

Data Technician

Name:

Course Date:

Table of contents

Day 1: Task 1	3
Day 1: Task 2	5
Day 3: Task 1	7
Day 4: Task 1: Written	10
Day 4: Task 2: SQL Practical	15
Course Notes	29
Additional Information	30



Day 1: Task 1

Please research and complete the below questions relating to key concepts of databases.

What is a primary key?	<p>A primary key is a unique identifier for each record in a database table. It ensures that each record can be uniquely identified and does not have duplicates.</p> <p>Properties of a Primary Key:</p> <ul style="list-style-type: none">➤ Unique: No two records can have the same primary key value.➤ Not Null: Every record must have a primary key value (it cannot be left empty).➤ Immutable: It should not change once it's set.
How does this differ from a secondary key?	<p>A secondary key is any field (or set of fields) used to search or retrieve data but does not have to be unique.</p> <p>Key Characteristics:</p> <ul style="list-style-type: none">➤ Can be duplicate: Multiple records can share the same value for a secondary key.➤ Used for querying: Helps in searching for records based on criteria other than the primary key.
How are primary and foreign keys related?	<ul style="list-style-type: none">● Primary Key: Identifies a unique record in a table.● Foreign Key: A foreign key is a field (or combination of fields) in one table that links to the primary key of another table. It establishes a relationship between the two tables. <p>Relationship:</p> <ul style="list-style-type: none">➤ A primary key ensures the uniqueness of data within its own table.➤ A foreign key establishes a relationship between two tables by referring to the primary key in another table.
Provide a real-world example of a one-to-one relationship	<p>Each person has one passport, and each passport belongs to only one person.</p>
Provide a real-world	<p>A single author can write many books, but each book is written by only one author.</p>



example of a one-to-many relationship	
Provide a real-world example of a many-to-many relationship	A student can enroll in many courses, and each course can have many students.

Day 1: Task 2

Please research and complete the below questions relating to key concepts of databases.

What is the difference between a relational and non-relational database?	<p>Relational Database:</p> <ul style="list-style-type: none">● Stores data in tables with rows and columns.● Uses SQL for querying.● Has a predefined schema and relationships (via foreign keys).● Examples: MySQL, PostgreSQL. <p>Non-relational Database:</p> <ul style="list-style-type: none">● Stores data in formats like JSON, key-value, documents, or graphs.● Uses NoSQL queries.● Schema-less and doesn't require predefined relationships.● Examples: MongoDB, Cassandra, Redis.
What type of data would benefit off the non-relational model? Why?	<ul style="list-style-type: none">● Unstructured/Semi-structured Data<ul style="list-style-type: none">➤ Example: Social media posts, comments.➤ Why: Flexible schema (JSON format) to handle varied data types.● High-Volume, High-Speed Data<ul style="list-style-type: none">➤ Example: IoT data, logs.➤ Why: Optimized for high-speed reads/writes and horizontal scalability.● Flexible Schema Needs<ul style="list-style-type: none">➤ Example: Product catalogs, user profiles.➤ Why: No need for a fixed schema, allowing easy data structure changes.● Graph-Based Data<ul style="list-style-type: none">➤ Example: Social networks, recommendations.➤ Why: Graph databases like Neo4j handle complex



relationships efficiently.

- **Scalable Applications**

- **Example:** Global platforms (e-commerce).
- **Why:** Horizontal scalability across multiple servers for large datasets.

- **Summary**

All of these data types benefit from NoSQL databases due to their flexibility, scalability, and ability to handle large, unstructured datasets efficiently.

Day 3: Task 1

Please research the below 'JOIN' types, explain what they are and provide an example of the types of data it would be used on.

Self-join	<ul style="list-style-type: none">• What it is: A self-join is a regular join, but the table is joined with itself.• When to use: When you want to compare rows within the same table — often used for hierarchical relationships like employees and their managers. <p>Extra Code Example (Use case: Find each employee's manager using the same employees table):</p> <pre>SELECT e1.name AS Employee, e2.name AS Manager FROM employees e1 JOIN employees e2 ON e1.manager_id = e2.id;</pre>
Right join	<ul style="list-style-type: none">• What it is: Returns all rows from the right table and the matched rows from the left table. If there's no match, <code>NULL</code> is returned for left table columns.• When to use: When you want to include all rows from the right table, even if there's no matching data on the left. <p>Extra Code Example (Use case: Get a list of all customers, including those who haven't made any orders):</p> <pre>SELECT orders.order_id, customers.name FROM orders RIGHT JOIN customers ON orders.customer_id = customers.id;</pre>
Full join	<ul style="list-style-type: none">• What it is: Combines the results of both left and right joins. Returns all rows from both tables, matching them where possible.• When to use: When you want to see all data from both tables,



	<p>including unmatched records on either side.</p> <p>Extra Code Example (Use case: Show all employees and all projects, whether or not they're assigned to each other):</p> <pre>SELECT employees.name, projects.project_name FROM employees FULL JOIN projects ON employees.project_id = projects.id;</pre>
Inner join	<ul style="list-style-type: none"> • What it is: Returns only rows where there is a match in both tables. • When to use: When you want only the data that exists in both tables — the most common type of join. <p>Extra Code Example (Use case: Show only customers who have made orders):</p> <pre>SELECT orders.order_id, customers.name FROM orders INNER JOIN customers ON orders.customer_id = customers.id;</pre>
Cross join	<ul style="list-style-type: none"> • What it is: Returns the Cartesian product of both tables — every row from the first table combined with every row from the second. • When to use: When you want all possible combinations — usually for generating test data or pairing items from different categories. <p>Extra Code Example (Use case: Create a list of all possible color-size combinations for a product):</p> <pre>SELECT color.name, size.label FROM colors CROSS JOIN sizes;</pre>
Left join	<ul style="list-style-type: none"> • What it is: Returns all rows from the left table, and matched rows from the right table. If there is no match, returns <code>NULL</code> on the right side.



- **When to use:** When you want all records from the left table, even if there's no match in the right table.

Extra Code Example (Use case: **Get a list of all customers, including those with no orders**):

```
SELECT customers.name, orders.order_id  
FROM customers  
LEFT JOIN orders ON customers.id = orders.customer_id;
```



Day 4: Task 1: Written

In your groups, discuss and complete the below activity. You can either nominate one writer or split the elements between you. Everyone however must have the completed work below:

Imagine you have been hired by a small retail business that wants to streamline its operations by creating a new database system. This database will be used to manage inventory, sales, and customer information. The business is a small corner shop that sells a range of groceries and domestic products. It might help to picture your local convenience store and think of what they sell. They also have a loyalty program, which you will need to consider when deciding what tables to create.

Write a 500-word essay explaining the steps you would take to set up and create this database. Your essay should cover the following points:

1. **Understanding the Business Requirements:**
 - a. *What kind of data will the database need to store?*
 - b. *Who will be the users of the database, and what will they need to accomplish?*
2. **Designing the Database Schema:**
 - a. *How would you structure the database tables to efficiently store inventory, sales, and customer information?*
 - b. *What relationships between tables are necessary (e.g., how sales relate to inventory and customers)?*
3. **Implementing the Database:**
 - a. *What SQL commands would you use to create the database and its tables?*
 - b. *Provide examples of SQL statements for creating tables and defining relationships between them.*
4. **Populating the Database:**
 - a. *How would you input initial data into the database? Give examples of SQL INSERT statements.*
5. **Maintaining the Database:**
 - a. *What measures would you take to ensure the database remains accurate and up to date?*
 - b. *How would you handle backups and data security?*

Your essay should include specific examples of SQL commands and explain why each step is necessary for creating a functional and efficient database for the retail business.

Please write



1. Understanding the Business Requirements

a. Data Storage Needs:

The business needs to store information on inventory (names, categories, prices, quantities, and suppliers), sales (items sold, sale date, and amounts), customers (names, contact details, and loyalty points), suppliers (names, phones and accounts) and loyalty transactions (points earned or redeemed). This data will support daily operations and strategic decisions.

b. Database Users:

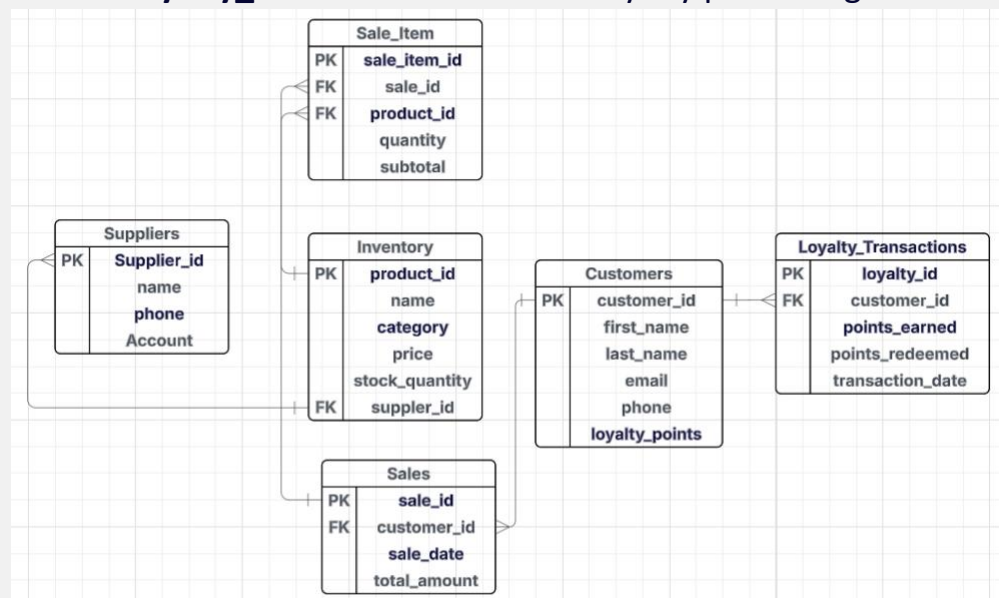
Typical users include cashiers, store managers, and database administrators. Cashiers will record sales and update inventory, managers will monitor stock and analyze sales, and admins will maintain data integrity and security.

2. Designing the Database Schema

a. Table Structure:

The database will include these main tables:

- **Customers:** to store personal and loyalty information.
- **Inventory:** to manage inventory.
- **Suppliers:** to record every supplier
- **Sales:** to record each transaction.
- **Sale_Items:** to track each product within a sale.
- **Loyalty_Transactions:** to track loyalty point usage.



b. Relationships:

- Each Sale is linked to one Customer.
- Each Sale contains multiple Products (via Sale_Items).
- Each Product can appear in many sales.
- Each Supplier supply multiple products.
- Loyalty transactions are tied to customers.

3. Implementing the Database

a. SQL Commands to Create Tables:

```
CREATE DATABASE retail_store;
```

```
USE retail_store;
```

```
CREATE TABLE Customers (  
    customer_id INT PRIMARY KEY AUTO_INCREMENT,  
    first_name VARCHAR(50),  
    last_name VARCHAR(50),  
    email VARCHAR(100),  
    phone VARCHAR(20),  
    loyalty_points INT DEFAULT 0  
);
```

```
CREATE TABLE Inventory (  
    product_id INT PRIMARY KEY AUTO_INCREMENT,  
    name VARCHAR(100),  
    category VARCHAR(50),  
    price DECIMAL(10,2),  
    stock_quantity INT,  
    supplier VARCHAR(100)  
);
```

```
CREATE TABLE Suppliers (  
    Supplier_id INT PRIMARY KEY AUTO_INCREMENT,  
    name VARCHAR(100),  
    phone VARCHAR(50),  
    account VARCHAR(100)  
);
```

```
CREATE TABLE Sales (  
    sale_id INT PRIMARY KEY AUTO_INCREMENT,
```

```

customer_id INT,
sale_date DATE,
total_amount DECIMAL(10,2),
FOREIGN KEY (customer_id) REFERENCES
Customers(customer_id)
);

CREATE TABLE Sale_Items (
sale_item_id INT PRIMARY KEY AUTO_INCREMENT,
sale_id INT,
product_id INT,
quantity INT,
subtotal DECIMAL(10,2),
FOREIGN KEY (sale_id) REFERENCES Sales(sale_id),
FOREIGN KEY (product_id) REFERENCES Products(product_id)
);

CREATE TABLE Loyalty_Transactions (
loyalty_id INT PRIMARY KEY AUTO_INCREMENT,
customer_id INT,
points_earned INT,
points_redeemed INT,
transaction_date DATE,
FOREIGN KEY (customer_id) REFERENCES
Customers(customer_id)
);

```

These SQL statements establish the schema and enforce relationships to maintain referential integrity.

4. Populating the Database

a. Inserting Initial Data:

```

INSERT INTO Inventory (name, category, price, stock_quantity,
supplier)
VALUES ('Chocolate Stick', 'Bakery', 3.20, 50, 'Fresh Bakes Ltd.');
```

```

INSERT INTO Customers (name, email, phone)
VALUES ('Sean Chen', 'Sean@example.com', '444-1234');
```



The statements above show sample sql codes to test and run the system.

5. Maintaining the Database

a. Ensuring Accuracy:

Implement input validation, use foreign keys, and regularly audit data for errors. Automated scripts can alert for low inventory or inactive customers.

b. Backups and Security:

Schedule daily backups using SQL tools. Limit access with user roles (e.g., READ ONLY, ADMIN) and encrypt sensitive data like emails. Use SSL for remote connections and enable logging for traceability.



Day 4: Task 2: SQL Practical

In your groups, work together to answer the below questions. It may be of benefit if one of you shares your screen with the group and as a team answer / take screen shots from there.

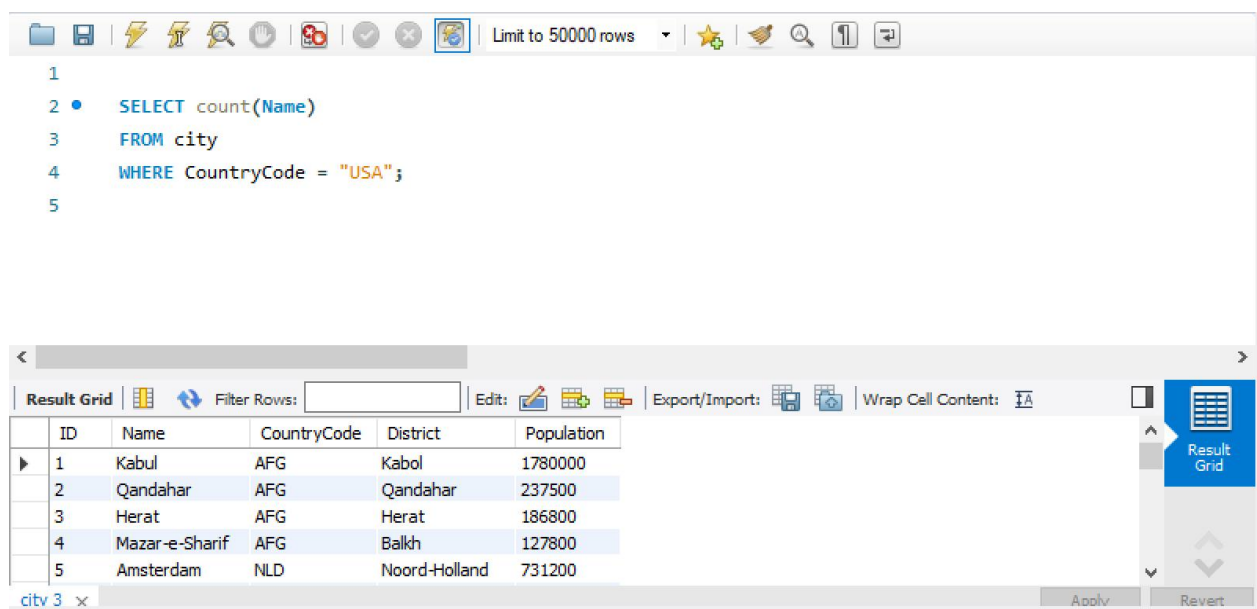
Setting up the database:

1. Download world_db(1)
2. Follow each step to create your database

For each question I would like to see both the syntax used and the output.

1. **Count Cities in USA:** *Scenario:* You've been tasked with conducting a demographic analysis of cities in the United States. Your first step is to determine the total number of cities within the country to provide a baseline for further analysis.

```
SELECT count(Name)
FROM city
WHERE CountryCode = "USA";
```



The screenshot shows a SQL query editor with a toolbar at the top. The query is entered in the main text area:

```
1
2 • SELECT count(Name)
3 FROM city
4 WHERE CountryCode = "USA";
5
```

Below the query editor, the 'Result Grid' is displayed, showing a table with 5 rows and 5 columns: ID, Name, CountryCode, District, and Population. The first five rows of data are visible:

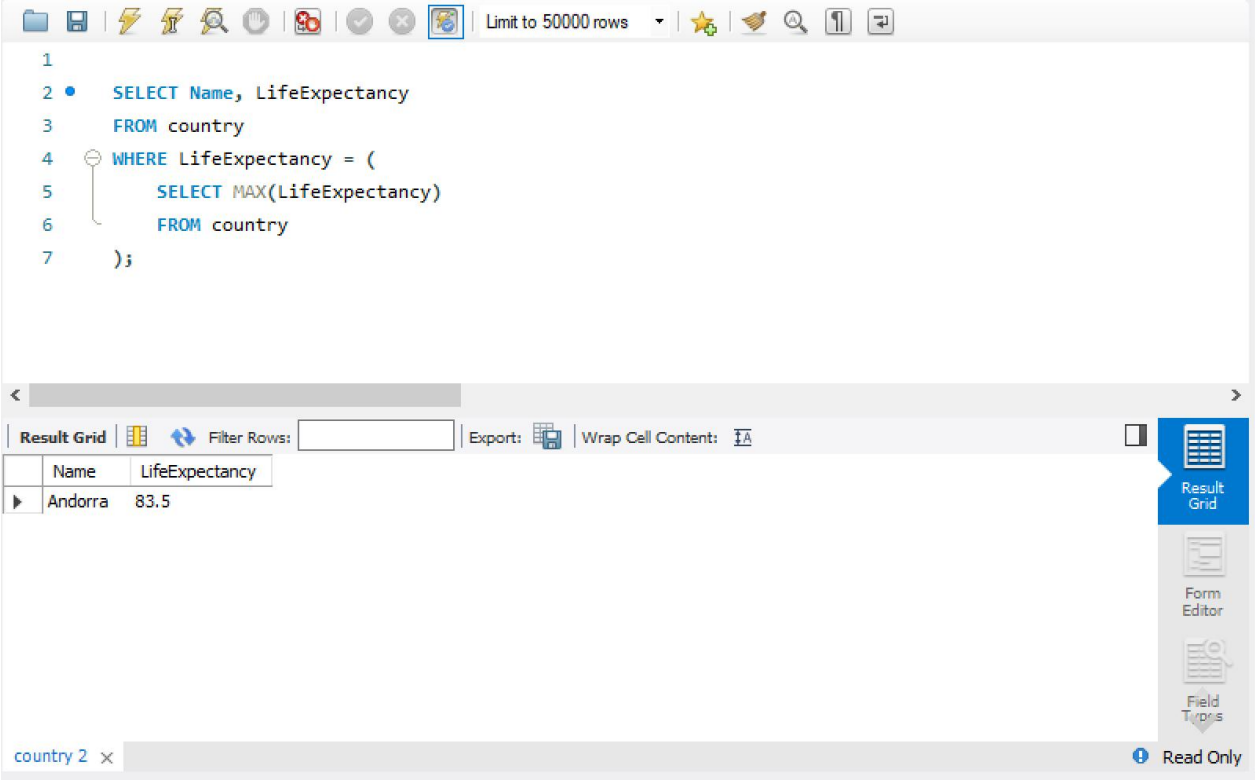
ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabul	1780000
2	Qandahar	AFG	Qandahar	237500
3	Herat	AFG	Herat	186800
4	Mazar-e-Sharif	AFG	Balkh	127800
5	Amsterdam	NLD	Noord-Holland	731200

The 'Result Grid' tab is selected on the right side of the interface. The 'Apply' and 'Revert' buttons are visible at the bottom right of the result grid.

2. **Country with Highest Life Expectancy:** *Scenario:* As part of a global health initiative, you've been assigned to identify the country with the highest life

expectancy. This information will be crucial for prioritising healthcare resources and interventions.

```
SELECT Name, LifeExpectancy
FROM country
WHERE LifeExpectancy = (
    SELECT MAX(LifeExpectancy)
    FROM country
);
```



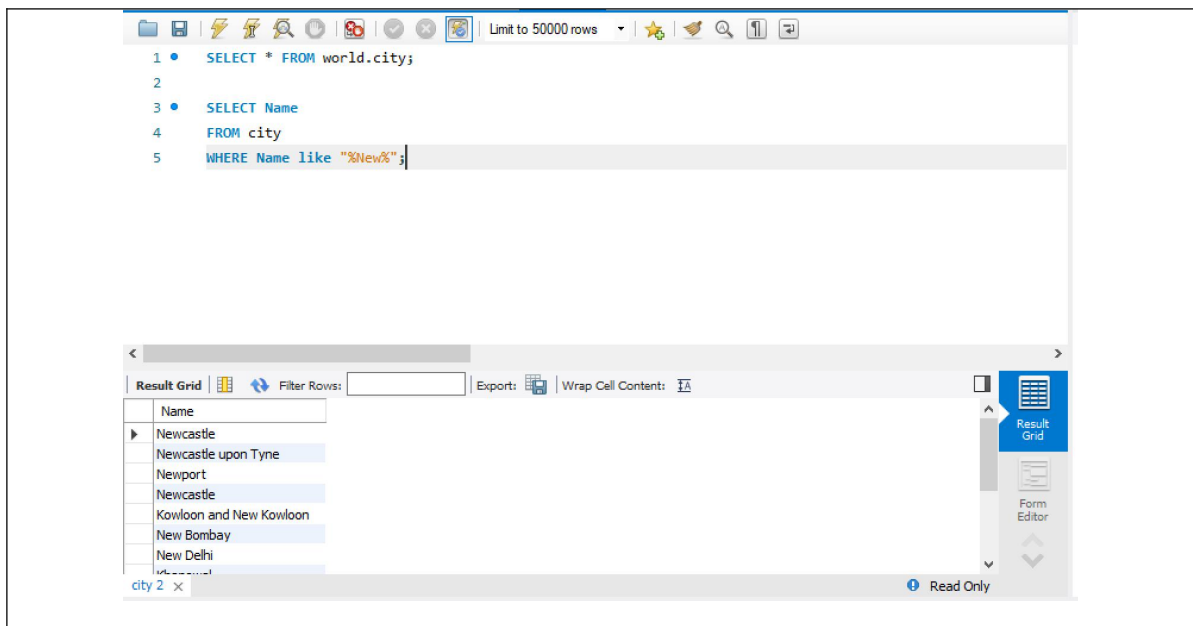
The screenshot shows a SQL query editor interface. The query is: `SELECT Name, LifeExpectancy FROM country WHERE LifeExpectancy = (SELECT MAX(LifeExpectancy) FROM country);`. The result grid displays one row:

Name	LifeExpectancy
Andorra	83.5

. The interface includes a toolbar with various icons, a 'Limit to 50000 rows' dropdown, and a 'Read Only' status indicator.

3. **"New Year Promotion: Featuring Cities with 'New' :** *Scenario:* In anticipation of the upcoming New Year, your travel agency is gearing up for a special promotion featuring cities with names including the word 'New'. You're tasked with swiftly compiling a list of all cities from around the world. This curated selection will be essential in creating promotional materials and enticing travellers with exciting destinations to kick off the New Year in style.

```
SELECT Name
FROM city
WHERE Name like "%New%";
```

4. **Display Columns with Limit (First 10 Rows):** *Scenario:* You're tasked with providing a brief overview of the most populous cities in the world. To keep the report concise, you're instructed to list only the first 10 cities by population from the database.

SELECT Name, Population
FROM city
ORDER BY Population DESC
Limit 10;

The screenshot shows a database query editor with the following SQL query:

```

1 • SELECT Name, Population
2 FROM city
3 ORDER BY Population DESC
4 Limit 10;

```

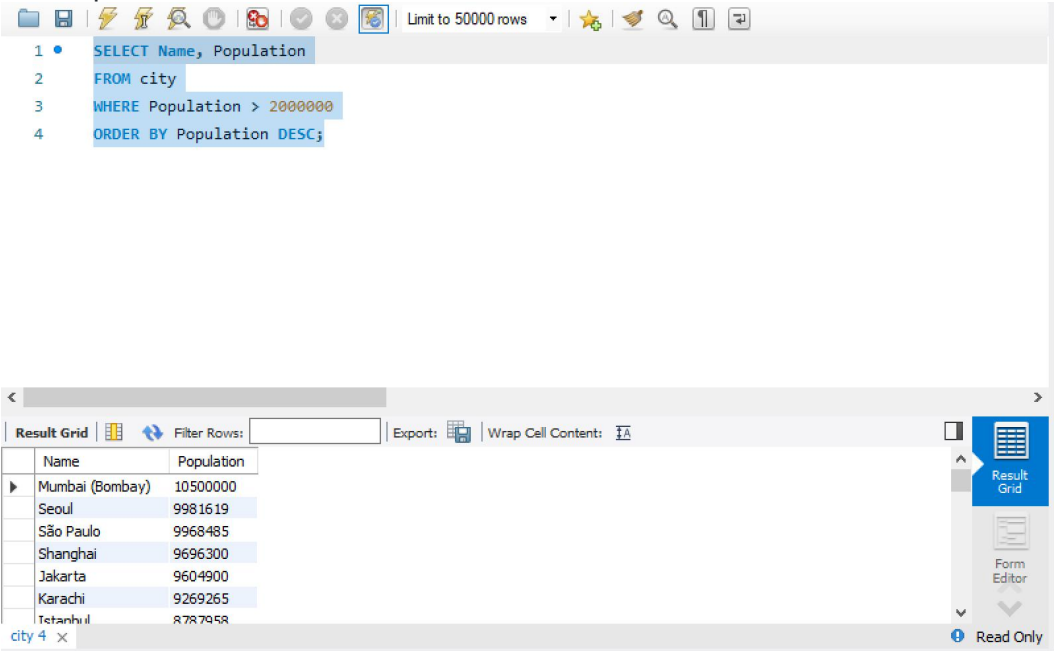
Below the query editor, the 'Result Grid' is displayed, showing the top 10 most populous cities:

Name	Population
Mumbai (Bombay)	10500000
Seoul	9981619
São Paulo	9968485
Shanghai	9696300
Jakarta	9604900
Karachi	9269265
Tetrahul	8787958

The interface includes a toolbar with icons for file operations, a 'Limit to 50000 rows' dropdown, and a 'Read Only' status indicator.

5. **Cities with Population Larger than 2,000,000:** *Scenario:* A real estate developer is interested in cities with substantial population sizes for potential investment opportunities. You're tasked with identifying cities from the database with populations exceeding 2 million to focus their research efforts.

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



The screenshot shows a database query tool interface. The top section displays the SQL query: `SELECT Name, Population FROM city WHERE Population > 2000000 ORDER BY Population DESC;`. Below the query, a toolbar includes icons for file operations, a 'Limit to 50000 rows' dropdown, and other utility icons. The bottom section shows the 'Result Grid' with a table of results. The table has two columns: 'Name' and 'Population'. The results are as follows:

Name	Population
Mumbai (Bombay)	10500000
Seoul	9981619
São Paulo	9968485
Shanghai	9696300
Jakarta	9604900
Karachi	9269265
Tetrahul	8787958

On the right side of the result grid, there are buttons for 'Result Grid', 'Form Editor', and a 'Read Only' status indicator.

6. **Cities Beginning with 'Be' Prefix:** *Scenario:* A travel blogger is planning a series of articles featuring cities with unique names. You're tasked with compiling a list of cities from the database that start with the prefix 'Be' to assist in the blogger's content creation process.

```
SELECT Distinct Name
FROM city
WHERE Name like "BE%";
```

1 • **SELECT Distinct Name**
 2 **FROM city**
 3 **WHERE Name like "B%";**

Result Grid

Name
Béjaia
Béchar
Benguela
Berazategui
Belize City
Belmopan
Belo Horizonte

city 8 x

7. **Cities with Population Between 500,000-1,000,000:** *Scenario:* An urban planning committee needs to identify mid-sized cities suitable for infrastructure development projects. You're tasked with identifying cities with populations ranging between 500,000 and 1 million to inform their decision-making process.

SELECT Name, Population
FROM city
WHERE Population Between 500000 AND 1000000
ORDER BY Population;

2 **FROM city**
 3 **WHERE Population Between 500000 AND 1000000**
 4 **ORDER BY Population;**

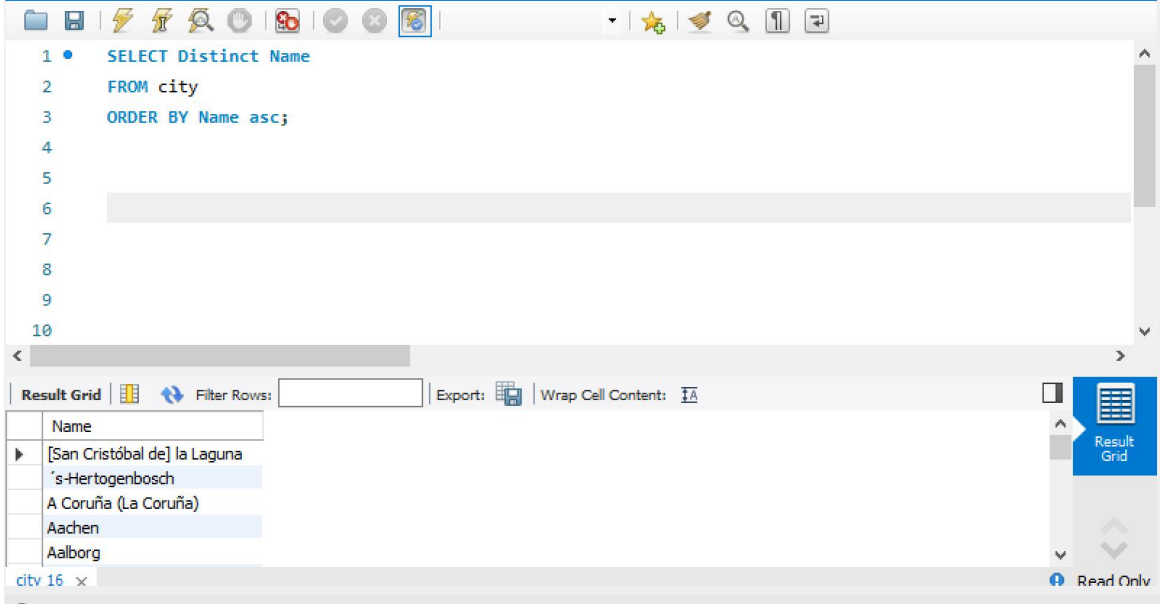
Result Grid

Name	Population
Amsterdam	731200
Rotterdam	593321
Oran	609823
Dubai	669181
Rosario	907718

city 13 x

8. **Display Cities Sorted by Name in Ascending Order:** *Scenario:* A geography teacher is preparing a lesson on alphabetical order using city names. You're tasked with providing a sorted list of cities from the database in ascending order by name to support the lesson plan.

```
SELECT Distinct Name
FROM city
ORDER BY Name ASC;
```



The screenshot shows a database query editor with a toolbar at the top. The SQL query is entered in the main text area: `SELECT Distinct Name`, `FROM city`, and `ORDER BY Name asc;`. Below the query, a 'Result Grid' is displayed with a table containing city names. The table has a single column labeled 'Name'. The first five rows are visible: '[San Cristóbal de] la Laguna', 's-Hertogenbosch', 'A Coruña (La Coruña)', 'Aachen', and 'Aalborg'. The table is scrollable, and a 'Read Only' status is indicated at the bottom right.

Name
[San Cristóbal de] la Laguna
s-Hertogenbosch
A Coruña (La Coruña)
Aachen
Aalborg

9. **Most Populated City:** *Scenario:* A real estate investment firm is interested in cities with significant population densities for potential development projects. You're tasked with identifying the most populated city from the database to guide their investment decisions and strategic planning.

```
SELECT Name, Population
FROM city
ORDER BY Population DESC
LIMIT 1;
```

The screenshot shows a database query editor with the following SQL query:

```

1 • SELECT Name, Population
2 FROM city
3 ORDER BY Population DESC
4 LIMIT 1;
5

```

Below the query editor, the 'Result Grid' is displayed with the following data:

Name	Population
Mumbai (Bombay)	10500000

The interface includes a toolbar at the top with icons for file operations, a 'Limit to 50000 rows' dropdown, and a 'Read Only' status at the bottom right.

10. **City Name Frequency Analysis: Supporting Geography Education Scenario:** In a geography class, students are learning about the distribution of city names around the world. The teacher, in preparation for a lesson on city name frequencies, wants to provide students with a list of unique city names sorted alphabetically, along with their respective counts of occurrences in the database. You're tasked with this sorted list to support the geography teacher.

```

SELECT Distinct Name, count(Name) AS Name_frequency
FROM city
GROUP BY Name
ORDER BY Name ASC;

```

The screenshot shows a database query editor with the following SQL query:

```

1 • SELECT Distinct Name, count(Name) AS Name_frequency
2 FROM city
3 GROUP BY Name
4 ORDER BY Name ASC;
5

```

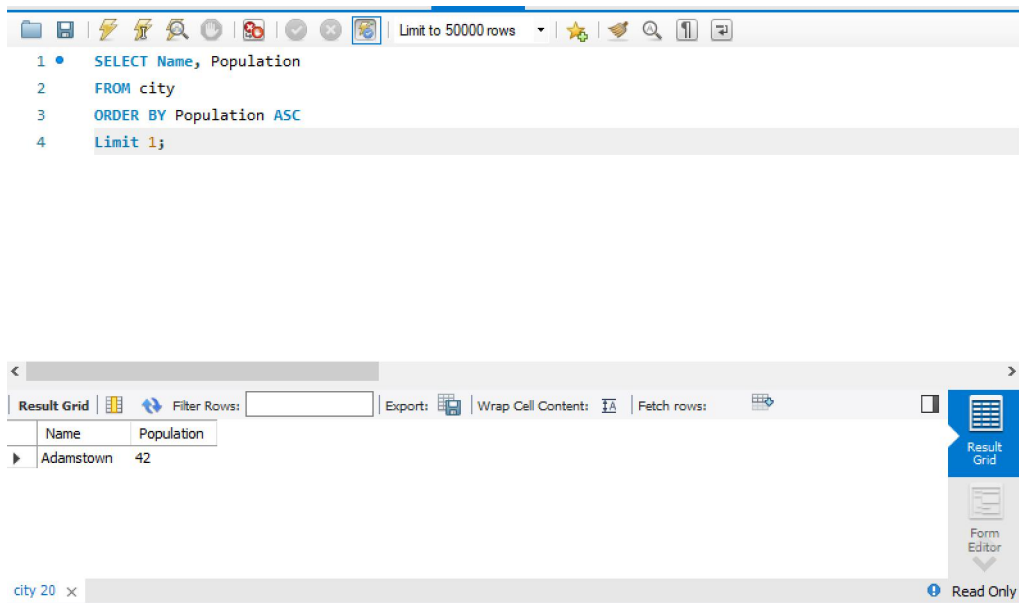
Below the query editor, the 'Result Grid' is displayed with the following data:

Name	Name_frequency
[San Cristóbal de] la Laguna	1
's-Hertogenbosch	1
A Coruña (La Coruña)	1
Aachen	1
Aalborg	1
Aba	1

The interface includes a toolbar at the top with icons for file operations, a 'Limit to 50000 rows' dropdown, and a 'Read Only' status at the bottom right.

11. **City with the Lowest Population:** *Scenario:* A census bureau is conducting an analysis of urban population distribution. You're tasked with identifying the city with the lowest population from the database to provide a comprehensive overview of demographic trends.

```
SELECT Name, Population
FROM city
ORDER BY Population ASC
Limit 1;
```



12. **Country with Largest Population:** *Scenario:* A global economic research institute requires data on countries with the largest populations for a comprehensive analysis. You're tasked with identifying the country with the highest population from the database to provide valuable insights into demographic trends.

```
SELECT Name, Population
FROM country
ORDER BY Population DESC
LIMIT 1;
```

The screenshot shows a database query editor with the following SQL query:

```

1 • SELECT Name, Population
2 FROM country
3 ORDER BY Population DESC
4 LIMIT 1;
5

```

The results are displayed in a table with columns 'Name' and 'Population':

Name	Population
China	1277558000

The interface includes a toolbar with icons for saving, undo, redo, and other database operations. A 'Limit to 50000 rows' dropdown is visible. The bottom status bar indicates 'country 3' and 'Read Only'.

13. **Capital of Spain:** *Scenario:* A travel agency is organising tours across Europe and needs accurate information on capital cities. You're tasked with identifying the capital of Spain from the database to ensure itinerary accuracy and provide travellers with essential destination information.

SELECT country.Name AS Country, city.Name AS city
FROM country
LEFT JOIN city
ON country.Capital = city.ID
WHERE country.Name = "Spain";

The screenshot shows a database query editor with the following SQL query:

```

1 • SELECT country.Name AS Country, city.Name AS city
2 FROM country
3 LEFT JOIN city
4 ON country.Capital = city.ID
5 WHERE country.Name = "Spain";

```

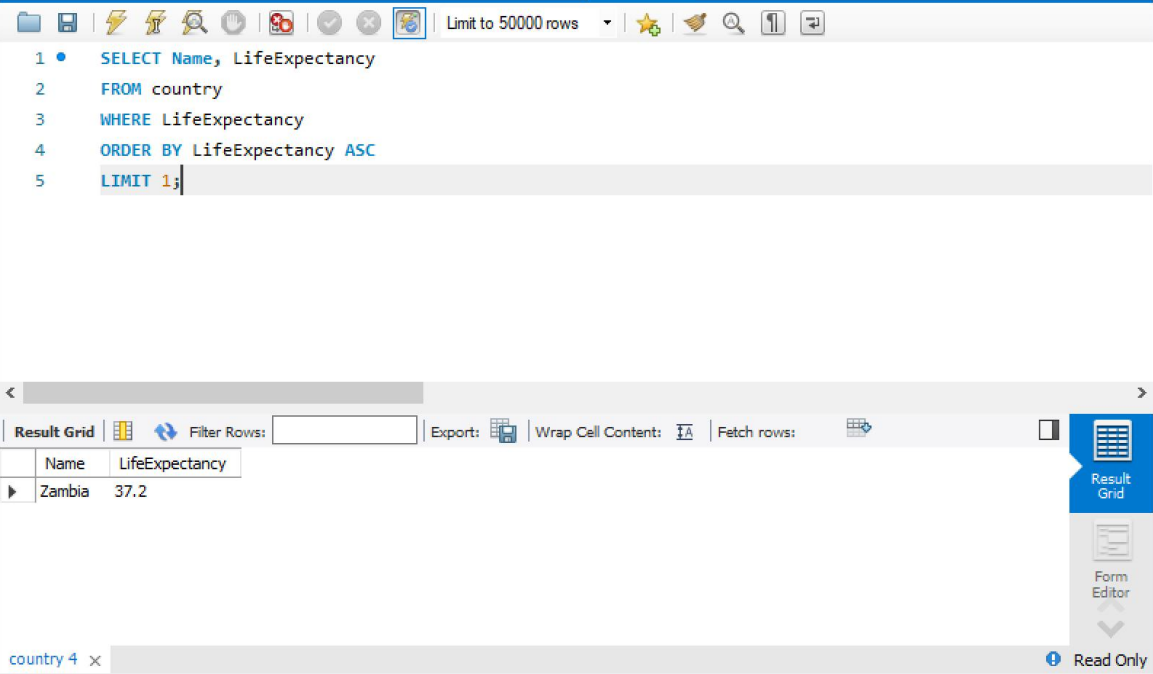
The results are displayed in a table with columns 'Country' and 'city':

Country	city
Spain	Madrid

The interface includes a toolbar with icons for saving, undo, redo, and other database operations. A 'Limit to 50000 rows' dropdown is visible. The bottom status bar indicates 'Result 3' and 'Read Only'.

14. **Country with Shortest Life Expectancy:** *Scenario:* A healthcare foundation is conducting research on global health indicators. You're tasked with identifying the country with the shortest life expectancy from the database to inform their efforts in improving healthcare systems and policies.

```
SELECT Name, LifeExpectancy
FROM country
WHERE LifeExpectancy
ORDER BY LifeExpectancy ASC
LIMIT 1;
```



The screenshot shows a database query editor interface. The top toolbar includes icons for file operations, a 'Limit to 50000 rows' dropdown, and other utility icons. The SQL query is entered in a text area and numbered 1 through 5. Below the query, a 'Result Grid' tab is active, displaying a table with two columns: 'Name' and 'LifeExpectancy'. The table contains one row with the data 'Zambia' and '37.2'. To the right of the table are buttons for 'Export', 'Wrap Cell Content', 'Fetch rows', 'Result Grid', and 'Form Editor'. At the bottom, a tab labeled 'country 4' is visible, and a 'Read Only' status indicator is present.

Name	LifeExpectancy
Zambia	37.2

15. **Cities in Europe:** *Scenario:* A European cultural exchange program is seeking to connect students with cities across the continent. You're tasked with compiling a list of cities located in Europe from the database to facilitate program planning and student engagement.

```
SELECT country.Continent AS Continent, city.Name AS city
FROM country
LEFT JOIN city
ON country.Code = city.CountryCode
WHERE country.Continent = "Europe";
```


The screenshot shows a SQL query editor with the following query:

```

1 • SELECT country.Continent AS Continent, city.Name AS city
2 FROM country
3 LEFT JOIN city
4 ON country.Code = city.CountryCode
5 WHERE country.Continent = "Europe";

```

Below the query editor is a "Result Grid" showing the results of the query. The grid has two columns: "Continent" and "city". The results are as follows:

Continent	city
Europe	Tirana
Europe	Andorra la Vella
Europe	Wien
Europe	Graz
Europe	Linz
Europe	Salzburg
Europe	Innsbruck

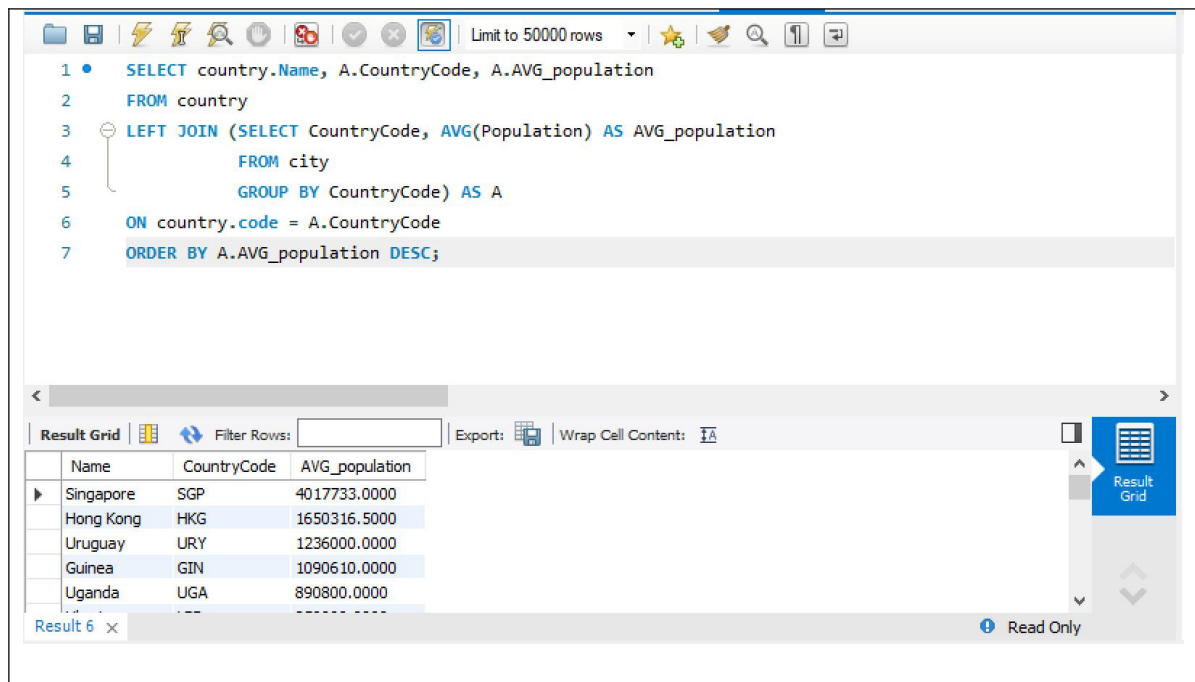
The interface also includes a toolbar with various icons, a "Limit to 50000 rows" dropdown, and a "Read Only" status indicator.

16. **Average Population by Country:** *Scenario:* A demographic research team is conducting a comparative analysis of population distributions across countries. You're tasked with calculating the average population for each country from the database to provide valuable insights into global population trends.

```

SELECT country.Name, A.CountryCode, A.AVG_population
FROM country
LEFT JOIN (SELECT CountryCode, AVG(Population) AS AVG_population
           FROM city
           GROUP BY CountryCode) AS A
ON country.code = A.CountryCode
ORDER BY A.AVG_population DESC;

```



```

1 SELECT country.Name, A.CountryCode, A.AVG_population
2 FROM country
3 LEFT JOIN (SELECT CountryCode, AVG(Population) AS AVG_population
4            FROM city
5            GROUP BY CountryCode) AS A
6 ON country.code = A.CountryCode
7 ORDER BY A.AVG_population DESC;

```

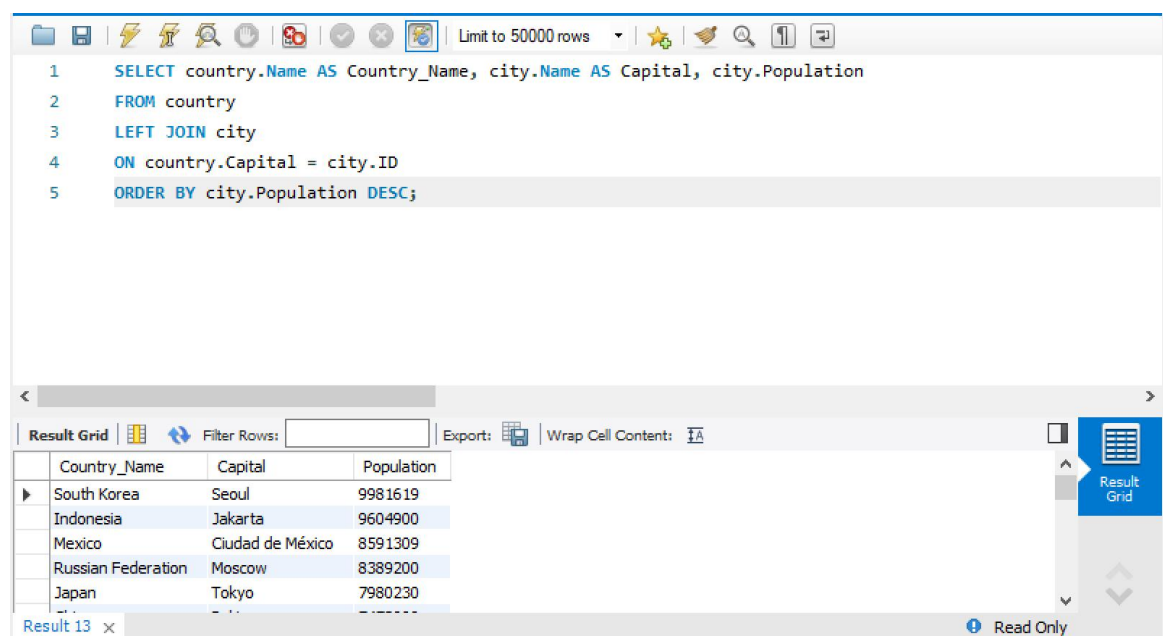
Name	CountryCode	AVG_population
Singapore	SGP	4017733.0000
Hong Kong	HKG	1650316.5000
Uruguay	URY	1236000.0000
Guinea	GIN	1090610.0000
Uganda	UGA	890800.0000

17. **Capital Cities Population Comparison:** *Scenario:* A statistical analysis firm is examining population distributions between capital cities worldwide. You're tasked with comparing the populations of capital cities from different countries to identify trends and patterns in urban demographics.

```

SELECT country.Name AS Country_Name, city.Name AS Capital, city.Population
FROM country
LEFT JOIN city
ON country.Capital = city.ID
ORDER BY city.Population DESC;

```



```

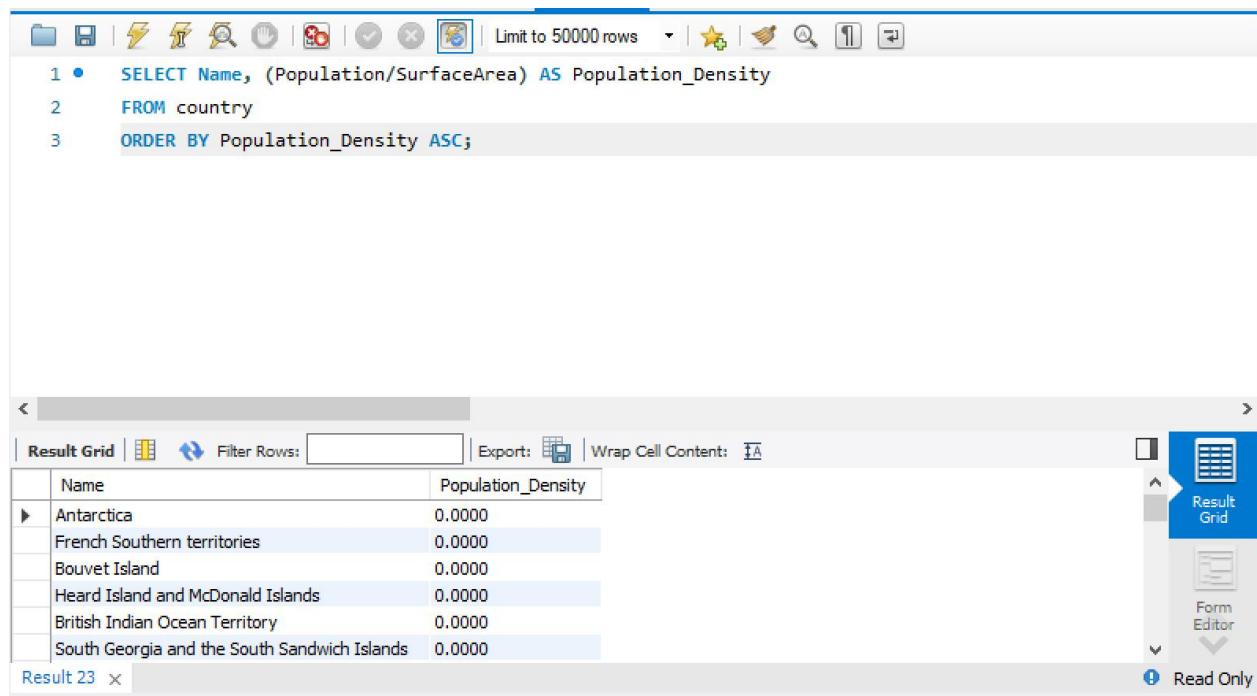
1 SELECT country.Name AS Country_Name, city.Name AS Capital, city.Population
2 FROM country
3 LEFT JOIN city
4 ON country.Capital = city.ID
5 ORDER BY city.Population DESC;

```

Country_Name	Capital	Population
South Korea	Seoul	9981619
Indonesia	Jakarta	9604900
Mexico	Ciudad de México	8591309
Russian Federation	Moscow	8389200
Japan	Tokyo	7980230

18. **Countries with Low Population Density:** *Scenario:* An agricultural research institute is studying countries with low population densities for potential agricultural development projects. You're tasked with identifying countries with sparse populations from the database to support the institute's research efforts.

```
SELECT Name, (Population/SurfaceArea) AS Population_Density
FROM country
ORDER BY Population_Density ASC;
```



The screenshot shows a database query tool interface. The SQL query is entered in the top text area, and the results are displayed in a table below. The table has two columns: 'Name' and 'Population_Density'. The results show several countries with a population density of 0.0000, ordered from top to bottom: Antarctica, French Southern territories, Bouvet Island, Heard Island and McDonald Islands, British Indian Ocean Territory, and South Georgia and the South Sandwich Islands. The interface includes a toolbar with various icons, a 'Limit to 50000 rows' dropdown, and a 'Result Grid' button on the right.

Name	Population_Density
Antarctica	0.0000
French Southern territories	0.0000
Bouvet Island	0.0000
Heard Island and McDonald Islands	0.0000
British Indian Ocean Territory	0.0000
South Georgia and the South Sandwich Islands	0.0000

19. **Cities with High GDP per Capita:** *Scenario:* An economic consulting firm is analysing cities with high GDP per capita for investment opportunities. You're tasked with identifying cities with above-average GDP per capita from the database to assist the firm in identifying potential investment destinations.

```
SELECT city.Name, A.GDP, A.AVG_GDP
FROM (SELECT Name, capital, (GNP/Population) AS GDP,
             (SELECT AVG(GNP/Population) FROM country) AS AVG_GDP
      FROM country
      WHERE (GNP/Population) > (SELECT AVG(GNP/Population) FROM
country)) AS A
LEFT JOIN city
ON A.Capital = city.ID
ORDER BY A.GDP DESC;
```

The screenshot shows a database query editor with a SQL query and its results. The query is as follows:

```

8
9 • SELECT city.Name, A.GDP, A.AVG_GDP
10 FROM (SELECT Name, capital, (GNP/Population) AS GDP,
11       (SELECT AVG(GNP/Population) FROM country) AS AVG_GDP
12       FROM country
13       WHERE (GNP/Population) > (SELECT AVG(GNP/Population) FROM country)) AS A
14 LEFT JOIN city
15 ON A.Capital = city.ID
16 ORDER BY A.GDP DESC;
17

```

The results are displayed in a table with the following columns: Name, GDP, and AVG_GDP. The table contains 6 rows of data:

Name	GDP	AVG_GDP
Luxembourg [Luxemburg/Lëtzebuerg]	0.037459	0.0060896850
Bern	0.036936	0.0060896850
Hamilton	0.035815	0.0060896850
Bandar Seri Begawan	0.035686	0.0060896850
Vaduz	0.034644	0.0060896850
George Town	0.033237	0.0060896850

The interface includes a toolbar at the top with various icons, a "Limit to 50000 rows" dropdown, and a "Result Grid" button on the right. The bottom status bar shows "Result 18" and "Read Only".

20. **Display Columns with Limit (Rows 31-40):** *Scenario:* A market research firm requires detailed information on cities beyond the top rankings for a comprehensive analysis. You're tasked with providing data on cities ranked between 31st and 40th by population to ensure a thorough understanding of urban demographics.

```

SELECT Name, Population
FROM city
ORDER BY population DESC
LIMIT 10 OFFSET 30;

```

The screenshot shows a database query editor with a SQL query and its results. The query is as follows:

```

1 • SELECT Name, Population
2 FROM city
3 ORDER BY population DESC
4 LIMIT 10 OFFSET 30;

```

The results are displayed in a table with the following columns: Name and Population. The table contains 6 rows of data:

Name	Population
Shenyang	4265200
Kanton [Guangzhou]	4256300
Singapore	4017733
Ho Chi Minh City	3980000
Chennai (Madras)	3841396
Pusan	3804522

The interface includes a toolbar at the top with various icons, a "Limit to 50000 rows" dropdown, and a "Result Grid" button on the right. The bottom status bar shows "city 20" and "Read Only".

Course Notes

It is recommended to take notes from the course, use the space below to do so, or use the revision guide shared with the class:



We have included a range of additional links to further resources and information that you may find useful, these can be found within your revision guide.

END OF WORKBOOK

Please check through your work thoroughly before submitting and update the table of contents if required.

Please send your completed work booklet to your trainer.

