**Tasks**

1. **Using count, get the number of cities in the USA?**

This query will count the number of rows in the cities table where the country column is equal to 'USA', giving you the number of cities in the USA.

**Query**: SELECT COUNT(Countrycode)

FROM city

WHERE Countrycode = 'USA';

1. **Using ORDER BY, LIMIT, what country has the highest life expectancy?**

This query will order the countries by life expectancy in descending order and then limit the result to only one row, which corresponds to the country with the highest life expectancy.

**Query:** SELECT name, LifeExpectancy

FROM country

ORDER BY LifeExpectancy DESC

LIMIT 1;

1. **Select 25 cities around the world that start with the letter 'F' in a single SQL query.**

This query selects all columns from the city table where the Name column starts with the letter 'F’ and limits the result to 25 rows.

**Query:** SELECT \* FROM city

WHERE Name LIKE 'F%'

LIMIT 25;

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

This query selects specific columns (Id, Name, Population) from the city table and restricts the result to the first 10 rows.

**Query:** SELECT Id, Name, Population

FROM city

LIMIT 10;

1. **Create a SQL statement to display columns Id, Name, Population from the city table and limit results to rows 31-40.**

This query will return the columns Id, Name, and Population from the "city" table, limited to rows 31-40 based on the Id column.

**LIMIT 30, 10**: Specifies that we want to retrieve 10 rows starting from the 31st row. The first number (30) represents the offset (number of rows to skip), and the second number (10) represents the number of rows to retrieve.

**Query:** SELECT Id, Name, Population

FROM city

ORDER BY Id

LIMIT 30, 10;

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

This query selects all columns from the city table and filters the result to only include cities where the Population column is larger than 2,000,000.

**Query:** SELECT \* FROM city

WHERE Population > 2000000;

1. **Create a SQL statement to find all city names from city table whose name begins with Be prefix.**

This query selects the Name column from the city table and filters the result to only include city names that begin with the prefix "Be" using the LIKE operator with the % wildcard character, which matches any sequence of characters.

**Query:** SELECT Name FROM city

WHERE Name LIKE 'Be%';

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

This query selects all columns (\*) from the city table and filters the result to only include cities where the Population column falls within the range of 500,000 to 1,000,000 using the BETWEEN operator. The BETWEEN operator is inclusive of both endpoints, so it will include cities with a population of exactly 500,000 or 1,000,000 in the result set.

**Query:** SELECT \* FROM city

WHERE Population BETWEEN 500000 AND 1000000;

1. **Create a SQL statement to display all cities from the city table sorted by Name in ascending order.**

This query will return all cities from the "city" table, sorted alphabetically by name in ascending order.

**Query:** SELECT \* FROM city

ORDER BY Name ASC;

1. **Create a SQL statement to find a city with the lowest population in the city table.**

This query selects all columns (\*) from the city table, orders the result by population in ascending order (ASC), which will put the cities with the lowest population at the top, and limits the result to only one row using LIMIT 1. Therefore, it will return the city with the lowest population.

**Query:** SELECT \* FROM city

ORDER BY Population ASC

LIMIT 1;

1. **Create a SQL statement to find a country with the largest population in the country table.**

This SQL query efficiently finds and retrieves the country with the highest population from the **country** table by sorting the rows based on the population column in descending order and then limiting the result to only the first row.

**Query:** SELECT \* FROM country

ORDER BY Population DESC

LIMIT 1;

**BONUS (These tasks require knowledge of JOINs. Challenge yourself. Don’t worry if you don’t get it right. )**

**Bonus Task1: Create a SQL statement to find the capital of Spain (ESP).**

**Solution**: This query retrieves the name of the capital city (city.Name) by joining the city and country tables. It matches the CountryCode column from the city table with the Code column from the country table and ensures that the city ID (city.ID) matches the capital ID (country.Capital) in the country table for the specified country code ('ESP').

**Query**: SELECT city.Name AS capital

FROM city

JOIN

country ON city.CountryCode = country.Code

WHERE

country.Code = 'ESP' AND country.Capital = city.ID;

**Bonus Task2: Create a SQL statement to find the country with the highest life expectancy.**

**Solution:** This SQL query efficiently finds and retrieves the country with the highest life expectancy from the **country** table by sorting the rows based on the life expectancy column in descending order and then limiting the result to only the first row.

**Query:** SELECT Name, LifeExpectancy

FROM country

ORDER BY LifeExpectancy DESC

LIMIT 1;

**Bonus Task3: Create a SQL statement to find all cities from the Europe continent.**

**Solution**: This query retrieves the names of all cities (city.Name) from the Europe continent by joining the city and country tables. It matches the CountryCode column from the city table with the Code column from the country table and filters for countries in the Europe continent (country.Continent = 'Europe').

**Query**: SELECT city.Name AS city\_name

FROM city

JOIN

country ON city.CountryCode = country.Code

WHERE

country.Continent = 'Europe';

**Bonus Task4: Create a SQL statement to find the most populated city in the city table.**

**Solution**:

**SELECT Name, population**: Specifies that we want to retrieve the Name and population columns from the city table.

**FROM city**: Indicates that we are querying data from the city table.

**WHERE population = (SELECT MAX (population) FROM city)**: Filters the rows from the city table where the population column matches the maximum population value obtained from the subquery. The subquery (**SELECT MAX(population) FROM city**) is executed first to determine the maximum population value in the **city** table.

**Query**: SELECT Name, population FROM city

WHERE population = (SELECT MAX(population) FROM city);