

Data Technician

Name: Amnah Bibi

Course Date: 10/02/2025

Table of contents

Day 1: Task 1 3

Day 1: Task 2 5

Day 3: Task 1 6

Day 4: Task 1: Written 8

Day 4: Task 2: SQL Practical 12

Course Notes 20

Additional Information 21



Day 1: Task 1

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

What is a primary key?	A primary key is something that is unique for example an ID number. To find it easily, it will be automatically indexed. No two rows in a table can have the same primary key value, and it must always have a value (it cannot be NULL).
How does this differ from a secondary key?	A secondary key is similar, it is when columns need to be looked up often. They will be indexed so we can look them up faster. It is any key that is not the primary key but still has the potential to be used for accessing data in a database. For example, if ID number is the primary key, the secondary key can be first name, surname etc.
How are primary and foreign keys related?	A foreign key is a column or a set of columns in one table that links to the primary key on another table. It will also be the primary key of the other table (but not always as sometimes there is a unique key). The foreign key establishes a relationship between the two tables and ensures the referential integrity of the data, meaning no foreign key in one table can reference a non-existent record in a related table, for example table 1 should have 'CustomerNo3' but so should table 2 for a relationship to be made.
Provide a real-world example of a one-to-one relationship	<p>Occurs when a record (row) in one table is related to exactly one record in another table.</p> <p>E.g., A person and their passport (a person has one passport and a passport is assigned to only one person).</p>
Provide a real-world example of a one-to-many relationship	<p>Occurs when one record (row) in a table can be related to multiple records in another table.</p> <p>E.g., A customer and their orders (a customer can have multiple orders but each order is placed by exactly one customer).</p>



**Provide a
real-world
example of a
many-to-
many
relationship**

Occurs when multiple records in one table can be associated with multiple records in another table. This often requires a junction table to manage the relationship.
E.g., Students and courses (a student can enrol in multiple courses and a course can have multiple 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>The main difference is how they store and organise data. A relational database organises data into separate tables with rows and columns, allowing for flexible access through primary and foreign keys. They use ACID (Atomicity, Consistency, Isolation, Durability) properties to ensure that transactions are processed reliably and maintain integrity. These databases use SQL (Structured Query Language) for querying and managing the data. Examples are MySQL, Oracle and SQL Server.</p> <p>On the other hand, a non-relational database has data stored in a more flexible format such as documents, graphs etc. There are no rigid table structures. As they don't require a fixed schema, their schema-less structure means the format can vary between records. NoSQL databases are designed to scale horizontally, meaning they can handle large volumes of data. NoSQL databases use different query languages or APIs that are specific to their models (e.g., MongoDB, which is document based, uses a JSON-like query language).</p>
What type of data would benefit off the non-relational model? Why?	<ul style="list-style-type: none">• Unstructured or semi-structured data (images, videos, social media posts) because they don't need a rigid schema. This means you can easily store diverse, rapidly changing data without worrying about fitting it into a traditional relational table• Large volumes of data as they scale out horizontally across many servers, making them a better fit for large volumes of data• Flexible or evolving schemas as you don't have to strictly define the schema upfront which makes it easier to modify the structure as the requirements change over time without downtime or data migration• Real-time data as it can handle large amounts of incoming data <p>Non-relational databases are better for handling types of data that require flexibility, scalability and speed. They are more beneficial for when data is unstructured, semi-structured, rapidly</p>



evolving, when scalability is important or when performance under heavy load is required. They can handle different types of data (documents, graphs etc) without needing to transform the data into a specific format.

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	<p>A type of join where a table is joined with itself. It's typically used when you need to compare rows within the same table, or when a table has hierarchical data, such as parent-child relationships.</p> <p>For example, employee data where one employee (the manager) is also represented as an employee in the same table. This is useful for organizational hierarchies.</p>
Right join	<p>A right join returns all records from the right table and the matching records from the left table. If there's no match, the result will contain NULL values for columns from the left table.</p> <p>For example, data about Orders and Customers, where you want to make sure to list all customers, even those who haven't placed any orders yet.</p>
Full join	<p>A full join returns all records when there's a match in either the left or right table. If there is no match, NULL values are returned for columns from the table without a match.</p> <p>This is useful when you need to retrieve all records from both tables, even if there's no match between the data, like all students and all courses regardless of enrolment status.</p>
Inner join	<p>An inner join returns only the records where there is a common value between the two tables. If there's no match, the rows are not included in the result.</p>

	<p>This is ideal when you want to find related records in both tables, like customers who have made purchases, or students enrolled in a course.</p>
Cross join	<p>A cross join returns the Cartesian product of the two tables, meaning it returns every possible combination of rows from the left and right tables. This can result in a large number of rows if the tables are large.</p> <p>Used for creating all possible combinations, like when you want to list all product-store pairs or generate all possible combinations of two sets of data (e.g., product options and store locations).</p>
Left join	<p>A left join returns all records from the left table and the matching records from the right table. If there's no match, the result will contain NULL values for columns from the right table.</p> <p>This is useful for situations where you want all data from one table (e.g., all employees) and any matching data from another table (e.g., department assignments), including cases where there's no match.</p>



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
your 500-
word essay
here

Designing a Database for a Small Corner Shop:

Understanding the business requirements:

The main purpose of the database will be to store and organise data related to inventory, sales transactions and customer information. This will allow the business to streamline its operations, by helping them to track their records.

The primary user of the database would be the store manager/owner as they will need to monitor stock levels, generate sales reports and track customer information. They will also use it to analyse sales trends and customer behaviours to improve business decisions. Employees will use the database to check inventory, update stock levels and input sales data into the system during transactions.

The data types that would be used in this database are: Numeric (both integer and real), string, date/time and Boolean. The numeric data would be for item quantities, costs and points accrued. The string data would include products and names. Boolean data would be used to check if a customer is a loyalty member.

Designing the database schema:

Table 1: Inventory - The columns will be Product ID (which will be our primary key), Name, Category, Stock Quantity and Price.

Table 2: Customer Information - The columns will be Customer ID (which will be our primary key), Customer Name and Contact Information.

Table 3: Sales - The columns will be Sales ID (which will be our primary key), Sale Date, Customer ID (which will be our foreign key) and Total Amount.

Table 4: Sales Details - This will link our Sales and Inventory table. The columns will be SaleDetailsID (which is our primary key), SaleID (which is our foreign key linking to Sales), ProductID (which is our foreign key linking to Inventory), Quantity and Subtotal.

Table 5: Loyalty Program Table – The columns will be CustomerID (which is our primary key) IsMember (which is our Boolean and will give TRUE/FALSE options for whether or not the customer is a member of our loyalty program) and Points.

The relationship between the tables is as follows:

Sales and Customers (One-to-Many)

- A customer can make multiple sales over time.
- A sale is linked to one customer (or NULL for guest checkouts).



- Foreign Key: Sales.CustomerID → Customers.CustomerID

Sales and Sales_Details (One-to-Many)

- A sale can include multiple products.
- The Sales_Details table serves as the bridge between Sales and Products.
- Foreign Key: Sales_Details.SaleID → Sales.SaleID

Products and Sales_Details (One-to-Many)

- Each product can be part of multiple sales.
- The Sales_Details table tracks which products were sold in each transaction.
- Foreign Key: Sales_Details.ProductID → Products.ProductID

Customers and Loyalty_Program (One-to-One)

- A customer has one loyalty program account.
- Foreign Key: Loyalty_Program.CustomerID → Customers.CustomerID

Implementing the database:

To create the database and tables for a small supermarket with a loyalty program, you would first use the CREATE DATABASE command to set up the database. Then, use CREATE TABLE commands to define the Inventory, Customers, Sales, SalesDetails and Loyalty Program tables, specifying appropriate columns and relationships:

```
CREATE TABLE Inventory (ProductID INT PRIMARY KEY, Name
VARCHAR(100), Category VARCHAR(50), Price DECIMAL(10,2),
StockQuantity INT);
```

```
CREATE TABLE Customers (CustomerID INT PRIMARY KEY, Name
VARCHAR(100), ContactInfo VARCHAR(255));
```

```
CREATE TABLE Sales (SaleID INT PRIMARY KEY, CustomerID INT,
SaleDate DATE, TotalAmount DECIMAL(10,2), FOREIGN KEY (ProductID)
REFERENCES Products(ProductID), FOREIGN KEY (CustomerID)
REFERENCES Customers(CustomerID));
```

```
CREATE TABLE LoyaltyProgram (CustomerID INT PRIMARY KEY,
IsMember BOOLEAN, Points INT, FOREIGN KEY (CustomerID)
REFERENCES Customers(CustomerID));
```

When tables have been created, data type can be declared. 'VARCHAR (255)' can be used for things like email. This will limit the characters length to 255. Also, values like price can be stored in decimal



'DECIMAL(10, 2)'. Here we are allowing for up to 10 digits, with 2 decimal places.

Populating the Database:

Once the database structure is in place, initial data needs to be inserted for the system to function properly, using SQL INSERT statements. For example,

```
INSERT INTO Inventory (ProductID, Name, Category, Stock Quantity, Price)
VALUES (1, Chocolate, 'Confectionery', 100, 2.50),
      (2, 'Croissants', 'Bakery', 50, 1.80),
      (3, 'Butter', 'Dairy', 75, 3.00);
```

```
INSERT INTO Customers (CustomerID, Customer Name, Contact Information)
VALUES (7, 'John Doe', '123-456-7890', 'johndoe@example.com'),
      (16, 'Jane Smith', '987-654-3210', 'janesmith@example.com');
```

```
INSERT INTO Sales (Customer_ID, Sale_ID, Sale_Date, Total_Amount)
VALUES (1, 7, '2025-02-13 10:15:00', 25.00);
```

```
INSERT INTO Sales_Details (Sales_DeatilsID, Sale_ID, Product_ID, Quantity,
Subtotal)
VALUES (9, 1, 7, 2, 5.00);
```

```
INSERT INTO LoyaltyProgram (CustomerID, IsMember Points)
VALUES (1, TRUE, 10);
```

Maintaining the database:

To ensure data accuracy and system reliability, we should:

- Take part in regular updates - Implement triggers/stored procedures to automatically adjust inventory levels in response to sales transactions. For example, for inventory management as items are sold, the stock quantities should be updated. Another important piece of information that should be updated is client details, i.e. those most active/still part of loyalty programme.
 - Backups - We should have regular backups to prevent data loss, this could be a scheduled daily back up.
 - Security Measures – It is crucial to implement user authentication and access controls to prevent changes being made from unknown parties and so that there is restricted access to sensitive information. Also, ensure customer data is encrypted and there are firewalls for protection.
-

Day 4: Task 2: SQL Practical

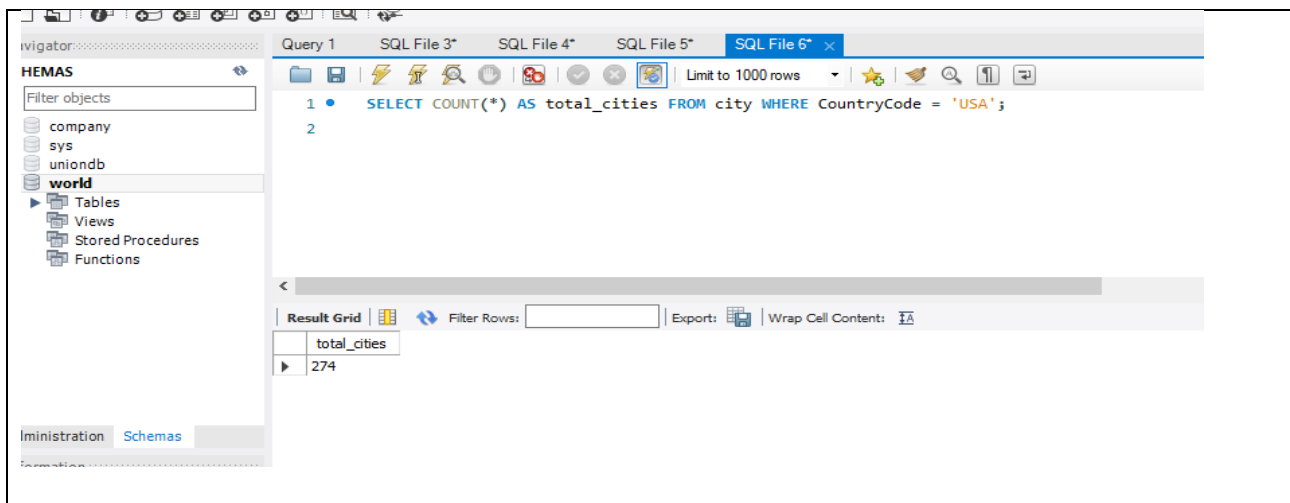
In your groups, work together to answer the below questions. It may be of benefit if one of you shares your screen with the group and as a team answer / take screen shots from there.

Setting up the database:

1. Download world_db(1) [here](#)
2. Follow each step to create your database [here](#)

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.



The screenshot shows a SQL IDE interface. On the left, a 'HEMAS' sidebar lists database objects: company, sys, uniondb, and world. The 'world' database is selected, showing a tree view with Tables, Views, Stored Procedures, and Functions. The main window displays 'Query 1' with the following SQL syntax:

```
1 • SELECT COUNT(*) AS total_cities FROM city WHERE CountryCode = 'USA';
```

The output is shown in a 'Result Grid' at the bottom, displaying a single row with the column 'total_cities' and the value '274'.

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.

The screenshot shows a SQL IDE interface with a schema browser on the left displaying the 'world' schema. The main query editor contains two SQL queries. The first query counts cities in the USA, and the second query lists countries by life expectancy. The result grid for the second query is displayed below the editor.

```

1 • SELECT COUNT(*) AS total_cities FROM city WHERE CountryCode = 'USA';
2
3 • SELECT Name, LifeExpectancy FROM country ORDER BY LifeExpectancy DESC;
4

```

Name	LifeExpectancy
Andorra	83.5
Macao	81.6
San Marino	81.1
Japan	80.7
Singapore	80.1
Australia	79.8

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.

The screenshot shows a SQL IDE interface with a schema browser on the left displaying the 'world' schema. The main query editor contains a SQL query that filters cities by name. The result grid for the query is displayed below the editor.

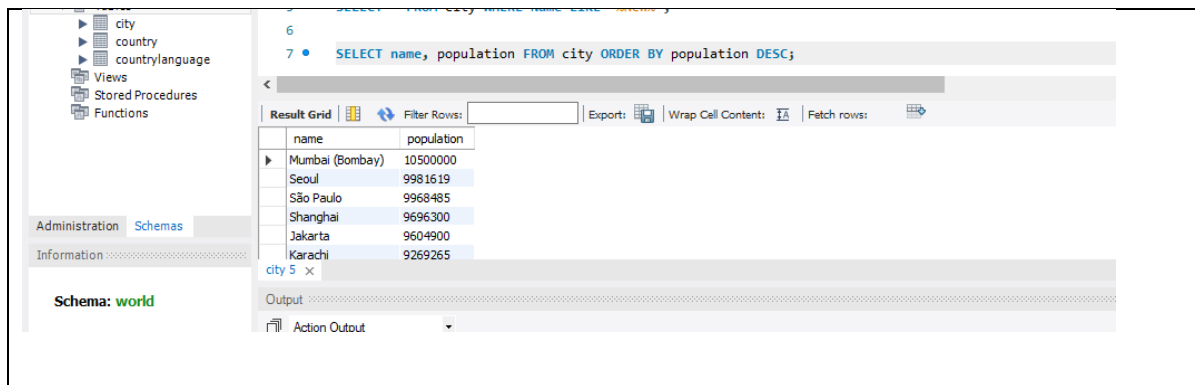
```

4
5 • SELECT * FROM city WHERE Name LIKE '%New%';
6

```

ID	Name	CountryCode	District	Population
137	Newcastle	AUS	New South Wales	270324
482	Newcastle upon Tyne	GBR	England	189150
502	Newport	GBR	Wales	139000
734	Newcastle	ZAF	KwaZulu-Natal	222993
936	Kowloon and New Kowloon	HKG	Kowloon and New Kowl	1987996
1106	New Bombay	IND	Maharashtra	307297

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.



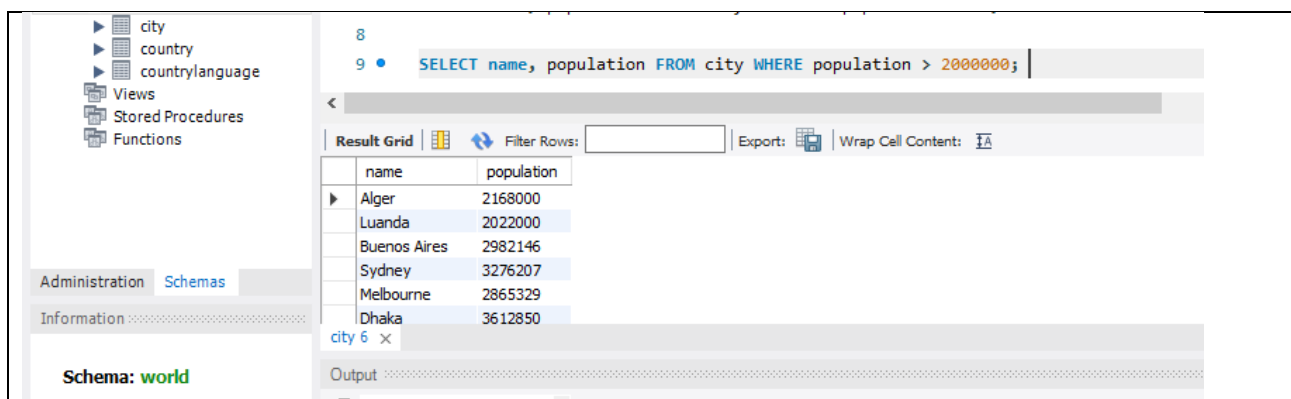
The screenshot shows a database interface with a left sidebar containing a tree view of database objects (city, country, countrylanguage, Views, Stored Procedures, Functions) and a bottom status bar indicating the 'Schema: world'. The main query editor displays the following SQL query:

```
6
7 • SELECT name, population FROM city ORDER BY population DESC;
```

The 'Result Grid' shows the following data:

name	population
Mumbai (Bombay)	10500000
Seoul	9981619
São Paulo	9968485
Shanghai	9696300
Jakarta	9604900
Karachi	9269265

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.



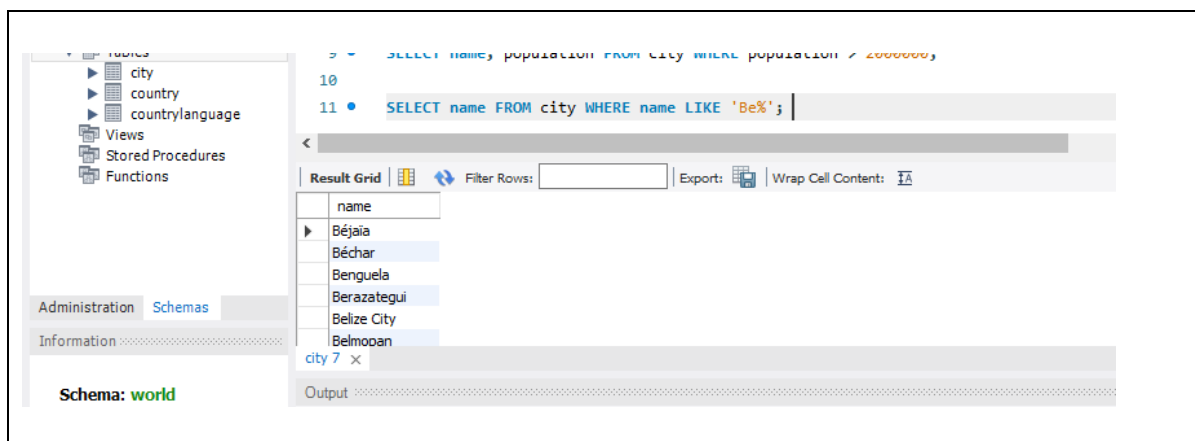
The screenshot shows the same database interface as before, but with a new query entered in the editor:

```
8
9 • SELECT name, population FROM city WHERE population > 2000000;
```

The 'Result Grid' shows the following data:

name	population
Alger	2168000
Luanda	2022000
Buenos Aires	2982146
Sydney	3276207
Melbourne	2865329
Dhaka	3612850

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.



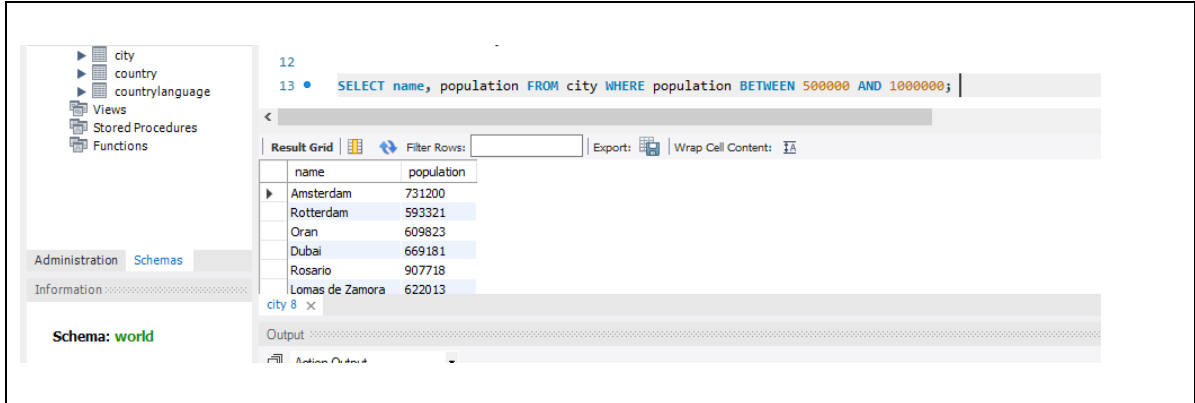
The screenshot shows the same database interface, with a new query entered in the editor:

```
10
11 • SELECT name FROM city WHERE name LIKE 'Be%';
```

The 'Result Grid' shows the following data:

name
Béjaia
Béchar
Benguela
Berazategui
Belize City
Belmopan

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.



The screenshot shows a database query interface. On the left, a tree view displays the database schema with folders for 'city', 'country', 'countrylanguage', 'Views', 'Stored Procedures', and 'Functions'. The 'city' folder is expanded, showing a table named 'city'. Below the tree, the 'Administration' tab is active, and the 'Schemas' section shows the 'world' schema selected. The main query editor displays the following SQL query:

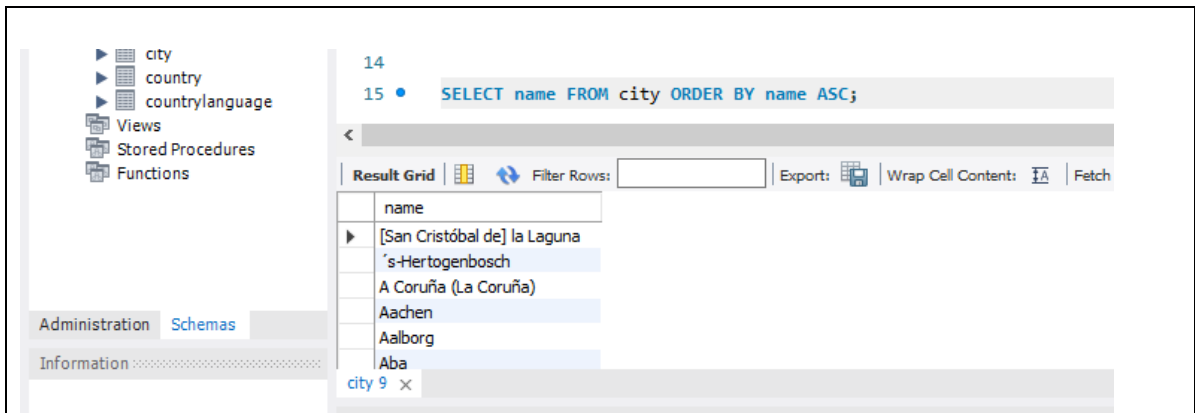
```
12  
13 • SELECT name, population FROM city WHERE population BETWEEN 500000 AND 1000000;
```

Below the query editor, the 'Result Grid' tab is active, showing a table with two columns: 'name' and 'population'. The table contains the following data:

name	population
Amsterdam	731200
Rotterdam	593321
Oran	609823
Dubai	669181
Rosario	907718
Lomas de Zamora	622013

The 'city 8' tab is selected at the bottom of the result grid.

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.



The screenshot shows a database query interface. On the left, a tree view displays the database schema with folders for 'city', 'country', 'countrylanguage', 'Views', 'Stored Procedures', and 'Functions'. The 'city' folder is expanded, showing a table named 'city'. Below the tree, the 'Administration' tab is active, and the 'Schemas' section shows the 'world' schema selected. The main query editor displays the following SQL query:

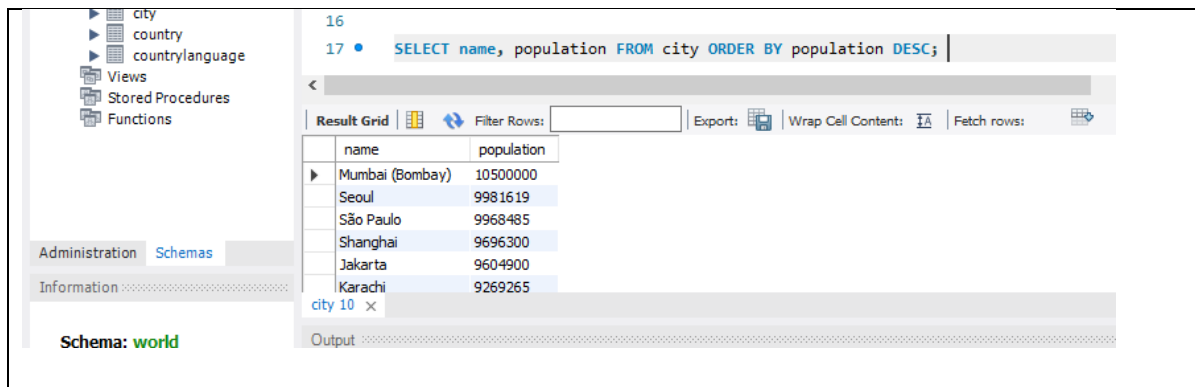
```
14  
15 • SELECT name FROM city ORDER BY name ASC;
```

Below the query editor, the 'Result Grid' tab is active, showing a table with one column: 'name'. The table contains the following data:

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

The 'city 9' tab is selected at the bottom of the result grid.

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.



The screenshot shows a database interface with a left sidebar containing a tree view of database objects: city, country, countrylanguage, Views, Stored Procedures, and Functions. The main area displays a query window with the following SQL statement:

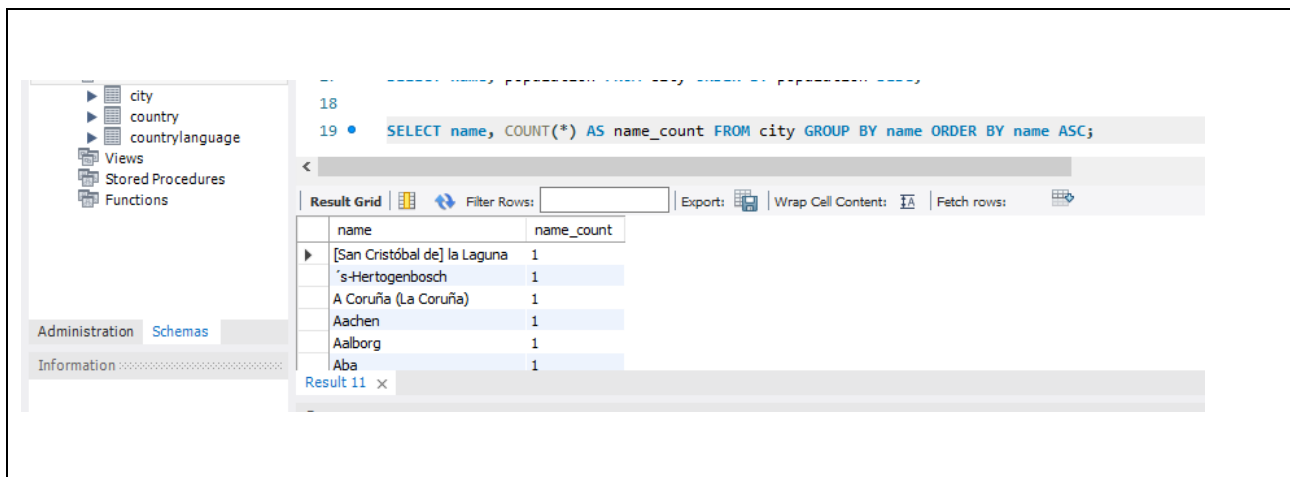
```
16
17 • SELECT name, population FROM city ORDER BY population DESC;
```

Below the query window is a 'Result Grid' showing the results of the query. The grid has two columns: 'name' and 'population'. The results are sorted in descending order of population.

name	population
Mumbai (Bombay)	10500000
Seoul	9981619
São Paulo	9968485
Shanghai	9696300
Jakarta	9604900
Karachi	9269265

The interface also shows a 'Schema: world' at the bottom left and an 'Output' section 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.



The screenshot shows a database interface with a left sidebar containing a tree view of database objects: city, country, countrylanguage, Views, Stored Procedures, and Functions. The main area displays a query window with the following SQL statement:

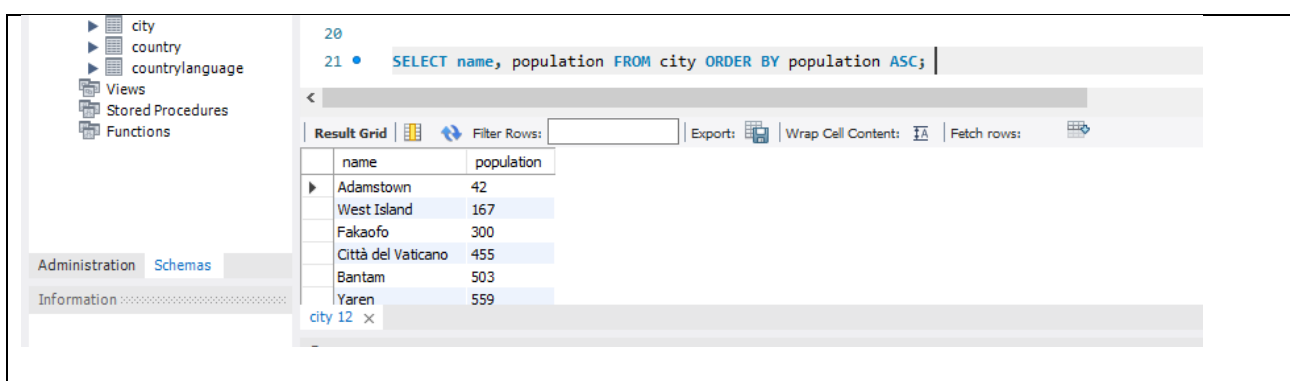
```
18
19 • SELECT name, COUNT(*) AS name_count FROM city GROUP BY name ORDER BY name ASC;
```

Below the query window is a 'Result Grid' showing the results of the query. The grid has two columns: 'name' and 'name_count'. The results are sorted in ascending order of city name.

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

The interface also shows a 'Schema: world' at the bottom left and an 'Output' section 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.



The screenshot shows a database interface with a left sidebar containing a tree view of database objects: city, country, countrylanguage, Views, Stored Procedures, and Functions. The main area displays a query window with the following SQL statement:

```
20
21 • SELECT name, population FROM city ORDER BY population ASC;
```

Below the query window is a 'Result Grid' showing the results of the query. The grid has two columns: 'name' and 'population'. The results are sorted in ascending order of population.

name	population
Adamstown	42
West Island	167
Fakaofu	300
Città del Vaticano	455
Bantam	503
Yaren	559

The interface also shows a 'Schema: world' at the bottom left and an 'Output' section at the bottom right.

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.

The screenshot shows a database management system interface. On the left, a tree view displays the database structure with folders for 'country', 'countrylanguage', 'Views', 'Stored Procedures', and 'Functions'. The 'country' folder is expanded. The main area shows a SQL query editor with the following query: `SELECT name, population FROM country ORDER BY population DESC;`. Below the query editor, a 'Result Grid' displays the results of the query. The results are as follows:

name	population
China	1277558000
India	1013662000
United States	278357000
Indonesia	212107000
Brazil	170115000
Pakistan	156483000

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.

The screenshot shows a database management system interface. On the left, a tree view displays the database structure with folders for 'IndepYear', 'Population', 'LifeExpectancy', 'GNP', 'GNPold', 'LocalName', 'GovernmentF', 'HeadOfState', 'Capital', and 'Code2'. The 'Capital' folder is expanded. The main area shows a SQL query editor with the following query: `SELECT city.name AS capital_city FROM country JOIN city ON country.capital = city.ID WHERE country.name = 'Spain';`. Below the query editor, a 'Result Grid' displays the results of the query. The results are as follows:

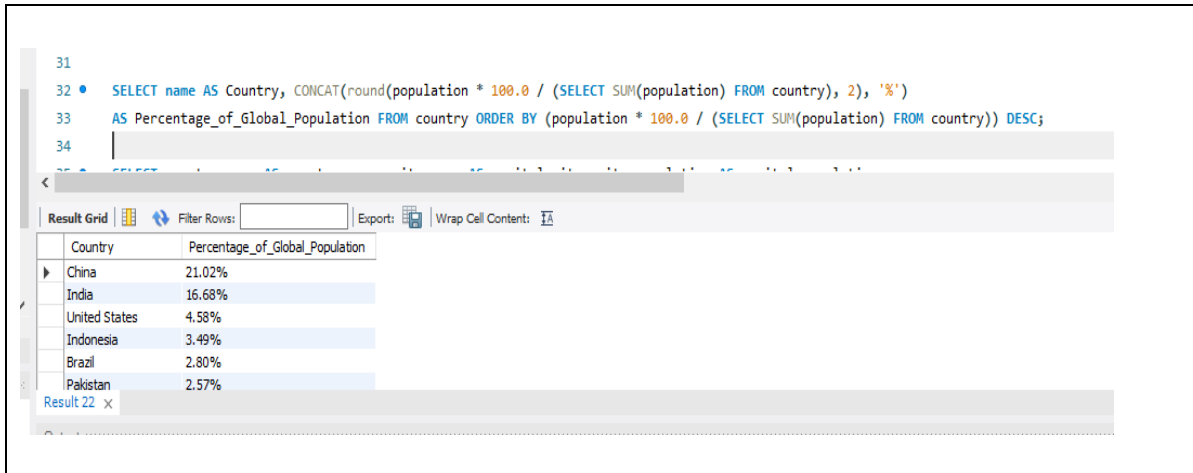
capital_city
Madrid

14. **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.

The screenshot shows a database management system interface. On the left, a tree view displays the database structure with folders for 'country', 'countrylanguage', 'Views', 'Stored Procedures', and 'Functions'. The 'country' folder is expanded. The main area shows a SQL query editor with the following query: `SELECT name FROM country WHERE continent = 'Europe';`. Below the query editor, a 'Result Grid' displays the results of the query. The results are as follows:

name
Albania
Andorra
Austria
Belgium
Bulgaria
Bosnia and Herzegovina

15. **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.



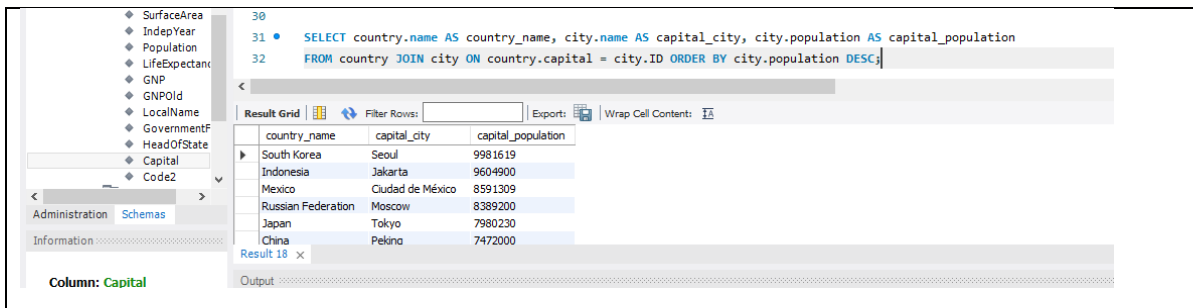
The screenshot shows a SQL query in a text editor and its corresponding result grid. The query calculates the percentage of the global population for each country, ordered by descending percentage.

```
31
32 • SELECT name AS Country, CONCAT(round(population * 100.0 / (SELECT SUM(population) FROM country), 2), '%')
33 AS Percentage_of_Global_Population FROM country ORDER BY (population * 100.0 / (SELECT SUM(population) FROM country)) DESC;
34
```

The result grid displays the following data:

Country	Percentage_of_Global_Population
China	21.02%
India	16.68%
United States	4.58%
Indonesia	3.49%
Brazil	2.80%
Pakistan	2.57%

16. **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.



The screenshot shows a SQL query in a text editor and its corresponding result grid. The query joins the country and city tables to compare capital city populations, ordered by descending population.

```
30
31 • SELECT country.name AS country_name, city.name AS capital_city, city.population AS capital_population
32 FROM country JOIN city ON country.capital = city.ID ORDER BY city.population DESC;
```

The result grid displays the following data:

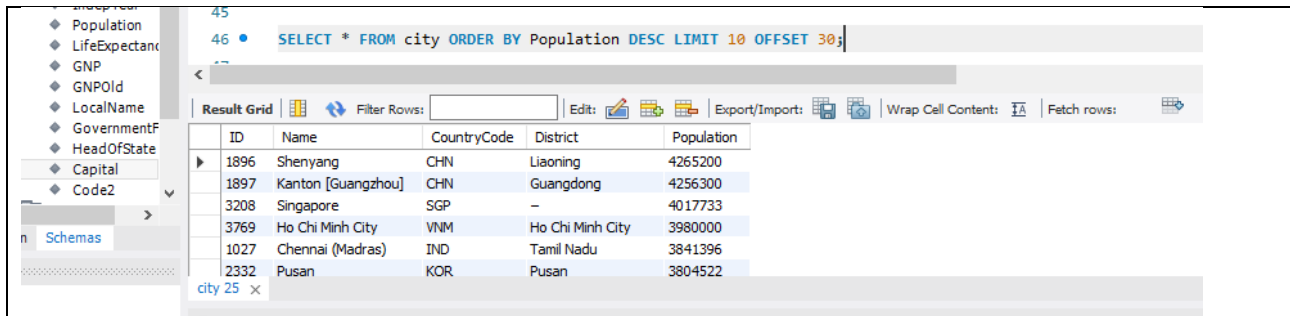
country_name	capital_city	capital_population
South Korea	Seoul	9981619
Indonesia	Jakarta	9604900
Mexico	Ciudad de México	8591309
Russian Federation	Moscow	8389200
Japan	Tokyo	7980230
China	Peking	7472000

17. **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.

18. **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.

no gdp data

19. **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.



The screenshot shows a database query interface. The SQL query entered is: `SELECT * FROM city ORDER BY Population DESC LIMIT 10 OFFSET 30;`. The results are displayed in a table with the following columns: ID, Name, CountryCode, District, and Population. The results show cities ranked between 31st and 40th by population.

ID	Name	CountryCode	District	Population
1896	Shenyang	CHN	Liaoning	4265200
1897	Kanton [Guangzhou]	CHN	Guangdong	4256300
3208	Singapore	SGP	-	4017733
3769	Ho Chi Minh City	VNM	Ho Chi Minh City	3980000
1027	Chennai (Madras)	IND	Tamil Nadu	3841396
2332	Pusan	KOR	Pusan	3804522



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.

