Data Technician

Name: Athiramol Padinjarekudiyil Sukumaran

Course Date: 17 August 2025 - 20 August 2025

Table of contents

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 a unique identifier for each record in a database table. • It must contain **unique values** (no duplicates). • It cannot be **NULL**. • Example: In a table of students, the *Student_ID* column can be the primary key because each student has a unique ID.
How does this differ from a secondary key?	A **secondary key** (also called an **alternate key** or **non-primary key**) is another column (or set of columns) that can be used to identify records but is **not the primary key**. • It may not always be unique. • It helps for **searching and indexing** data. **Example:** In the student table: • *Student_ID* = Primary Key • *Email* = Secondary Key (unique, but not the main identifier) • *Name* = Secondary Key (not unique but useful for searching).
How are primary and foreign keys related?	• A **foreign key** is a column in one table that refers to the **primary key** in another table. • This creates a **relationship** between the two tables and maintains **referential integrity**. **Example:** • Table: `Students` → *Student_ID* (Primary Key) • Table: `Enrollments` → *Student_ID* (Foreign Key referencing Students).

Provide a real-world example of a one-to-one relationship	A **BRP ID** and a **citizen**. • Each citizen has **one BRP ID**. • Each BRP ID belongs to **one citizen**.
Provide a real-world example of a one-to-many relationship	A **teacher** and **students**. • One teacher can have **many students**. • Each student has **one teacher** (in that class).
Provide a real-world example of a many-to-many relationship	**Students and Courses**. • A student can enroll in **many courses**. • A course can have **many students**. • This is usually represented by a **junction table** (e.g., *Enrollments*).

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?

Relational Database (SQL)	Non-Relational Database (NoSQL)
Stores data in **tables (rows & columns)**	Stores data in **flexible formats** like documents, key-value pairs, graphs, or wide-columns
Follows **schemas** (strict structure)	**Schema-less** (flexible structure)
Uses **SQL** for queries	Uses various query methods (JSON, APIs, custom languages)
Best for **structured data**	Best for **unstructured / semi-structured data**
Example: MySQL, PostgreSQL, Oracle	Example: MongoDB, Cassandra, Redis

What type of data would benefit off the non-relational model?

Why?

Unstructured or semi-structured data such as:

- Social media posts (text, images, videos).
- E-commerce product catalogs (different attributes per product).
- IoT sensor data.
- User profiles in an app (where fields may vary per user).

Because **non-relational databases are flexible** and can store data without needing a fixed schema.

They **scale easily** (handle big data with millions of users).

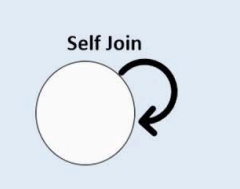
They are **faster for certain workloads** like retrieving documents, user activity, or logs.

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

A self-join is when a table is joined with itself. This is useful when you want to compare rows within the same table.



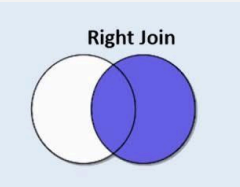
Example use case:

- Table: `Employees`
- Columns: `EmployeeID`, `Name`, `ManagerID`
- Use: To find which employees report to which managers (both are in the same table).

```
SELECT e1.Name AS Employee, e2.Name AS Manager
FROM Employees e1
LEFT JOIN Employees e2 ON e1.ManagerID = e2.EmployeeID;
```

Right join

Returns all rows from the **right table** and the matching rows from the left table. If there is no match, NULL is returned for the left table's columns.



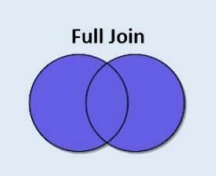
Example use case:

- Table1: `Orders`
- Table2: `Customers`
- Use: To get **all customers**, even those who haven't placed an order.

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
RIGHT JOIN Customers ON Orders.CustomerID =
Customers.CustomerID;
```

Full join

Returns all rows when there is a match in **either table**. Unmatched rows from either table are filled with NULL.



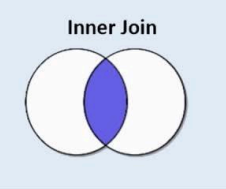
Example use case:

- Table1: `Employees`
- Table2: `Departments`
- Use: To see all employees and all departments, including employees without departments and departments without employees.

```
SELECT Employees.Name, Departments.DepartmentName
FROM Employees
FULL OUTER JOIN Departments ON Employees.DepartmentID =
Departments.DepartmentID;
```

Inner join

Returns only rows where there is a match in **both tables**.



Example use case:

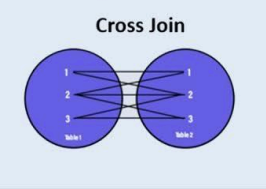
- Table1: `Orders`
- Table2: `Customers`

- Use: To get only orders that have a corresponding customer.

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID =
Customers.CustomerID;
```

Cross join

Returns the **Cartesian product** of both tables (every row from the first table combined with every row from the second table).



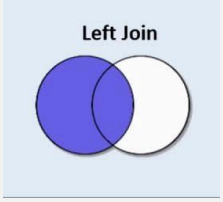
Example use case:

- Table1: `Shirts`
- Table2: `Colors`
- Use: To create all possible combinations of shirts and colors.

```
SELECT Shirts.Name, Colors.Color
FROM Shirts
CROSS JOIN Colors;
```

Left join

Returns all rows from the **left table** and matching rows from the right table. If there is no match, NULL is returned for the right table's columns.



Example use case:

- Table1: `Customers`

- Table2: `Orders`
- Use: To see all customers and their orders, including customers with no orders.

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID =
Orders.CustomerID;
```

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

Creating a Database System for ABC Retail

When setting up a database for a small retail business such as **ABC Retail**, the main objective is to streamline daily operations, improve data accuracy, and support decision-making.

This database will handle key functions such as inventory management, customer details, sales transactions, and loyalty program tracking. The following steps outline the process of building this system.

❖ Understanding the Business Requirements

The first step is to understand what kind of data **ABC Retail** needs to store. For inventory, the database should include product details such as product ID, name, category, supplier, cost price, selling price, and quantity in stock. For customers, it should capture customer ID, name, contact details, and loyalty points. For sales, the database must record transaction ID, date, customer ID, products purchased, quantities, and total amounts.

The main users of this database will be shop employees and the store owner. Employees will need to record sales quickly and check stock levels, while the owner will require reports on sales trends, profit margins, and customer loyalty performance.

❖ Designing the Database Schema

The database schema should be structured into multiple related tables. A possible design for **ABC Retail** could include:

- **Products Table:** Stores details about each product.
- **Customers Table:** Stores customer information and loyalty points.
- **Sales Table:** Records each transaction with a date and customer reference.

- **Sales_Items Table:** A junction table that links sales to the specific products purchased.

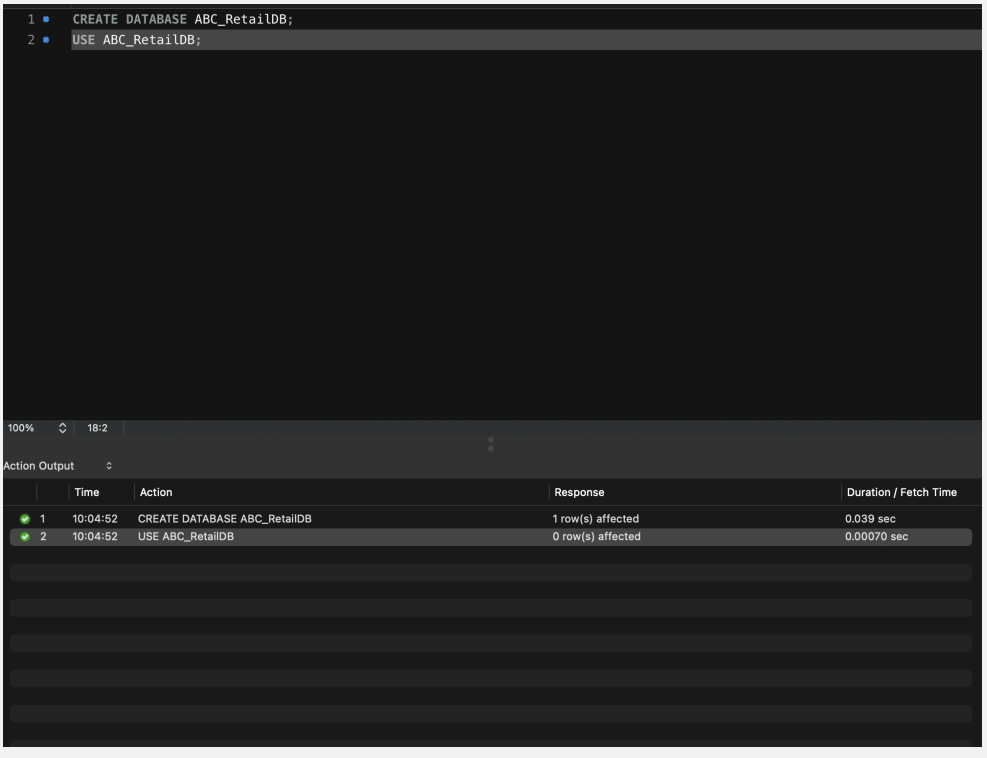
Relationships between these tables are essential. For example, each sale (Sales Table) may involve multiple products (linked through Sales_Items), and each sale is associated with one customer (Customers Table). Inventory levels will be updated based on the items sold.

❖ Implementing the Database

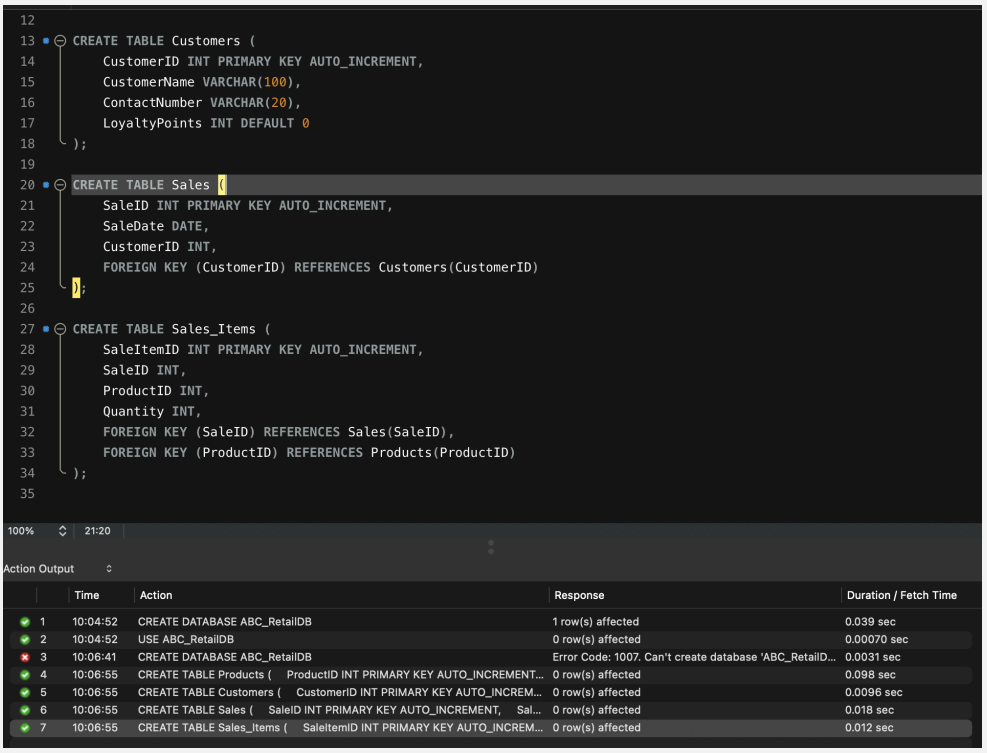
To implement the schema, SQL commands are used.

- First, the database itself is created:

```
CREATE DATABASE ABC_RetailDB;
USE ABC_RetailDB;
```



- Next, tables are created



```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY AUTO_INCREMENT,
    CustomerName VARCHAR(100),
    ContactNumber VARCHAR(20),
    LoyaltyPoints INT DEFAULT 0
);

CREATE TABLE Sales (
    SaleID INT PRIMARY KEY AUTO_INCREMENT,
    SaleDate DATE,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

CREATE TABLE Sales_Items (
    SaleItemID INT PRIMARY KEY AUTO_INCREMENT,
    SaleID INT,
    ProductID INT,
    Quantity INT,
    FOREIGN KEY (SaleID) REFERENCES Sales(SaleID),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
```

100% 21:20

Action Output

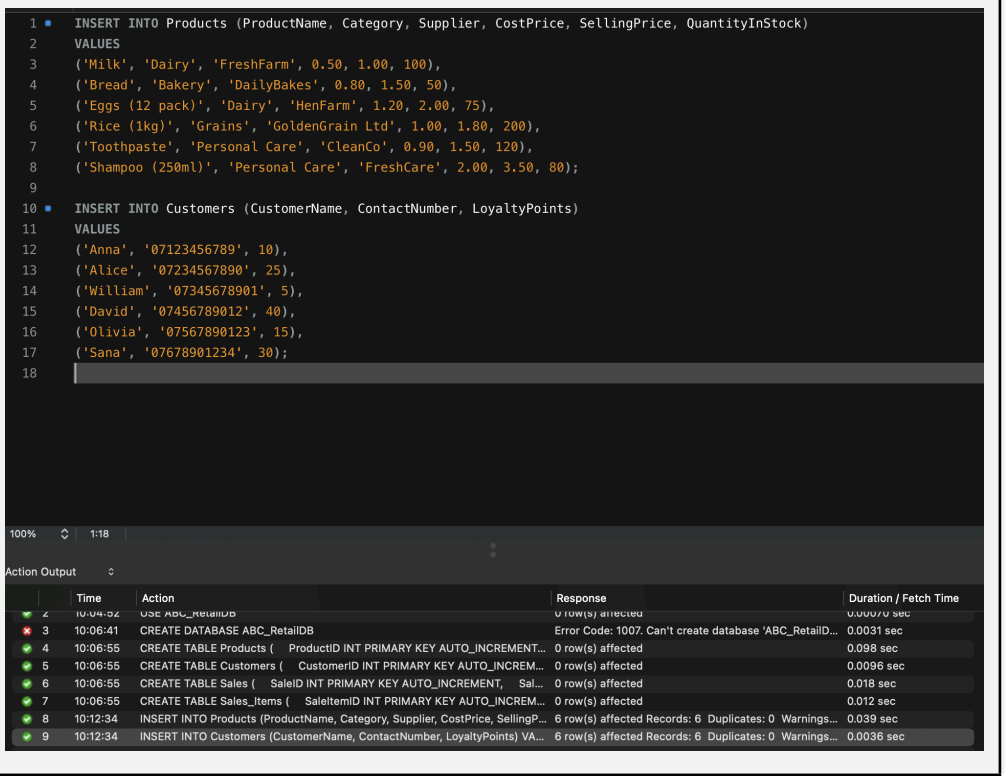
	Time	Action	Response	Duration / Fetch Time
1	10:04:52	CREATE DATABASE ABC_RetailDB	1 row(s) affected	0.039 sec
2	10:04:52	USE ABC_RetailDB	0 row(s) affected	0.00070 sec
3	10:06:41	CREATE DATABASE ABC_RetailDB	Error Code: 1007. Can't create database 'ABC_RetailD...	0.0031 sec
4	10:06:55	CREATE TABLE Products (ProductID INT PRIMARY KEY AUTO_INCREMENT...	0 row(s) affected	0.098 sec
5	10:06:55	CREATE TABLE Customers (CustomerID INT PRIMARY KEY AUTO_INCREM...	0 row(s) affected	0.0096 sec
6	10:06:55	CREATE TABLE Sales (SaleID INT PRIMARY KEY AUTO_INCREMENT, Sal...	0 row(s) affected	0.018 sec
7	10:06:55	CREATE TABLE Sales_Items (SaleItemID INT PRIMARY KEY AUTO_INCREM...	0 row(s) affected	0.012 sec

```
CREATE TABLE Products (
    ProductID INT PRIMARY KEY AUTO_INCREMENT,
    ProductName VARCHAR(100),
    Category VARCHAR(50),
    Supplier VARCHAR(100),
    CostPrice DECIMAL(10,2),
    SellingPrice DECIMAL(10,2),
    QuantityInStock INT
);

CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY AUTO_INCREMENT,
    CustomerName VARCHAR(100),
    ContactNumber VARCHAR(20),
    LoyaltyPoints INT DEFAULT 0
);

CREATE TABLE Sales (
    SaleID INT PRIMARY KEY AUTO_INCREMENT,
    SaleDate DATE,
    CustomerID INT,
```

```
    FOREIGN KEY (CustomerID) REFERENCES
Customers(CustomerID)
);

CREATE TABLE Sales_Items (
    SaleItemID INT PRIMARY KEY AUTO_INCREMENT,
    SaleID INT,
    ProductID INT,
    Quantity INT,
    FOREIGN KEY (SaleID) REFERENCES Sales(SaleID),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
```

- **Populating the Database**

Initial data for **ABC Retail** is added using **`INSERT`** statements. For example:



```sql
USE ABC_RetailDB;

INSERT INTO Sales (SaleDate, CustomerID) VALUES
('2025-08-10', 1),
('2025-08-10', 2),
('2025-08-11', 4),
('2025-08-12', 3),
('2025-08-12', 6),
('2025-08-13', 5);
```

100% 20:9

```sql
INSERT INTO Sales_Items (SaleID, ProductID, Quantity) VALUES
(1, 1, 2),
(1, 2, 1),
(2, 3, 1),
(2, 6, 1),
(2, 5, 2),
(3, 4, 2),
```

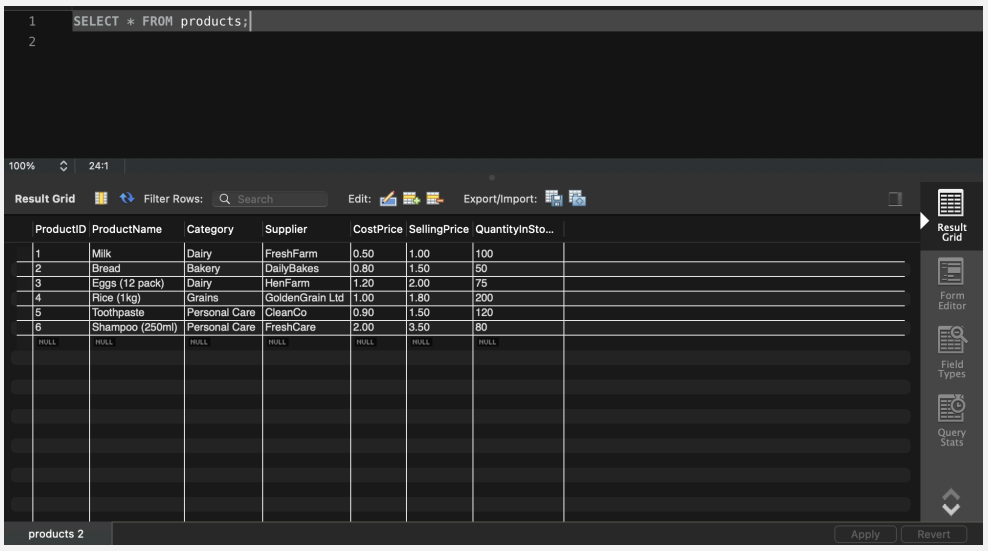
INSERT INTO Products (ProductName, Category, Supplier, CostPrice, SellingPrice, QuantityInStock)
VALUES
('Milk', 'Dairy', 'FreshFarm', 0.50, 1.00, 100),
('Bread', 'Bakery', 'DailyBakes', 0.80, 1.50, 50),
('Eggs (12 pack)', 'Dairy', 'HenFarm', 1.20, 2.00, 75),
('Rice (1kg)', 'Grains', 'GoldenGrain Ltd', 1.00, 1.80, 200),
('Toothpaste', 'Personal Care', 'CleanCo', 0.90, 1.50, 120),
('Shampoo (250ml)', 'Personal Care', 'FreshCare', 2.00, 3.50, 80);

INSERT INTO Customers (CustomerName, ContactNumber, LoyaltyPoints)
VALUES
('Anna', '07123456789', 10),
('Alice', '07234567890', 25),
('William', '07345678901', 5),
('David', '07456789012', 40),
('Olivia', '07567890123', 15),
('Sana', '07678901234', 30);

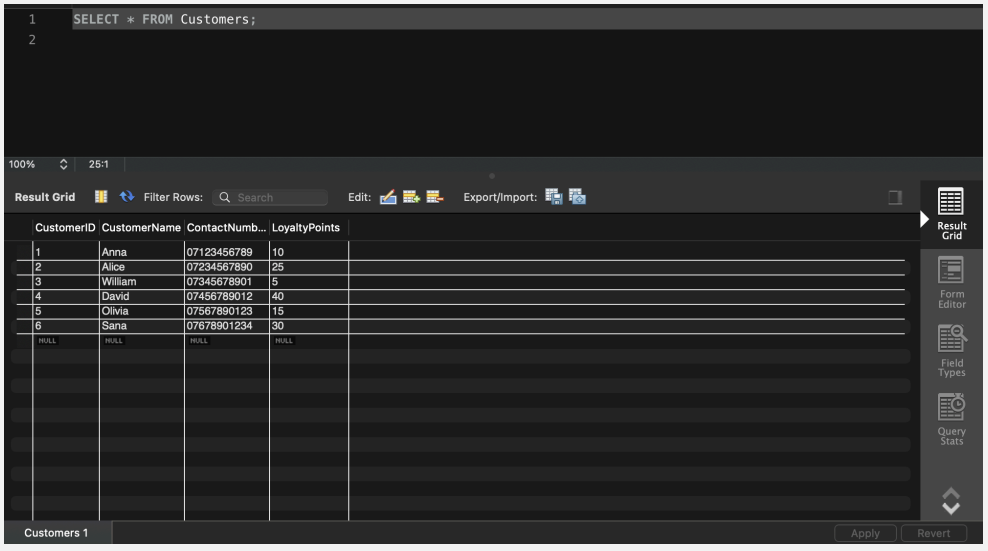
INSERT INTO Sales (SaleDate, CustomerID) VALUES
('2025-08-10', 1),
('2025-08-10', 2),
('2025-08-11', 4),
('2025-08-12', 3),

```
('2025-08-12', 6),
('2025-08-13', 5);

INSERT INTO Sales_Items (SaleID, ProductID, Quantity) VALUES
(1, 1, 2),
(1, 2, 1),
(2, 3, 1),
(2, 6, 1),
(2, 5, 2),
(3, 4, 2);
```

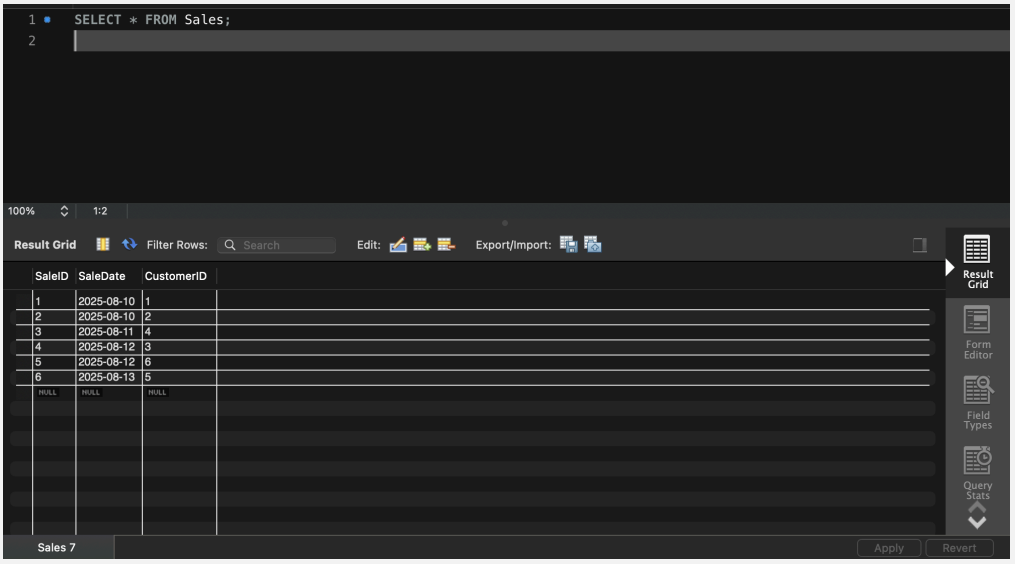
View the **Products** table



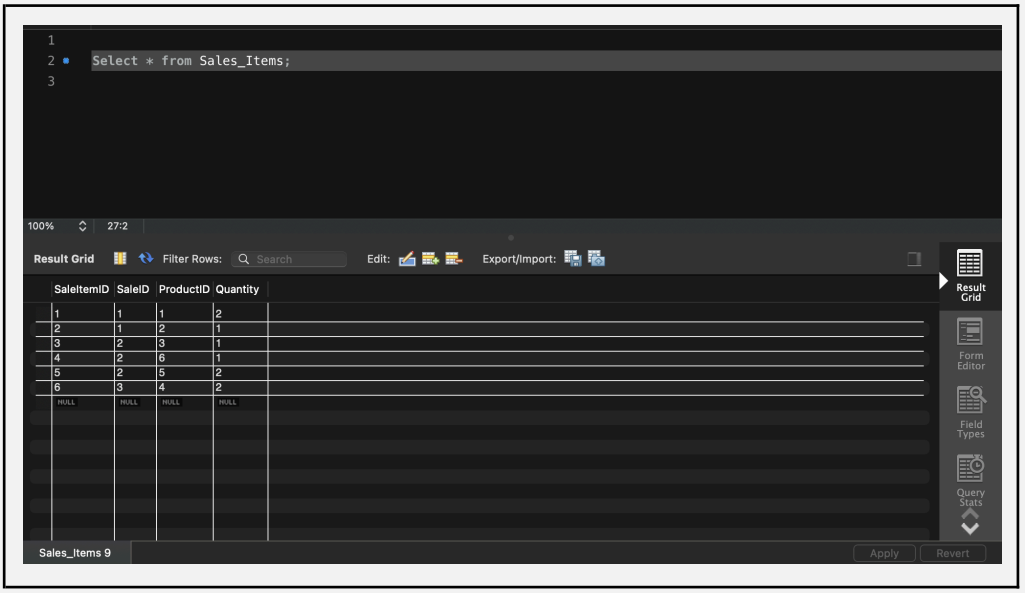
View the **Customers** table



View the **Sales** table



View the **Sales_Items** table



❖ Maintaining the Database

Once operational, the database must be maintained to ensure accuracy. This includes updating stock levels automatically after each sale, monitoring loyalty points, and validating customer information. Regular database backups should be scheduled to prevent data loss. Security measures, such as granting limited access rights to employees, will protect sensitive customer information.

Data integrity can also be maintained by using constraints (e.g., preventing negative stock levels) and by regularly running queries to identify inconsistencies.

Conclusion

By carefully analyzing requirements, designing a clear schema, implementing tables with SQL, and maintaining the system with proper data management practices, **ABC Retail** will benefit from an efficient database. This system will not only streamline sales and inventory tracking but also enhance customer engagement through the loyalty program, ultimately improving business operations and growth.

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.

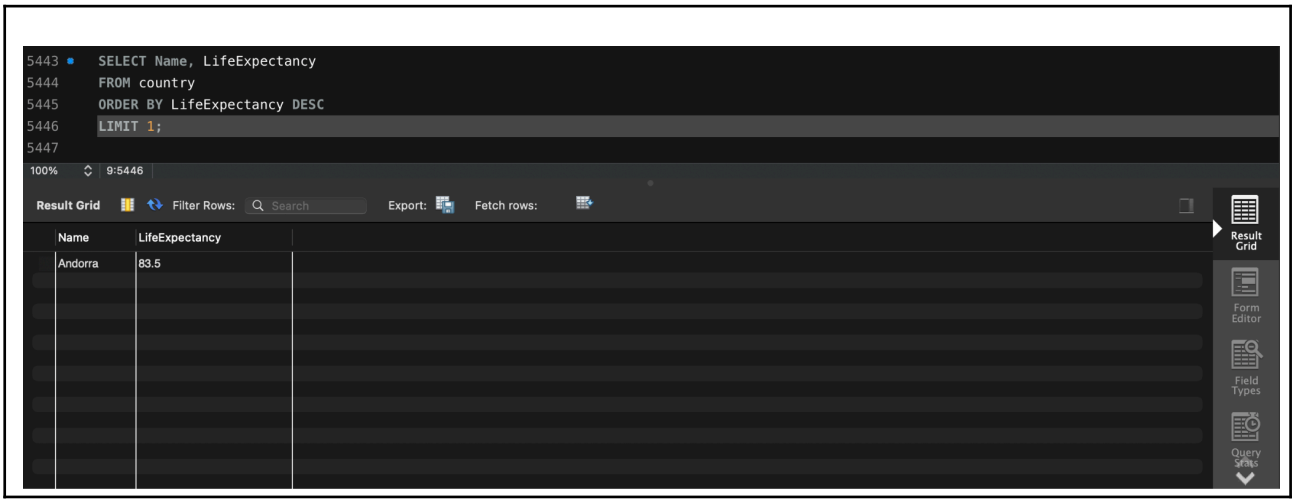
Setting up the database:

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.

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.



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.

```
5447
5448
5449 •  SELECT Name, CountryCode
5450    FROM city
5451    WHERE Name LIKE '%New%';
```

100% 25:5451

Result Grid | Filter Rows: Search | Export:

Name	CountryCode
Newcastle	AUS
Newcastle upon Tyne	GBR
Newport	GBR
Newcastle	ZAF
Kowloon and New Kowloon	HKG
New Bombay	IND
New Delhi	IND
Khanewal	PAK
New York	USA
New Orleans	USA
Newark	USA
Newport News	USA
New Haven	USA
New Bedford	USA

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.

```
5453 •  SELECT Name, CountryCode, Population
5454    FROM city
5455    ORDER BY Population DESC
5456    LIMIT 10;
5457
```

100% 10:5456

Result Grid | Filter Rows: Search | Export: | Fetch rows:

Name	CountryCode	Population
Mumbai (Bombay)	IND	10500000
Seoul	KOR	9981619
São Paulo	BRA	9968485
Shanghai	CHN	9696300
Jakarta	IDN	9604900
Karachi	PAK	9269265
Istanbul	TUR	8787958
Ciudad de México	MEX	8591309
Moscow	RUS	8389200
New York	USA	8008278

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.

```
5458 •  SELECT Name, CountryCode, Population
5459    FROM city
5460    WHERE Population > 2000000
5461    ORDER BY Population DESC;
5462
```

100% 26:5461

Result Grid | Filter Rows: Search | Export:

Name	CountryCode	Population
Mumbai (Bombay)	IND	10500000
Seoul	KOR	9981619
São Paulo	BRA	9968485
Shanghai	CHN	9696300
Jakarta	IDN	9604900
Karachi	PAK	9269265
Istanbul	TUR	8787958
Ciudad de México	MEX	8591309
Moscow	RUS	8389200
New York	USA	8008278
Tokyo	JPN	7980230
Peking	CHN	7472000
London	GBR	7285000
Delhi	IND	7206704
Cairo	EGY	6789479
Teheran	IRN	6758845
Lima	PER	6464693
Chongqing	CHN	6351600
Bangkok	THA	6320174
Santafé de Bogotá	COL	6260862
Rio de Janeiro	BRA	5598953
Tianjin	CHN	5286800
Kinshasa	COD	5064000
Lahore	PAK	5063499
Santiago de Chile	CHL	4703954
St Petersburg	RUS	4694000
Calcutta [Kolkata]	IND	4399819
Wuhan	CHN	4344600
Baghdad	IRQ	4336000
Harbin	CHN	4289800
Shenyang	CHN	4265200
Kanton [Guangzhou]	CHN	4256300

Result Grid | Form Editor | Field Types | Query Stats | Execution Plan

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.

```
5463 •  SELECT Name, CountryCode
5464    FROM city
5465    WHERE Name LIKE 'Be%';
```

100% 23:5465

Result Grid | Filter Rows: Search | Export:

Name	CountryCode
Béjaïa	DZA
Béchar	DZA
Benguela	AGO
Berazategui	ARG
Belize City	BLZ
Belmopan	BLZ
Belo Horizonte	BRA
Belém	BRA
Belford Roxo	BRA
Betim	BRA
Bento Gonçalves	BRA
Belfast	GBR
Benoni	ZAF
Bekasi	IDN
Bengkulu	IDN
Belgaum	IND
Bellary	IND
Berhampore (Baharampur)	IND
Beawar	IND
Bettiah	IND
Beerseba	ISR
Bene Beraq	ISR
Bergamo	ITA
Beppu	JPN
Beograd	YUG
Benxi	CHN
Bengbu	CHN
Bei´an	CHN
Beipiao	CHN
Beihai	CHN
Bello	COL
Beirut	LBN

Result Grid | Form Editor | Field Types | Query Stats | Execution Plan

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.

```
5467 •  SELECT Name, CountryCode, Population
5468    FROM city
5469    WHERE Population BETWEEN 500000 AND 1000000
5470    ORDER BY Population DESC;
```

100% 26:5470

Result Grid | Filter Rows: Search | Export:

Name	CountryCode	Population
Amman	JOR	1000000
Mogadishu	SOM	997000
Volgograd	RUS	993400
Sendai	JPN	989975
Peshawar	PAK	988005
Baotou	CHN	980000
Adelaide	AUS	978100
Madurai	IND	977856
Mekka	SAU	965700
Köln	DEU	962507
Managua	NIC	959000
Detroit	USA	951270
Shenzhen	CHN	950500
Haora (Howrah)	IND	950435
Campinas	BRA	950043
Brazzaville	COG	950000
Khartum	SDN	947483
Karaj	IRN	940968
Taichung	TWN	940589
Santa Cruz de la Sierra	BOL	935361
Varanasi (Benares)	IND	929270
Patna	IND	917243
Hohhot	CHN	916700
Rosario	ARG	907718
Voronez	RUS	907700
Soweto	ZAF	904165
Torino	ITA	903705
San Jose	USA	894943
Srinagar	IND	892506
Agra	IND	891790
Kampala	UGA	890800
Mandalay	MMR	885300

Result Grid | Form Editor | Field Types | Query Stats | Execution Plan

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.

```
5472 •  SELECT Name, CountryCode
5473    FROM city
5474    ORDER BY Name ASC;
```

100% 19:5474

Result Grid | Filter Rows: Search | Export: | Fetch rows:

Name	CountryCode
[San Cristóbal de] la Laguna	ESP
's-Hertogenbosch	NLD
A Coruña (La Coruña)	ESP
Aachen	DEU
Aalborg	DNK
Aba	NGA
Abadan	IRN
Abaetetuba	BRA
Abakan	RUS
Abbotsford	CAN
Abeokuta	NGA
Aberdeen	GBR
Abha	SAU
Abidjan	CIV
Abiko	JPN
Abilene	USA
Abohar	IND
Abottabad	PAK
Abu Dhabi	ARE
Abuja	NGA
Acámbaro	MEX
Acapulco de Juárez	MEX
Acarigua	VEN
Accra	GHA
Achalpur	IND
Acheng	CHN
Acuña	MEX
Adamstown	PCN
Adana	TUR
Addis Abeba	ETH
Adelaide	AUS
Aden	YEM

Result Grid | Form Editor | Field Types | Query Stats | Execution Plan

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.

```
5476
5477 •  SELECT Name, CountryCode, Population
5478    FROM city
5479    ORDER BY Population DESC
5480    LIMIT 1;
```

100% 9:5480

Result Grid | Filter Rows: Search | Export: | Fetch rows:

Name	CountryCode	Population
Mumbai (Bombay)	IND	10500000

Result Grid | Form Editor | Field Types | Query Stats | Execution Plan

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.

```
5481
5482 •  SELECT Name, COUNT(*) AS frequency
5483    FROM city
5484    GROUP BY Name
5485    ORDER BY Name ASC;
```

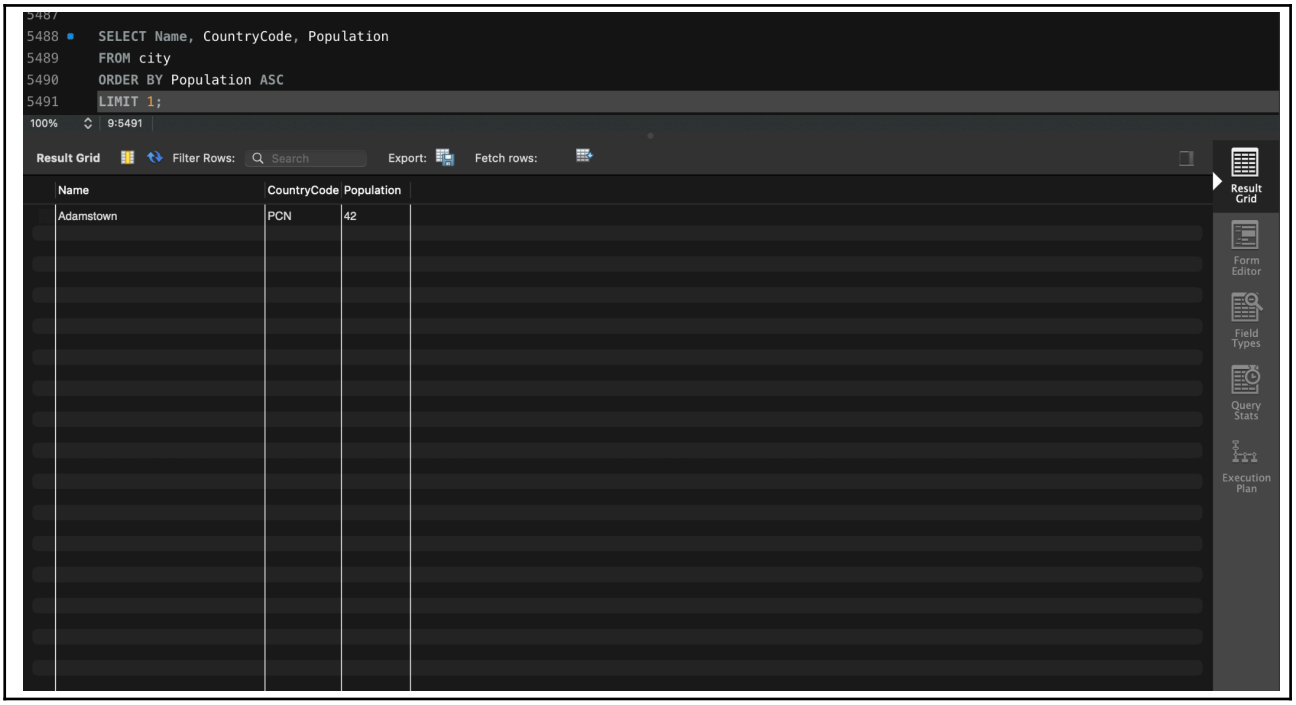
100% 19:5485

Result Grid | Filter Rows: Search | Export: | Fetch rows:

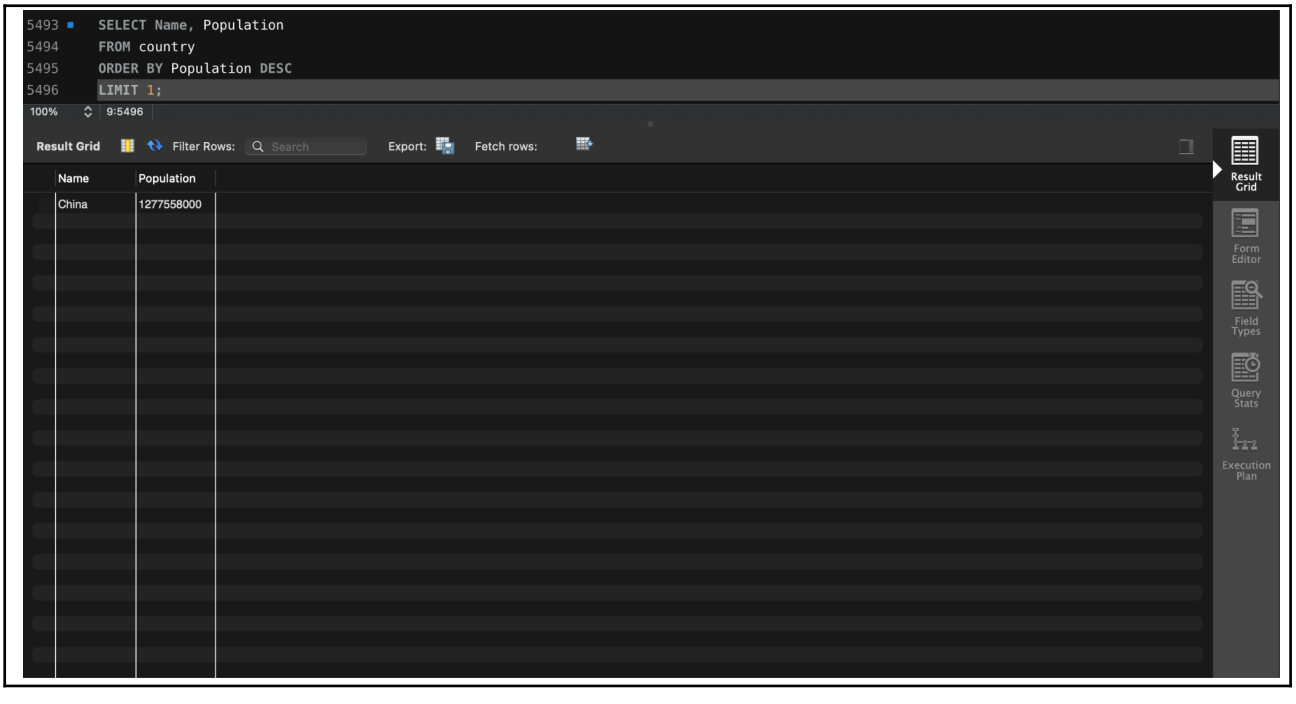
Name	frequency
[San Cristóbal de] la Laguna	1
´s-Hertogenbosch	1
A Coruña (La Coruña)	1
Aachen	1
Aalborg	1
Aba	1
Abadan	1
Abaetetuba	1
Abakan	1
Abbotsford	1
Abeokuta	1
Aberdeen	1
Abha	1
Abidjan	1
Abiko	1
Abilene	1
Abohar	1
Abottabad	1
Abu Dhabi	1
Abuja	1
Acámbaro	1
Acapulco de Juárez	1
Acarigua	1
Accra	1
Achalpur	1
Acheng	1
Acuña	1
Adamstown	1
Adana	1
Addis Abeba	1
Adelaide	1
Aden	1

Result Grid | Form Editor | Field Types | Query Stats | Execution Plan

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.



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.



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.

```
5498 •  SELECT city.Name AS Capital
5499    FROM city
5500    JOIN country ON city.ID = country.Capital
5501    WHERE country.Name = 'Spain';
```

100% 30:5501

Result Grid | Filter Rows: Search | Export:

Capital
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.

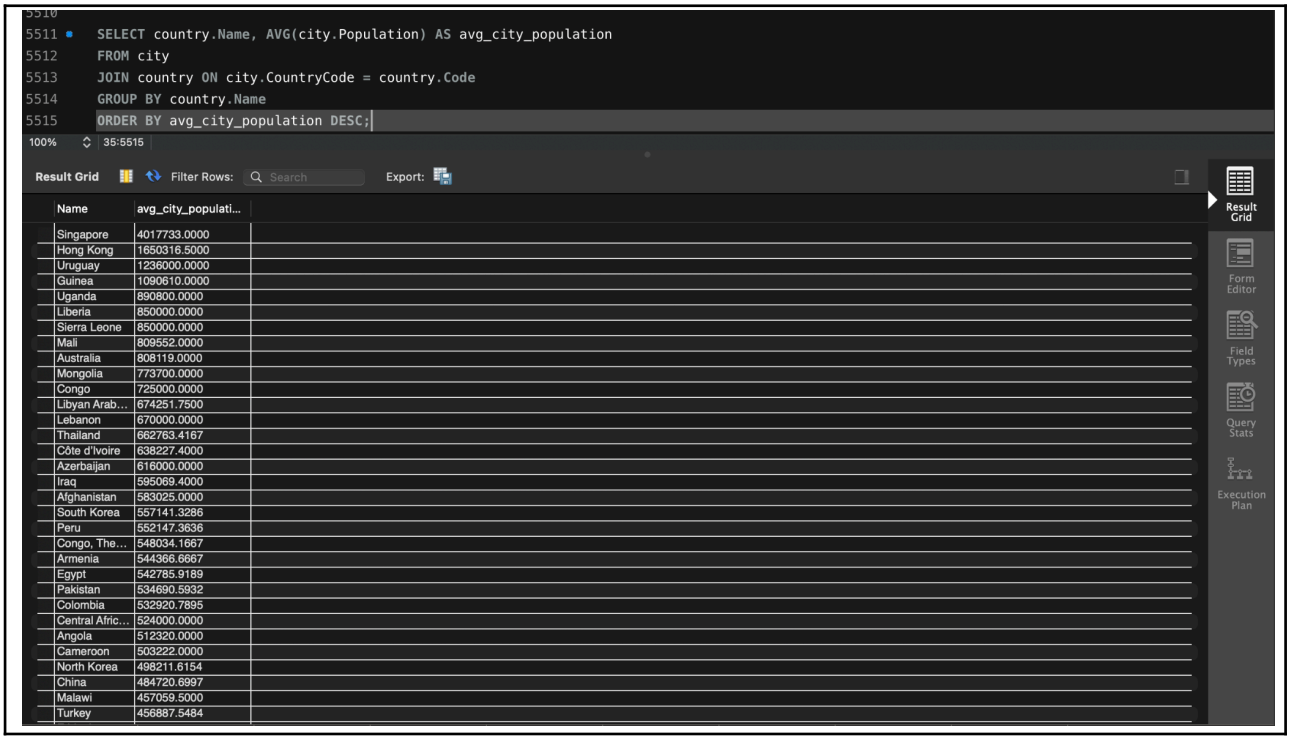
```
5505 •  SELECT city.Name, country.Name AS Country
5506    FROM city
5507    JOIN country ON city.CountryCode = country.Code
5508    WHERE country.Continent = 'Europe';
```

100% 36:5508

Result Grid | Filter Rows: Search | Export:

Name	Country
Amsterdam	Netherlands
Rotterdam	Netherlands
Haag	Netherlands
Utrecht	Netherlands
Eindhoven	Netherlands
Tilburg	Netherlands
Groningen	Netherlands
Breda	Netherlands
Apeldoorn	Netherlands
Nijmegen	Netherlands
Enschede	Netherlands
Haarlem	Netherlands
Almere	Netherlands
Arnhem	Netherlands
Zaanstad	Netherlands
's-Hertogenbosch	Netherlands
Amersfoort	Netherlands
Maastricht	Netherlands
Dordrecht	Netherlands
Leiden	Netherlands
Haarlemmermeer	Netherlands
Zoetermeer	Netherlands
Emmen	Netherlands
Zwolle	Netherlands
Ede	Netherlands
Delft	Netherlands
Heerlen	Netherlands
Alkmaar	Netherlands
Tirana	Albania
Andorra la Vella	Andorra
Antwerpen	Belgium

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.



```
5511  SELECT country.Name, AVG(city.Population) AS avg_city_population
5512  FROM city
5513  JOIN country ON city.CountryCode = country.Code
5514  GROUP BY country.Name
5515  ORDER BY avg_city_population DESC;
```

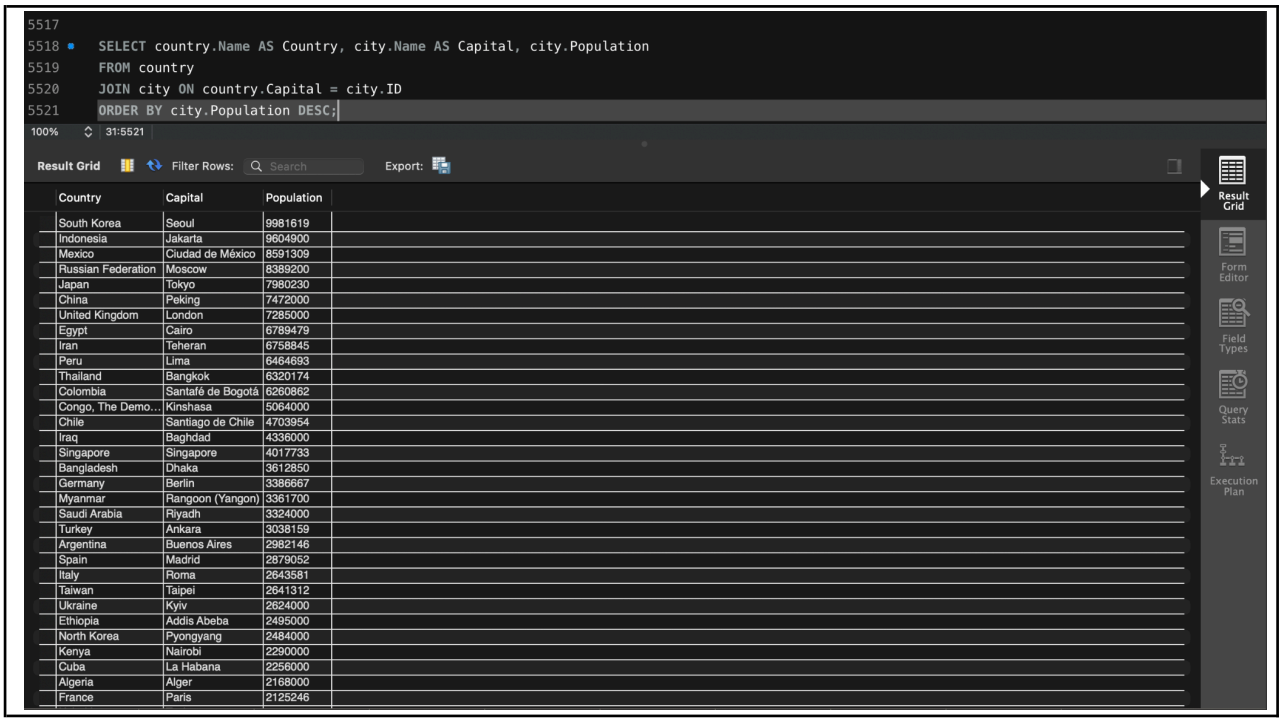
100% 35:5515

Result Grid Filter Rows: Search Export:

Name	avg_city_populati...
Singapore	4017733.0000
Hong Kong	1650316.5000
Uruguay	1236000.0000
Guinea	1090610.0000
Uganda	890800.0000
Liberia	850000.0000
Sierra Leone	850000.0000
Mali	809552.0000
Australia	808119.0000
Mongolia	773700.0000
Congo	725000.0000
Libyan Arab...	674251.7500
Lebanon	670000.0000
Thailand	662763.4167
Côte d'Ivoire	638227.4000
Azerbaijan	616000.0000
Iraq	595069.4000
Afghanistan	583025.0000
South Korea	557141.3286
Peru	552147.3636
Congo, The...	548034.1667
Armenia	544366.6667
Egypt	542785.9189
Pakistan	534690.5932
Colombia	532920.7895
Central Afric...	524000.0000
Angola	512320.0000
Cameroon	503222.0000
North Korea	498211.6154
China	484720.6997
Malawi	457059.5000
Turkey	456887.5484

Result Grid, Form Editor, Field Types, Query Stats, Execution Plan

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.



```
5517
5518  SELECT country.Name AS Country, city.Name AS Capital, city.Population
5519  FROM country
5520  JOIN city ON country.Capital = city.ID
5521  ORDER BY city.Population DESC;
```

100% 31:5521

Result Grid Filter Rows: Search Export:

Country	Capital	Population
South Korea	Seoul	9981619
Indonesia	Jakarta	9604900
Mexico	Ciudad de México	8591309
Russian Federation	Moscow	8389200
Japan	Tokyo	7980230
China	Peking	7472000
United Kingdom	London	7285000
Egypt	Cairo	6789479
Iran	Teheran	6758845
Peru	Lima	6464693
Thailand	Bangkok	6320174
Colombia	Santafé de Bogotá	6260862
Congo, The Demo...	Kinshasa	5064000
Chile	Santiago de Chile	4703954
Iraq	Baghdad	4336000
Singapore	Singapore	4017733
Bangladesh	Dhaka	3612850
Germany	Berlin	3386667
Myanmar	Rangoon (Yangon)	3361700
Saudi Arabia	Riyadh	3324000
Turkey	Ankara	3038159
Argentina	Buenos Aires	2982146
Spain	Madrid	2879052
Italy	Roma	2643581
Taiwan	Taipei	2641312
Ukraine	Kyiv	2624000
Ethiopia	Addis Abeba	2495000
North Korea	Pyongyang	2484000
Kenya	Nairobi	2290000
Cuba	La Habana	2256000
Algeria	Alger	2168000
France	Paris	2125246

Result Grid, Form Editor, Field Types, Query Stats, Execution Plan

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.

```
SELECT Name, Population / SurfaceArea AS density
FROM country
ORDER BY density ASC
LIMIT 10;
```

Result Grid

Name	density
Antarctica	0.0000
Bouvet Island	0.0000
South Georgia and the South Sandwich Islands	0.0000
British Indian Ocean Territory	0.0000
Heard Island and McDonald Islands	0.0000
United States Minor Outlying Islands	0.0000
French Southern territories	0.0000
Greenland	0.0259
Svalbard and Jan Mayen	0.0513
Falkland Islands	0.1643

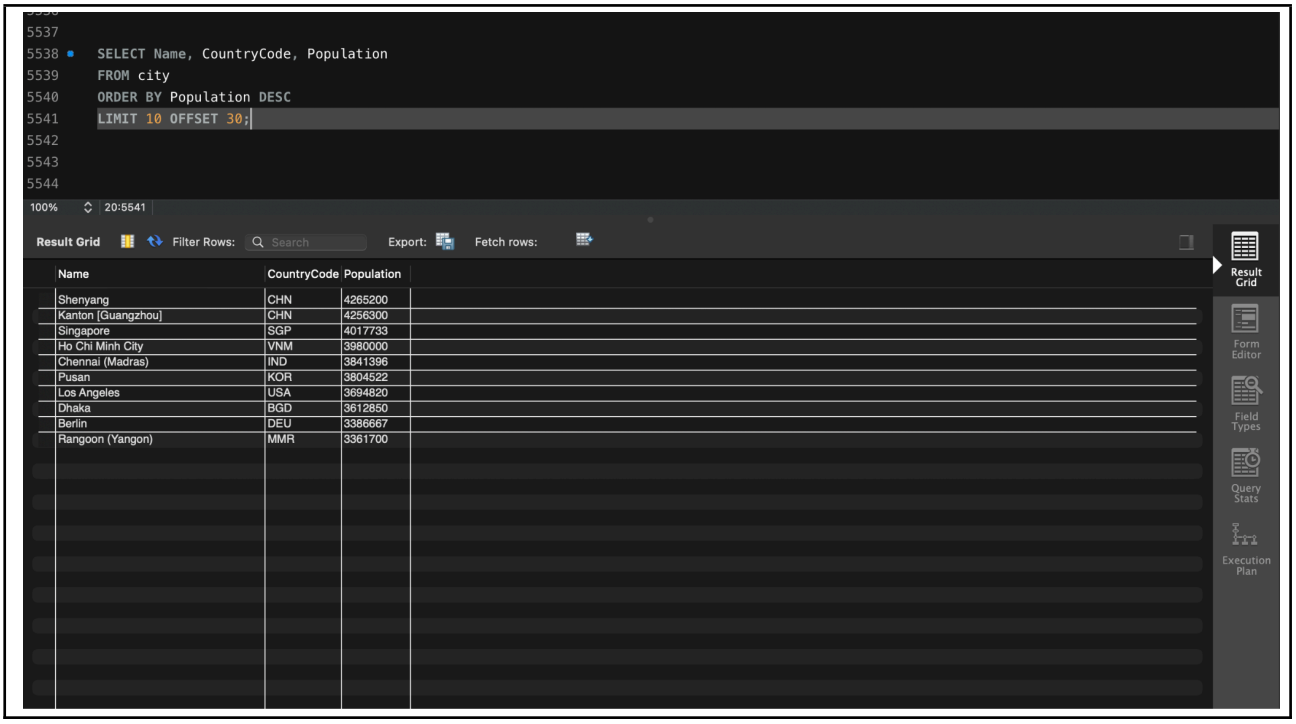
18. **Cities with High GDP per Capita:** *Scenario:* An economic consulting firm is analysing cities with high GDP per capita for investment opportunities. You're tasked with identifying cities with above-average GDP per capita from the database to assist the firm in identifying potential investment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

```
SELECT c.Name AS City, c.Population, co.Name AS Country,
       (co.GNP / co.Population) AS gdp_per_capita
FROM city c
JOIN country co ON c.CountryCode = co.Code
WHERE (co.GNP / co.Population) > (
    SELECT AVG(GNP / Population) FROM country WHERE Population > 0
)
ORDER BY gdp_per_capita DESC;
```

Result Grid

City	Population	Country	gdp_per_capi...
Luxembourg [Luxemburg/Lëtzebuerg]	80700	Luxembourg	0.037459
Zürich	336800	Switzerland	0.036936
Geneve	173500	Switzerland	0.036936
Basel	166700	Switzerland	0.036936
Bern	122700	Switzerland	0.036936
Lausanne	114500	Switzerland	0.036936
Saint George	1800	Bermuda	0.035815
Hamilton	1200	Bermuda	0.035815
Bandar Seri Begawan	21484	Brunei	0.035686
Schaan	5346	Liechtenstein	0.034644
Vaduz	5043	Liechtenstein	0.034644
George Town	19600	Cayman Isl...	0.033237
København	495699	Denmark	0.032664
Århus	284846	Denmark	0.032664
Odense	183912	Denmark	0.032664
Aalborg	161161	Denmark	0.032664
Frederiksberg	90327	Denmark	0.032664
Oslo	508726	Norway	0.032577
Bergen	230948	Norway	0.032577
Trondheim	150166	Norway	0.032577
Stavanger	108848	Norway	0.032577
Bærum	101340	Norway	0.032577
New York	8008278	United States	0.030575
Los Angeles	3694820	United States	0.030575
Chicago	2896016	United States	0.030575
Houston	1953631	United States	0.030575
Philadelphia	1517550	United States	0.030575
Phoenix	1321045	United States	0.030575

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.



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:

- Day 1

1. What is a Database?

- A database is an organised collection of data stored electronically.
- **Functions: Input → Organise → Retrieve data quickly.**
- Traditional databases are organised into:
  - **Fields → Attributes of data**
  - **Records (Rows) → Individual entries**
  - **Files (Tables) → Collections of records**

2. Key Database Terminology

- Database – A collection of related data.
- Table – Holds data in rows and columns.
- Row / Record – A single entry in a table.
- Field / Column – An attribute of data.
- Primary Key – Unique identifier for each record.
- Foreign Key – Links one table to another.
- Schema – Structure/blueprint of the database.
- View – A virtual table created from queries.
- SQL – Language to interact with databases.

3. Why Use a Database Instead of Excel?

- Handles large datasets efficiently.
- Provides relationships between multiple entities.
- Supports faster searching & indexing.
- Enforces data integrity (rules like primary/foreign keys).

4. Database Design

- Entity: Object/person/event of interest (e.g., Customer, Product, Subscription).

- Entity Description format example:
  - Customer (custID, firstname, surname, email)
  - Product (productID, title, subject, price)
  - Subscription (subID, startDate, endDate, custID, productID)
- Primary Key – Uniquely identifies each entity.
- Composite Key – Uses multiple fields together if one field alone isn't unique.
- Secondary Key – Extra searchable field (e.g., Product title).

5. Relationships Between Entities

- **One-to-One → Husband & Wife**
- **One-to-Many → School & Pupils**
- **Many-to-Many → Actor & Film**
- **Solution for many-to-many → Use a linking table (e.g., Subscription links Customer & Product).**
- Referential Integrity: Ensures foreign keys must exist in the related table.

6. Database Management Systems (DBMS)

- Software that allows users to interact with a database indirectly.
- RDBMS (Relational DBMS) – Stores data in tables with relationships.
  - Examples: MySQL, PostgreSQL, Oracle, SQL Server, SQLite.
- RDBMS uses SQL for querying data.

7. Schema Types

- Conceptual Schema – High-level, business-focused overview.
- Star Schema – Central fact table with surrounding dimension tables (good for analytics).
- Snowflake Schema – A more normalised version of star schema.

8. Non-Relational Databases (NoSQL)

- Designed for large, unstructured, or semi-structured data.
- Flexible schema, scalable, handles real-time analytics.
- Types:
  - **Document-based → MongoDB, Couchbase**
  - **Key-Value → Redis, Memcached**
  - **Graph → Neo4j**
  - **Wide-Column → Cassandra, HBase**
- Differences vs Relational:
  - More schema flexibility
  - Performance optimised for specific tasks
  - May prioritise scalability & speed over strict consistency

9. Key Takeaways

- Databases are structured ways to store and manage data.
- Relational databases use tables, primary keys, and relationships.
- Non-relational (NoSQL) databases are better for big, flexible, or real-time data.
- Understanding relationships, schemas, and integrity rules is essential in database design.

- Day 2

1. What is SQL?

- SQL (Structured Query Language):
    1. Domain-specific programming language.
    2. Developed by IBM in the 1970s.
    3. Used for storing, manipulating, and retrieving data from relational databases (RDBMS).
- Key RDBMS components:
    1. **System administrator → manages users & database monitoring.**
    2. **Schema → defines databases, tables, indexes, and database objects.**

2.SQL in Data Analysis

- CRUD operations: Create, Read, Update, Delete.
- Main uses:
    - Data selection & filtering – Queries to extract information.
    - Data modification – Insert, update, or delete records.
    - Data calculation – Using arithmetic operators & built-in functions.

3. SQL Data Types

- CHAR(n) – Fixed-length string.
- VARCHAR(n) – Variable-length string (better for names).
- INT – Whole number.
- DECIMAL(p,s) – Precise numeric (p = digits, s = decimals).
- DOUBLE – Floating-point number.
- DATE – Stores date (YYYY-MM-DD).
- TIME – Stores time (HH:MM:SS).
- DATETIME – Stores both date & time.
- YEAR – Year (YY or YYYY).
- TEXT – Long text (up to 65,535 characters).
- BLOB – Binary data.
- ENUM – Single choice from a list (e.g., 'Mr', 'Mrs').
- SET – Multiple choices from a list.
- BOOL – True/False.

4. SQL Basic Syntax

General query structure	SELECT column-name FROM table-name;
Select one column	SELECT Score FROM Studentscoring;
Select two columns	SELECT Student, Score FROM Studentscoring;
Select all columns	SELECT * FROM Studentscoring;
Filtering Data (WHERE Clause)	SELECT column-name FROM table-name WHERE condition;
Add column	ALTER TABLE Customers ADD Country VARCHAR(50);
Delete column	ALTER TABLE Customers DROP COLUMN Age;
Delete record	DELETE FROM Studentscoring WHERE Student='Caleb';
Add record:	INSERT INTO Studentscoring (ID, Student, Score) VALUES (5, 'Athi', 95);
Update record:	UPDATE Studentscoring SET Score=99 WHERE Student='Charlotte';
Ascending order (default)	SELECT * FROM Studentscoring ORDER BY Score ASC;
Descending order	SELECT * FROM Studentscoring ORDER BY Score DESC;
Use functions with AS (alias)	SELECT MIN(Score) AS Lowest FROM Studentscoring; SELECT AVG(Score) AS AverageScore FROM Studentscoring;

5. Comparison Operators

- = equal
- <> not equal
- > greater than
- < less than
- >= greater than or equal

- <= less than or equal

6. Logical Operators

- **LIKE** → pattern matching (`'C%'`, `'%son'`)
- **BETWEEN** → within a range
- **AND / OR** → multiple conditions
- **IN** → match any from a list
- **NOT** → opposite condition

7. Key Takeaways

- SQL is the main tool for interacting with RDBMS.
- Know **data types, SELECT queries, filtering, and operators**.
- Learn **modifying data** (INSERT, UPDATE, DELETE).
- Use **ORDER BY** for sorting and built-in functions for calculations.

- Day 3

SQL Subqueries, Grouping, NULLs & Joins

1. Subqueries

- **Definition**: A query within another query.
- Used when the result of one query is needed inside another.
- Common use cases:
  - **Correlated subqueries**
  - **EXISTS / NOT EXISTS**
  - **IN / NOT IN**
  - **ANY / ALL**
- Example: Find order items with price greater than the average of all orders

```
SELECT OrderID, Price
FROM Orders
```

```
WHERE Price > (SELECT AVG(Price) FROM Orders);
```

2. Grouping Data

- **GROUP BY** groups rows with the same value into summary rows.
- Typically used with aggregate functions: `COUNT()`, `SUM()`, `AVG()`, `MIN()`, `MAX()`.

```
SELECT column_name(s), AGGREGATE_FUNCTION(column)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

3. Working with NULL Values

- **NULL** = missing or unknown value (not 0 or empty string).

```
WHERE column IS NULL;
WHERE column IS NOT NULL;
```

```
SELECT * FROM Students
WHERE Course IS NULL;
```

4. Joins in SQL

- Joins combine data from multiple tables based on related columns.
- **Why use joins?** → Businesses store related data separately to reduce redundancy and improve integrity.

Types of Joins:

1. **INNER JOIN** – Returns only rows with matching values in both tables.

```
SELECT Customers.Name, Orders.OrderID
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

2. LEFT JOIN – Returns all rows from the left table and matched rows from the right table.

```
SELECT Customers.Name, Orders.OrderID
FROM Customers
LEFT JOIN Orders
```

```
ON Customers.CustomerID = Orders.CustomerID;
```

1. **RIGHT JOIN** – (Opposite of LEFT JOIN, not always supported in SQLite).
2. **FULL OUTER JOIN** – Returns all rows when there is a match in one of the tables (not supported in all DBs).

5. Key Takeaways (Day 3)

- **Subqueries**: Help filter or calculate values dynamically.
- **GROUP BY + Aggregates**: Summarise data for reporting.
- **NULL handling**: Use `IS NULL` / `IS NOT NULL`.
- **Joins**: Core SQL tool for combining multiple tables.
    - Most used join = **INNER JOIN**.

Day 4- Portfolio

Here, I created a dataset and preferred entering the data step by step using SQL commands, rather than uploading it from a CSV or SQL file.

Hospital Database Management for Starline Hospital, UK

Introduction

Business Name: Starline Hospital, UK
Industry: Healthcare

Starline Hospital is a modern healthcare facility in the UK that provides a wide range of medical services to patients. Traditionally, the hospital maintained patient records, doctor schedules, appointments, and treatments using **paper-based systems**, which made data management time-consuming and prone to errors.

To improve efficiency, accuracy, and accessibility, Starline Hospital is **moving from traditional paper records to a digital database system**. This transition allows the hospital to:

- Store patient, doctor, appointment, and treatment information in a structured way.
- Easily retrieve and analyze data for better decision-making.
- Track patient history, treatments, and appointments efficiently.
- Generate reports such as total treatment costs per patient or doctor schedules.

The SQL database developed in this portfolio represents a **simplified version of Starline Hospital's digital system**, demonstrating key operations such as storing, retrieving, and summarizing hospital data.

Steps

- **Database Creation**
- **Table Creation**
- **Inserting Data**
- **Analysis**

1. Create Database

```
CREATE DATABASE HospitalDB;

USE HospitalDB;
```

2. Create Tables

```
-- Patients table
CREATE TABLE Patients (
    PatientID INT PRIMARY KEY,
    Name VARCHAR(50),
    Age INT,
    Gender VARCHAR(10)
);

-- Doctors table
CREATE TABLE Doctors (
    DoctorID INT PRIMARY KEY,
    Name VARCHAR(50),
    Specialty VARCHAR(50)
);

-- Appointments table
CREATE TABLE Appointments (
    AppointmentID INT PRIMARY KEY,
    PatientID INT,
    DoctorID INT,
    AppointmentDate DATE,
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),
    FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID)
);

-- Treatments table
CREATE TABLE Treatments (
    TreatmentID INT PRIMARY KEY,
    PatientID INT,
    TreatmentDescription VARCHAR(100),
    Cost DECIMAL(8,2),
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID)
);
```

3. Insert Data

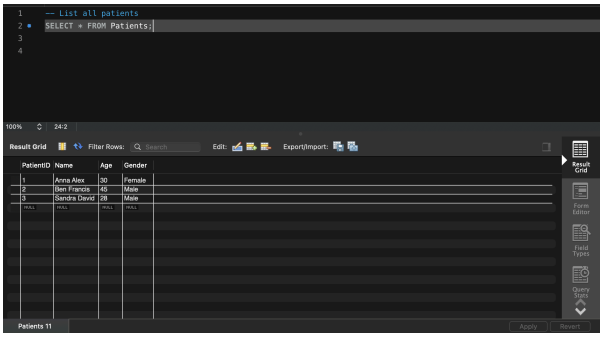
Patients
INSERT INTO Patients VALUES
(1, 'Anna Alex', 30, 'Female'),
(2, 'Ben Francis', 45, 'Male'),
(3, 'Sandra David', 28, 'Male');

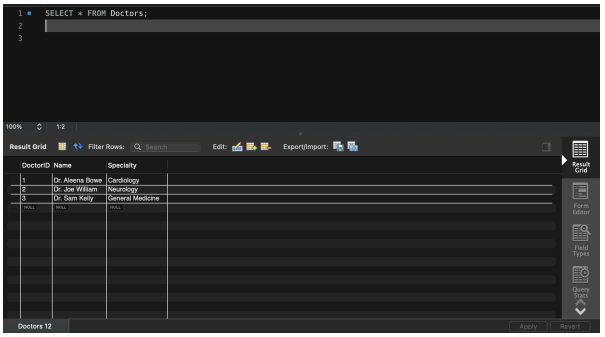
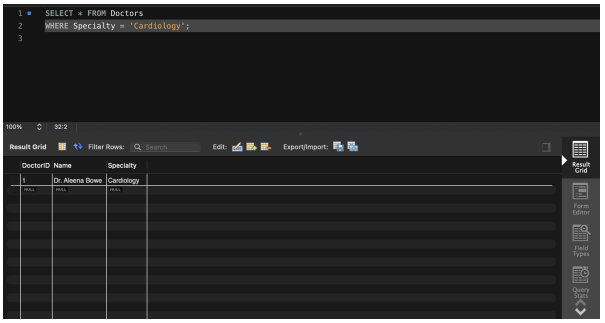
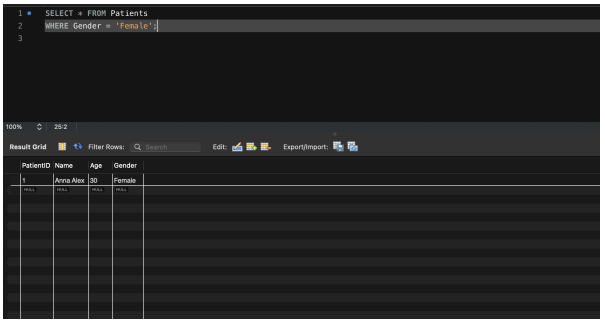
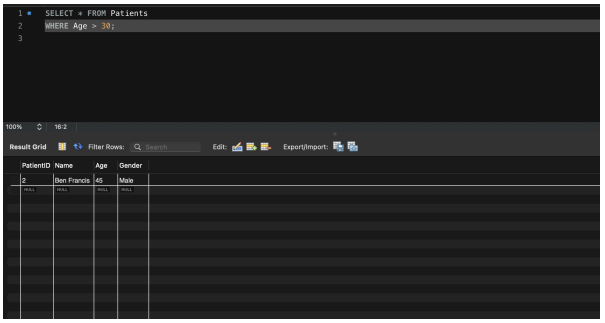
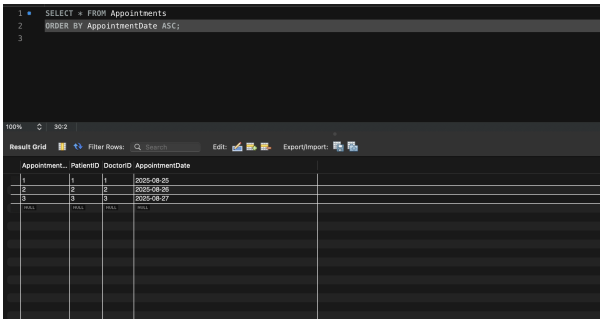
Doctors
INSERT INTO Doctors VALUES
(1, 'Dr. Aleena Bowe', 'Cardiology'),
(2, 'Dr. Joe William', 'Neurology'),
(3, 'Dr. Sam Kelly', 'General Medicine');

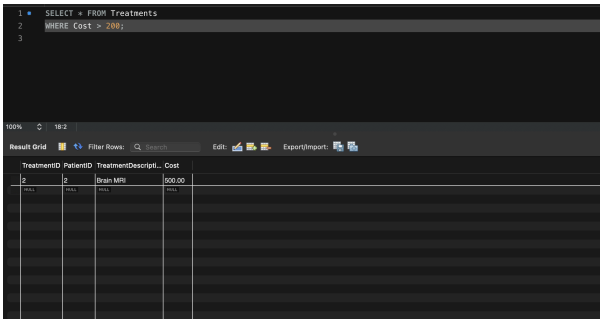
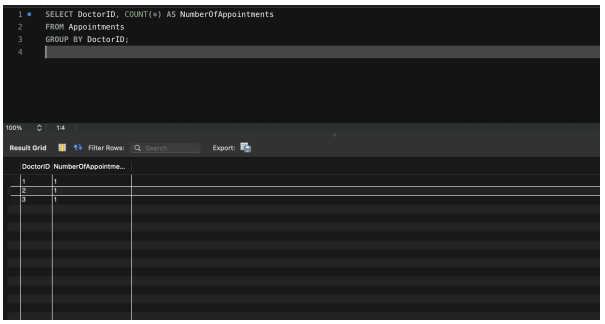
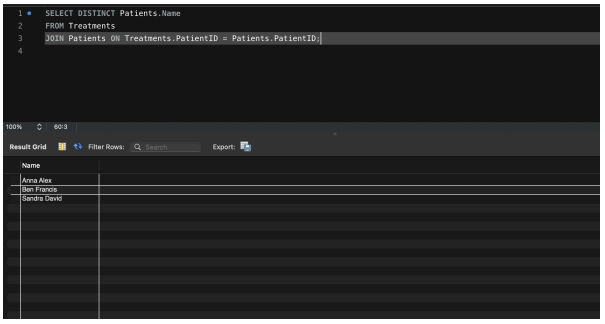
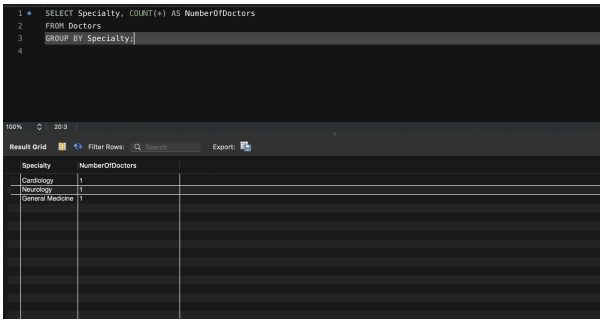
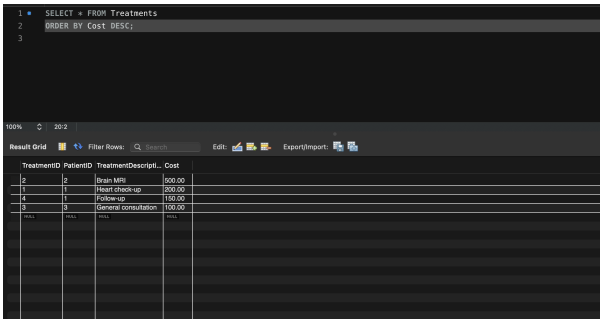
Appointments
INSERT INTO Appointments VALUES
(1, 1, 1, '2025-08-25'),
(2, 2, 2, '2025-08-26'),
(3, 3, 3, '2025-08-27');

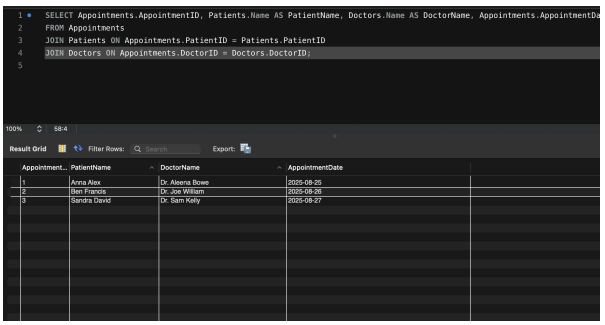
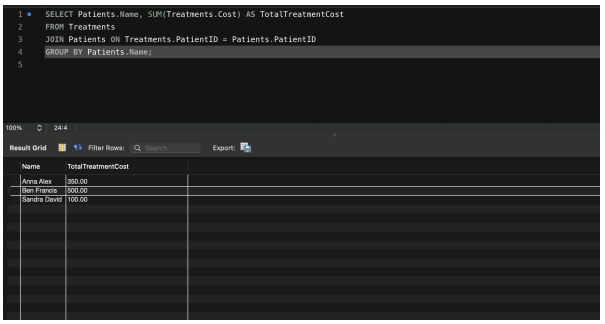
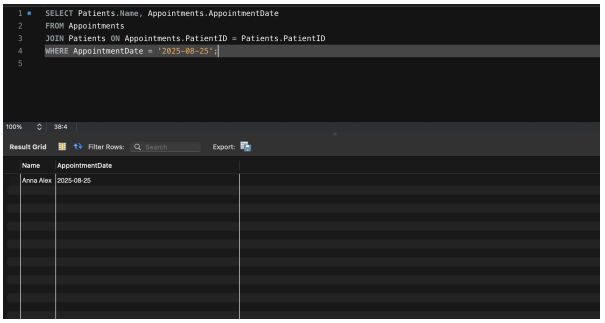
Treatments
INSERT INTO Treatments VALUES
(1, 1, 'Heart check-up', 200.00),
(2, 2, 'Brain MRI', 500.00),
(3, 3, 'General consultation', 100.00),
(4, 1, 'Follow-up', 150.00);

4. Analysing Data

CODE	OUTPUT
1. List all patients SELECT * FROM Patients;	

2. List all doctors SELECT * FROM Doctors;	
3. Find all cardiology doctors SELECT * FROM Doctors WHERE Specialty = 'Cardiology';	
4. List all female patients SELECT * FROM Patients WHERE Gender = 'Female';	
5. List all patients above 30 years old SELECT * FROM Patients WHERE Age > 30;	
6. List all appointments in ascending order by date SELECT * FROM Appointments ORDER BY AppointmentDate ASC;	

7. List all treatments that cost more than 200 SELECT * FROM Treatments WHERE Cost > 200;	
8. Find the number of appointments for each doctor SELECT DoctorID, COUNT(*) AS NumberOfAppointments FROM Appointments GROUP BY DoctorID;	
9. List all patients who have received treatments SELECT DISTINCT Patients.Name FROM Treatments JOIN Patients ON Treatments.PatientID = Patients.PatientID;	
10. Show total number of doctors in each specialty SELECT Specialty, COUNT(*) AS NumberOfDoctors FROM Doctors GROUP BY Specialty;	
11. List treatments sorted by cost in descending order SELECT * FROM Treatments ORDER BY Cost DESC;	

12. Show appointments with patient and doctor names SELECT Appointments.AppointmentID, Patients.Name AS PatientName, Doctors.Name AS DoctorName, Appointments.AppointmentDate FROM Appointments JOIN Patients ON Appointments.PatientID = Patients.PatientID JOIN Doctors ON Appointments.DoctorID = Doctors.DoctorID;	
13. Total cost of treatments per patient SELECT Patients.Name, SUM(Treatments.Cost) AS TotalTreatmentCost FROM Treatments JOIN Patients ON Treatments.PatientID = Patients.PatientID GROUP BY Patients.Name;	
14. Patients with appointments on '2025-08-25' SELECT Patients.Name, Appointments.AppointmentDate FROM Appointments JOIN Patients ON Appointments.PatientID = Patients.PatientID WHERE AppointmentDate = '2025-08-25';	

Conclusion

This SQL portfolio demonstrates how Starline Hospital, UK can transition from traditional paper-based records to a digital database system. By creating a structured database with tables for patients, doctors, appointments, and treatments, the hospital can efficiently store, manage, and retrieve important information.

The queries developed in this portfolio show how to perform basic data retrieval, filter records, combine information from multiple tables using joins, and calculate totals using aggregate functions. These operations provide meaningful insights, such as total treatment costs per patient, doctor schedules, and patient appointment details.

Overall, this project highlights the benefits of using SQL databases in healthcare, including improved accuracy, faster access to data, better decision-making, and more efficient management of hospital operations. This beginner-friendly portfolio provides a foundation for further learning and more advanced database management techniques in the future.

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.