my previous commitment with an E-learning course in PostgreSQL, I noticed major similarities between the keywords used in PostgreSQL and MySQL. This helped me to recognise the potential for further professional growth in SQL. Motivated by this, I made a commitment to invest time and effort into mastering advanced SQL techniques. By setting this goal, I am actively placing myself for continuous professional development, with the aim of achieving proficiency in SQL.

References

https://blog.devart.com/types-of-relationships-in-sql-server-database.html

https://medium.com/@jromasanta/a-step-by-step-database-normalization-for-dummies-or-not-1b32b725e1be

https://www.geeksforgeeks.org/denormalization-in-databases/

https://nulab.com/learn/software-development/er-diagrams-vs-eer-diagrams-whats-the-difference/

https://dev.mysql.com/doc/workbench/en/wb-eer-color-key.html

https://stackoverflow.com/questions/23041390/mysql-workbench-generate-tables-from-eer-diagram