

**Data Technician**

|  |
| --- |
|  |

|  |
| --- |
| Name: Supriya Kamble |
| Course Date: March 2025 |
|  |

**Table of contents**

[Day 1: Task 1 3](#_Toc193099469)

[Day 1: Task 2 4](#_Toc193099470)

[Day 3: Task 1 4](#_Toc193099471)

[Day 4: Task 1: SQL Practical 6](#_Toc193099472)

[Day 4: Task 2: Written (Optional) 14](#_Toc193099473)

[**Course Notes** 18](#_Toc193099474)

[**Additional Information** 19](#_Toc193099475)

# 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 column in a relational database table is distinctive,unique identifier for each row.  Eg driver’s license number, telephone number, student id. |
| How does this differ from a secondary key? | A primary key uniquely identifies each row in a table, while a secondary key (or alternate key) provides additional unique identification options or aids in data retrieval.  Unlike primary key, Secondary Key can be null, and a table can have multiple secondary keys. |
| How are primary and foreign keys related? | A foreign key column in a table, points to a column with unique values in another table (often the primary key column) to create a way of cross-referencing the two tables. |
| Provide a real-world example of a one-to-one relationship | **Country – Capital City:** Each country has one office, and each capital city belongs to one country. |
| Provide a real-world example of a one-to-many relationship | **Car and Models:** A car maker makes many different models, but a particular car model is built only by a single car maker. |
| Provide a real-world example of a many-to-many relationship | **Employees and Projects:** An employee can work on many projects, and a project can involve many employees. |

# 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 databases store data in structured tables with rows and columns, using SQL for queries.  Non-relational (NoSQL) databases offer flexible data models like key-value, document, or graph, optimized for large, diverse, or dynamic data. |
| What type of data would benefit off the non-relational model?  Why? | You should consider using a non-relational database (NoSQL) when you need to handle large volumes of unstructured or semi-structured data, prioritize scalability and performance over strict data consistency, or when your data schema is flexible and likely to change frequently. |

# 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 | The Self-Join allows you to join a table to itself. This implies that each row of the table is combined with itself and with every other row of the table. The SELF JOIN can be viewed as a join of two copies of the same table.    Example  **SELECT** e.employee\_name AS employee, m.employee\_name AS manager  **FROM** Employees AS e  **JOIN** Employees AS m  **ON** e.manager\_id = m.employee\_id;  Eg, we want to retrieve a list of all the Employees and their managers working in them from the Employee table.  Self joins are particularly useful when working with hierarchical data such as organizational structures, where each employee has a manager. |
| Right join | The Right Join selects data from the right table (Table B) and matches this data with the rows from the left table (Table A). The Right Join returns a result set that includes all rows in the right table, whether they have matching rows from the left table. In case, a row in the right table does not have any matching rows in the left table, the column of the left table in the result set will have nulls.    Example  **SELECT** emp\_no , emp\_name ,d\_name, location  **FROM** employee   **RIGHT JOIN** dept   **ON** employee.dept\_no = department.dept\_no;  It will display all the departments and location with the associated Employee no and Employee name. The output will contain some Employee no and Employee name will have null values.  It allows us to deal with missing values in database and helps in analysing relationships between data. |
| Full join | The Full Join returns a result that includes rows from both left and right tables. In case, no matching rows exist for the row in the left table, the columns of the right table will have nulls. Correspondingly, the column of the left table will have nulls if there are no matching rows for the row in the right table. An SQL Full Outer Join will retrieve not only the matching rows but also the unmatched rows as well.    Example  **SELECT** employee.e\_name, employee.e\_dept, department.d\_name, department.d\_location  **FROM** employee  **FULL JOIN** department  **ON** employee.e\_dept = department.d\_name;  Here, each employee has a corresponding department. If any employee didn't have department or vice versa, the result would still include that row with NULL in the respective columns. |
| Inner join | Inner Join statement returns only those records or rows that have matching values and is used to retrieve data that appears in both tables.    **SELECT** employee.e\_name, employee.e\_dept, department.d\_name, department.d\_location  **FROM** employee  **FULL JOIN** department  **ON** employee.e\_dept = department.d\_name;  In our example, to list each employee and the associated departments they have handled we want to extract only matching data from the Employee and Department.  By retrieving only the rows where a match is found, it helps filter relevant information efficiently. |
| Cross join | The Cross Join command in SQL, also known as a cartesian join, returns all combinations of rows from each table.    Envision that you need to find all combinations of size and colour. In that case, a CROSS JOIN will be an asset. Note, that this join does not need any condition to join two tables. In fact, CROSS JOIN joins every row from the first table with every row from the second table and its result comprises all combinations of records in two tables.  Example  **SELECT** \*  **FROM** employee  **CROSS JOIN** department;  It returns all records from both tables. Therefore, if there are rows in “Employee” or “Department” that do not match any entries in either table, those rows will also be listed.  The SQL CROSS JOIN is particularly useful in scenarios where every record needs to be paired with every other record, such as in creating combinations or generating test data. |
| Left join | The Left Outer Join gives the output of the matching rows between both tables. In case, no records match from the left table, it shows those records with null values.    Example  **SELECT** emp\_no , emp\_name ,d\_name, location  **FROM** employee  **LEFT JOIN** dept   **ON** employee.dept\_no = department.dept\_no;  In the output, in case, there are no employees matching department number, NULL values will be listed in the corresponding rows for department name and location.  By using examples like employee and department data, we can clearly see how LEFT JOIN ensures that no valuable information is left out, even when there are unmatched records. |

# Day 4: Task 1: 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**](https://justit831-my.sharepoint.com/:u:/g/personal/danpe_justit_co_uk/Ef6vAaaYVi5FhHqKGxqnn60B9g2khoYekEIO3Y7J00UcJQ?e=pv9NNE)
2. **Follow each step to create your database** [**here**](https://justit831-my.sharepoint.com/:b:/g/personal/danpe_justit_co_uk/EdeCKl2Sas1Hl7u9amDy0fIB9jGVCKxSR0u2-lFOvS5rXw?e=xKv1U7)

**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(ID) AS TotalCities  From city  JOIN country  ON city.CountryCode = country.Code  WHERE country.Code ='USA';  **OR**  SELECT Count(ID) AS TotalCities  from city  where countrycode in(Select code  from country  where code = 'USA'); |

1. **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);  **OR**  SELECT Name,LifeExpectancy  From country  order by LifeExpectancy desc  limit 1; |

1. **"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%' ; |

1. **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 \*  From city  ORDER BY Population DESC  LIMIT 10 ; |

1. **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 \*  From city  WHERE Population > 2000000 ; |

1. **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 \*  From city  WHERE Name like 'Be%' ; |

1. **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 \*  From city  WHERE Population between 500000 and 1000000 ; |

1. **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 \*  From city  ORDER BY Name ; |

1. **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 ; |

1. **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 name, COUNT(\*) as NOofOccurence  FROM city  GROUP BY name  HAVING COUNT(\*) > 1  order by name ; |

1. **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  limit 1; |

1. **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 ; |

1. **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 Name,Capital  FROM country  WHERE Name= 'Spain'; |

1. **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 c.Name,cn.Name  From city c  JOIN country cn  ON c.CountryCode= cn.code  WHERE cn.Continent ='Europe'; |

1. **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 Name,Avg(Population) as AveragePopulation  From country  group by Name; |

1. **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 cn.Name,c.Name as Capital,c.population  From city c  JOIN country cn  ON c.CountryCode= cn.code  order by c.population desc; |

1. **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  from country  where population in (select min(population) from country); |

1. **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 ,cn.Name as country,cn.GNP as HighGNP  From city c  join country cn  on c.countrycode = cn.Code  where cn.GNP > (Select avg(GNP) FROM country); |

1. **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.

|  |
| --- |
| WITH citypop AS  ( SELECT Name,Population,  ROW\_NUMBER() OVER(ORDER BY Population desc) AS RN  FROM city  )  SELECT Name,Population, RN  FROM citypop  WHERE RN BETWEEN 31 AND 40; |

# Day 4: Task 2: Written (Optional)

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****:*
   1. *What kind of data will the database need to store?*
   2. *Who will be the users of the database, and what will they need to accomplish?*
2. ***Designing the Database Schema****:*
   1. *How would you structure the database tables to efficiently store inventory, sales, and customer information?*
   2. *What relationships between tables are necessary (e.g., how sales relate to inventory and customers)?*
3. ***Implementing the Database****:*
   1. *What SQL commands would you use to create the database and its tables?*
   2. *Provide examples of SQL statements for creating tables and defining relationships between them.*
4. ***Populating the Database****:*
   1. *How would you input initial data into the database? Give examples of SQL INSERT statements.*
5. ***Maintaining the Database****:*
   1. *What measures would you take to ensure the database remains accurate and up to date?*
   2. *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 | **1.Understanding the Business Requirements** The first step is setting up a database.  a. The database needs to store data related to inventory, sales, and customers.   * **Inventory:** This will include products available for sale, their quantities, prices, and other relevant product details such as descriptions or categories (e.g. groceries, domestic products). * **Sales:** The database will track sales transactions, including items sold, the date of sale, quantities, prices, and payment details. * **Customer:** This will include customer contact details (name, email, address), and information about their purchase history, which can be used for a loyalty program. * **Loyalty program**: The database must track loyalty points for each customer, based on their purchases.   b. Users of the database will include **store staff** for inventory management, **sales personnel** for processing transactions, and the **business owner or manager** for overseeing operations.  The goal of the database is to provide an organized way to manage these elements. **2.Designing the Database Schema** The database schema must be designed to efficiently store and organize the data. This can be achieved by creating several key tables:  **a.Tables**   1. **Products** table: Stores inventory data.   product\_id, product\_name, category, price, stock\_quantity   1. **Customers** table: Stores customer information.   customer\_id, name, email, phone\_number, address   1. **Sales** table: Tracks sales transactions.   sale\_id, customer\_id, sale\_date, total\_amount   1. **Sale\_Items** table: Stores details of items sold in each sale.sale\_item\_id, sale\_id, product\_id, quantity, price   **5.Loyalty\_Program** table: Tracks customer loyalty points. loyalty\_id, customer\_id, points\_balance  **b.Relationships**:   * **Customers to Sales**: A one-to-many relationship, as each customer can have multiple sales. * **Sales to Sale\_Items**: A one-to-many relationship, as each sale can contain multiple items. * **Products to Sale\_Items**: A many-to-one relationship, as each sale item corresponds to one product. * **Customers to Loyalty\_Program**: A one-to-one relationship, as each customer has one loyalty record.  **3.Implementing the Database** To implement the database, SQL commands are used to create the necessary tables and relationships. The following are examples of SQL statements:  CREATE DATABASE ShopDB;  USE ShopDB;  CREATE TABLE Products (  product\_id INT PRIMARY KEY ,  product\_name VARCHAR(255) NOT NULL,  category VARCHAR(100),  price DECIMAL(10, 2) NOT NULL,  stock\_quantity INT NOT NULL );  CREATE TABLE Customers (  customer\_id INT PRIMARY KEY ,  name VARCHAR(255) NOT NULL,  email VARCHAR(255) NOT NULL UNIQUE,  phone\_number VARCHAR(20),  address TEXT );  CREATE TABLE Sales (  sale\_id INT PRIMARY KEY ,  customer\_id INT,  sale\_date DATETIME DEFAULT CURRENT\_TIMESTAMP,  total\_amount DECIMAL(10, 2),  FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id) );  CREATE TABLE Sale\_Items (  sale\_item\_id INT PRIMARY KEY ,  sale\_id INT,  product\_id INT,  quantity INT NOT NULL,  price DECIMAL(10, 2),  FOREIGN KEY (sale\_id) REFERENCES Sales(sale\_id),  FOREIGN KEY (product\_id) REFERENCES Products(product\_id) );  CREATE TABLE Loyalty\_Program ( loyalty\_id INT PRIMARY KEY , customer\_id INT, points\_balance INT DEFAULT 0, FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id) );  **4.Populating the Database**  Below are the examples of SQL INSERT statements to populate/insert the data into the tables:  INSERT INTO Products (product\_name, category, price, stock\_quantity) VALUES ('Banana', 'Groceries', 0.89, 50),  ('Soap', 'Domestic Products', 2.39, 10);  INSERT INTO Customers (name, email, phone\_number, address) VALUES ('Joe Campbell', 'joec@gmail.com', '7895374563', '92 Main View');  INSERT INTO Sales (customer\_id, total\_amount) VALUES (1, 25.10);  INSERT INTO Sale\_Items (sale\_id, product\_id, quantity, price) VALUES (1, 1, 50, 0.89); -- 50 Banana **5.Maintaining the Database** The different types of activities can be performed to maintain the database. It ensures the accuracy and security of the database.   * **Data security**: Using secure connections and encryption methods to protect sensitive customer information. * **Backups**: Ensuring the database is backed up periodically to avoid data loss. This action should be performed regularly. * **Data validation**: Implementing checks to prevent data inconsistencies (e.g., ensuring stock quantities are updated after sales).   Additionally, an administrator might schedule regular updates to the inventory and sales data, ensuring everything is up to date. **Conclusion** In conclusion, creating a database system for a small retail business involves understanding the requirement, designing an efficient database schema, implementing the database with SQL commands, and maintaining the system to ensure accuracy and security. |

|  |
| --- |
| **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:

|  |
| --- |
| **Additional Information** |

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.**