ADVANCED DATABASES COURSEWORK

STUDENT ID: @00704794

MSC DATA SCIENCE

School of Computing, Science & Engineering

University of Salford, Manchester.

KOSEMANI ABIOLA M.

APRIL 11, 2023

TASK 1.

ABSTRACT

What this task is set to achieve is to develop a library database management system, which enables the library to effectively control and monitor all library transactions.

The System, as described in the previous paragraph, is expected to result in an errorless, secure, stable swift management system that helps the library make better use of item resources in the library's Catalog. This project is developed using a structured query language (SQL) and it mainly focuses on various library functionalities from basic to more advanced operations.

As part of the design requirement of the project's database system based on the library's requirements, the project proposed databased is designed and normalized using 3NF, while fully explaining and justifying all database decisions by documenting the process gone through to implement this design using T-SQL statement in Microsoft SQL server management studio.

INTRODUCTION

A Library is a collection of material, books, or media which can be used in any way and not merely for display purposes. A library provides physical (hard copies) or digital access (soft copies) materials and may be a physical location or a virtual space, or both. A collection of printed materials and other material in a variety of formats like DVDs, CDs or cassettes, journals, and books can be added to the libraries' collections as well as their access to information, music, and audio-visual content stored in bibliometric databases.

The main aim of this system is to develop a new programmed system that will convey the everlasting solution to the manual base operations and to make available a channel through which staff can maintain the record easily and customers can access the information about the library at whatever place they might find themselves.

The library database management system project enables new members to be able to register with the library for item loans by providing some details which include name, address, username, password, and optional details like email and phone number. By registering using their details with the library, any registered member is entitled to loan items from the library provided their library membership status remains active. With a library decides to end his/her membership status, the library wants a system that keeps the membership details for marketing purposes. As a registered member, you are entitled to borrow any item of your choice from the library if available, otherwise, a reservation can be made for such item prior to its availability

The item borrowed from the library will be automatically assigned an overdue date. For each item borrowed from the library which has not been returned by its due date, a fine of 10 pounds will be levied every day and 24 hours per day.

All loan fines owned by the library members can be paid using various payment methods, fine associated with a particular item can either be a one-off payment or a gradual payment. A functionality that keeps a record of all the loan fines, amounts paid, and outstanding balances are also put in place.

As part of the design requirement system based on the library’s requirements, the project proposed database is designed and normalized using 3NF, while fully explaining and justifying all database design decisions.

SYSTEM DESIGN AND SOFTWARE DEVELOPMENT LIFECYCLE.

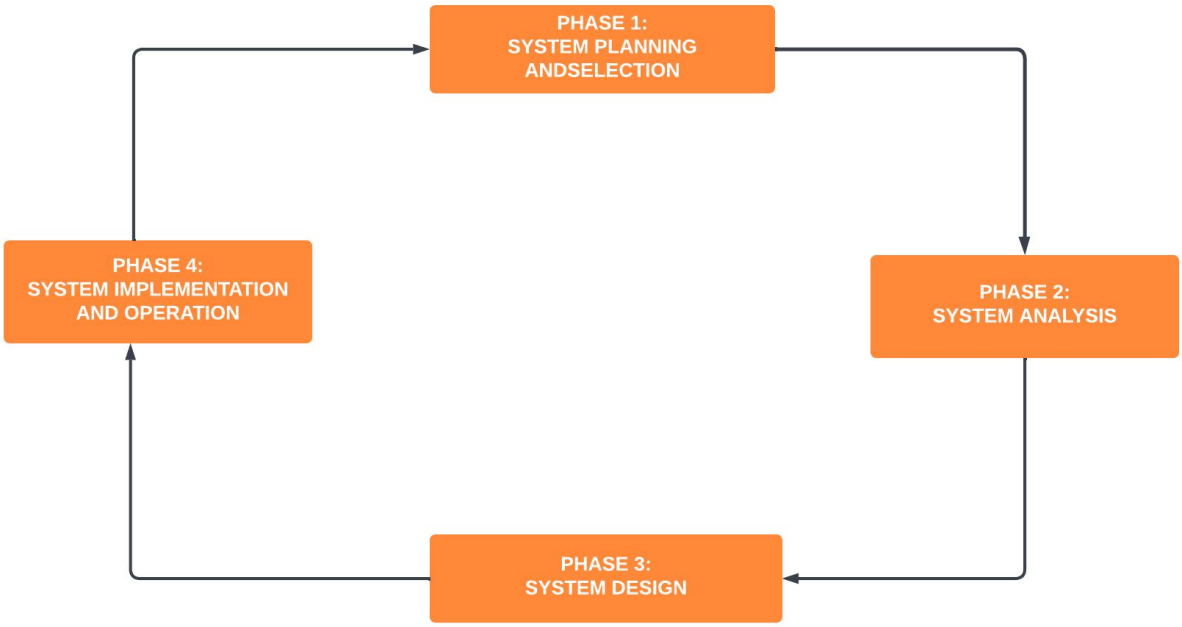


Fig 1: Diagram of a system development lifecycle.

Even though the focus of this project is on database design, how to design a high-quality database for our library management system, it's important to understand the overall principles that go into system design and all that it entails and to be able to do this effectively, it's important to investigate the concept of the system development lifecycle **(SDLC).**

When we develop a system, we go through phases, and this phase is there to give us guidelines on precisely what needs to happen to put a structure in place to avoid chaos. A system development lifecycle keeps in check the phases needed to be followed in the system development. Each of these phases is there to guide us on what's expected of each phase, with one phase not expected to happen without the other.

Throughout the ages, we have adapted and adjusted to this process to better fit our modern needs, think of Agile software development, which is a methodology of the system development lifecycle **(SDLC),** it's an application of the above phases that fits a fast-paced development lifecycle, where you constantly loop through the stages.

When we look at the **SDLC** what we are constantly looking at are the phases.

Phase 1 involves the system planning and selection, this stage involves deciding on the features you want and the reasons behind it, and everything each part of the system must do, technically the system requirements, and this part is what leads us to the second phase which is the system analysis.

Phase 2 is the part where we decide on what we want based on Phase 1, what is needed to do, and what kind of things is required in order to put in place. This is where we analyse what the system needs. And once we are done with the system analysis, we go on to design the system which leads us to phase 3, which is the system design.

In phase 3, what we basically do here is create architectural diagrams based on what is determined in phase 1, and talk about Entities, attributes, and the kind of data needed to be stored against the entities. This is the stage where we put together a concrete diagram and where we refine the phase 1 and phase 2 processes so we can have a concrete design.

In Phase 4, we go into system implementation and operation, which implies building and operating.

The reason behind the arrows flowing back and forth in the SDLC diagram is that it's a lifecycle. A system is never completely done with, there's always a thing to add to a system.

You may have an existing system and you might still want to follow the SDLC process, even though you might not have to start from zero.

The goal of **SDLC** is to get to a stage where we can design a robust system, with low coupling and high cohesion, and if possible, with little or zero downtime. The process of **SDLC** can be implemented in different ways.

SYSTEM DESIGN

System design is all about going from chaos to structure, creating structure which can be easily understood and communicated. There are different techniques to design databases depending on the business needs and requirements. The two popular way of database design includes: Top-down and bottom-up.

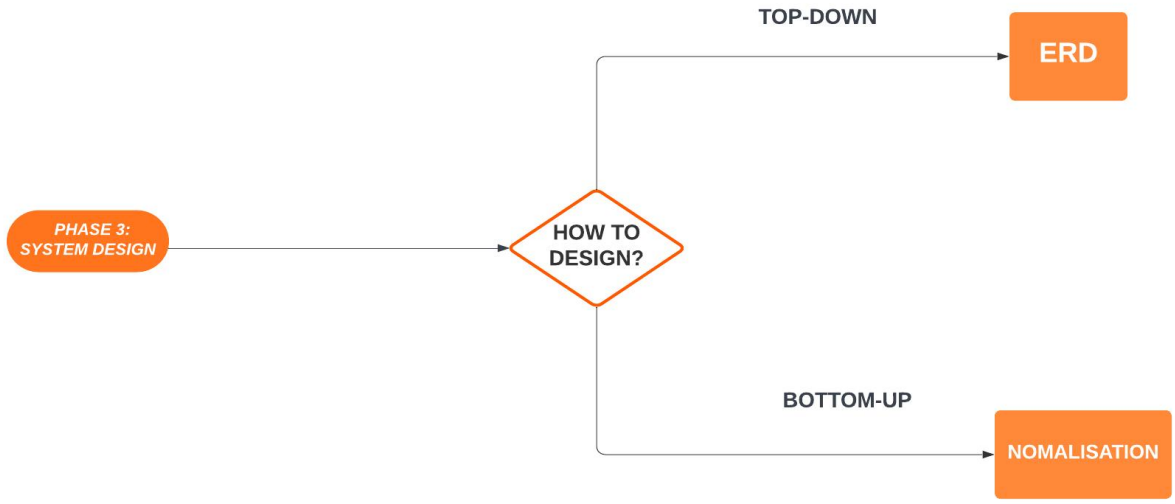


Fig 2. Database design methods.

What's the difference between top-down and bottom-up database design?

Top-down basically means starting from **zero or scratch**, there's no existing system, just only some requirements and ideas are provided, and requirements are gathered up-front an example of this is the library database management project, we must go through the **phase 1** and **phase 2** of **SDLC** to understand and formulate deep knowledge of the project requirements.

Bottom up is when we already have a **specific data** or an **existing system** in place. This design process is about trying to shape a new system around the existing data

What to use?

In this database library management system, I will be making use of both top-down and bottom-up to design my database.

DETEMINING THE RELATIONSHIPS BETWEEN THE ENTITIES

What's a database relationship?

A database relationship can be defined as the association between two or more tables, whereby one table is making use of a foreign key, that references a primary key in another table, this is a logical association between two or more tables in the database. This can be created by using a join table to query or retrieve data.

Why is database relationship important?

Creating and maintaining relationships among tables in the database not only makes it easier for us to maintain and improve the structure of our tables but also helps us reduce the number of redundant data in our database.

Establishing a good relationship among the tables also helps reduce the amount of adjustment made to the table structures, this in turn increases the effectiveness of the structure and reduces the number of unnecessary data that may be present in a table.

TYPES OF DATABASE RELATIONSHIPS

One-to-One (1: 1): A spouse in a marriage who doesn't have more than one partner or maintains just a single partner is an example of a one-to-one relationship. Only one record or sometimes no record can exist between two different tables in this relationship, the foreign table A is going to be a Primary key in table B.

One-to-Many (1: N): A spouse in a marriage with more than one or more partner is an example of one relationship. This is like a record in table A that forms an association with other tables B and C using table A's primary key as a foreign key in tables B and C.

Many-to-Many (M: N): A many-to-many relationship just as it sounds is a complicated relationship, which I have tried to avoid during my database design process, this is normally done by bringing in an intermediary table to breakdown the complex relationship, an example is the relationship between Authors and A book. Multiple records in a table having a relationship with multiple records from another table.

ANOMILIES AND DATABASE NORMALIZATION

Modification anomalies are problems that arise when the database is not structured correctly. All the design techniques (entity relationship diagram, bottom-up and top-down design) which I have decided to adopt during this project database design, are to have a correct structure and reduce database anomalies, to avoid production anomalies which can make it difficult to maintain the database.

There are three types of anomalies

- Update Anomalies
- Insert Anomalies
- Delete Anomalies

Avoiding anomalies is the key to database design, this is achieved by creating a loosely coupled system of data that has relationships and link data together but reduce anomalies and redundancy. The way to avoid these anomalies and redundancy in top-down design is through entity relationship diagram, and the way to avoid it in bottom-up is through a process called normalization.

Normalization: This is a structured way to avoid anomalies, it's a structured way to go about looking at the attributes, and data that you are storing and at the same time creating data schema and deriving entities from that data schema in a way that concretely tells us if we are going to end up with anomalies and redundancy.

EDGAR CODD PROPOSED THE THEORY OF NORMALIZATION.

Edgar Codd, one of the inventors of SQL, proposed the normalization theory. And normalization all sums up to be a design technique to reduce redundancy and anomalies.

To do normalization, one must understand two major things, first is the functional dependencies of data and the second is to understand the normal forms and what it entails to apply normalization. Normalization happens through a process of running attributes through the normal forms with each normal form aiming to separate further relationships by breaking it down into smaller instances to create less redundancy and anomalies. The normal forms from 0 through BCNF are the most common normal forms to run through, the normal form used in the database design is the 3 NF.

ENTITY RELATIONSHIP DIAGRAM (ERD).

QUESTION 1.

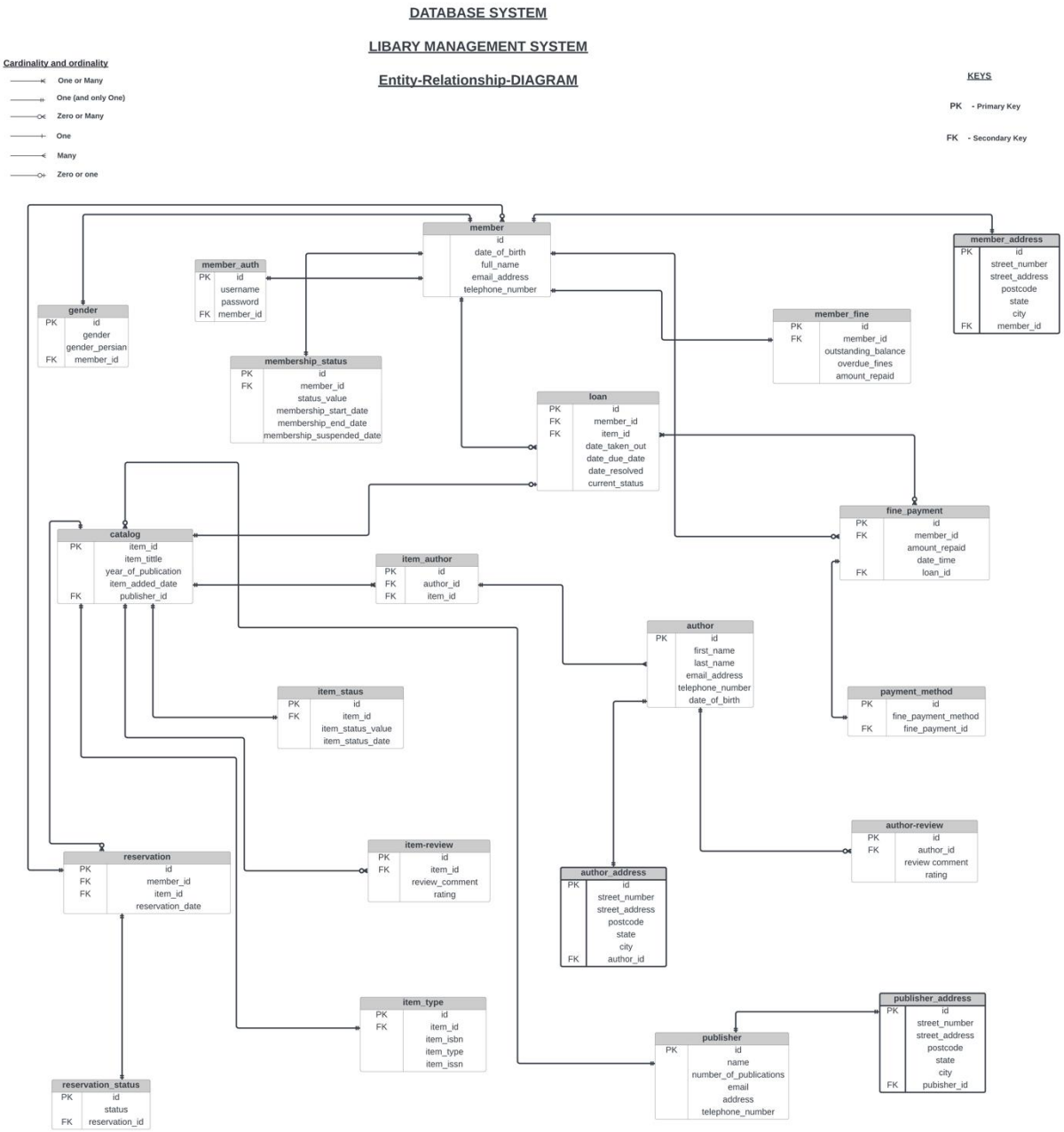


Fig 3: Entity relationship diagram for library database management system.

ENTITY RELATIONSHIP DIAGRAM EXPLANANTION

Entity relationship diagram or Entity relationship model is a top-down approach of thinking in system design, it's a diagram that functions to structure a high-level requirement. It's all about forming the relationship between entities.

The above entity relationship diagram in fig 3, showcase the intended relationships among the entities.

ENTITIES

The entities in the ER-Diagram includes:

- Member.
- Member_auth
- Gender
- Membership_status
- Member_address
- Member_fine
- Loan
- Fine_payment
- Payment_method
- Catalog
- Item_status
- Item_type
- Item_review
- Reservation
- Resevation_status
- Author
- Item_author
- Author_address
- Author_review
- Publisher
- Publisher_address

ATTRIBUTES

- Member.
  - Id
  - Date_of_birth
  - Full_name
  - Email_address
  - Telephone_number

- Member_auth
  - Id
  - Username
  - Password
  - Member_id

- Gender
  - Id
  - Gender
  - Member_id

- Membership_status
  - Id
  - Membership_start_date
  - Membership_end_date
  - Membership_suspended_date
  - Member_id

- Member_address
  - Id
  - Street_number
  - Street_address
  - Postcode
  - City
  - State
  - Member_id

- Member_fine
  - Id
  - Member_id
  - Total_fines
  - Amount_repaid
  - Outstanding_balance

- Loan
  - Id
  - Member_id
  - Item_id
  - Total_fines
  - Date_taken_out
  - Date_due_back
  - Date_returned
  - Status

- Fine_payment
  - Id,
  - Member_id
  - Time
  - Loan_id
  - Amount_repaid

- Payment_method
  - Id,
  - Fine_payment
  - Fine_payment_method

- Catalog
  - Item_id
  - Item_title
  - Year_of_publication
  - Date_added
  - Publisher_id

- Item_status
  - Id
  - Item_id
  - Item_status_value
  - Item_status_date

- Item_type
  - Id
  - Item_id
  - Item_type
  - Item_isbn
  - Item_issn

- Item_review
  - Id
  - Item_id
  - Rating
  - Review_comment

- Reservation
  - Id
  - Item_id
  - Reservation
  - Member_id

- Resevation_status
  - Id
  - Reservation_id
  - Status

- Author
  - Id
  - First_name
  - Last_name
  - No_of_books_written
  - Date_of_birth
  - Email_address
  - Telephone_number

- Item_author
  - Id
  - Item_id
  - Author_id

- Author_address
  - Id
  - Street_number
  - Street_address
  - Postcode
  - City
  - State
  - Author_id

- Author_review
  - Id
  - Item_id
  - Rating
  - Review_comment

- Publisher
  - Id
  - Publisher_name
  - Number_of_publications
  - Telephone_number
  - Email

- Publisher_address
  - Id
  - Street_number
  - Street_address
  - Postcode
  - City
  - State
  - Author_id

RELATIONSHIPS – CARDINALITY

- Member to Member_auth. (1: 1)
- Member to Gender (1: 1)
- Member to Membership_status (1: 1)
- Member to Member_address. (1: 1)
- Member to Member_fine (1: N)
- Member to Loan (1: N)
- Member to Fine_payment. (1: N)
- Loan to Fine_payment. (1: N)
- Fine_payment to Payment_method (1: 1)
- Publisher to Catalog (Item) (1: N)
- Catalog (Item) to Item_status (1: 1)
- Catalog (Item) to Item_type (1: 1)
- Catalog (Item) to Loan (1: 1)
- Catalog (Item) to Item_review (1: N)
- Member to Reservation (1: N)
- Item to Reservation (1: 1)
- Reservation to Resevation_status (1: 1)
- Catalog (Item) to Item_author (1: N)
- Author to Item_author (1: N)
- Author to Author_address (1: 1)

- Author to Author_review (1: N)
- Publisher to Publisher_address (1: 1)

MANY-TO-MANY RELATIONSHIPS BETWEEN ITEM AND AUTHOR.

In the relational model, it isn't possible to store a many to many relationships, technically you can do it, but it's not advisable because you create more overhead. **Insert overhead, update overhead, delete overhead and potential redundancy.**

As a rule of thumb always try to resolve many to many relationships, there will be some situations where many to many relationships maybe valid and those situations are very dependent on the system.

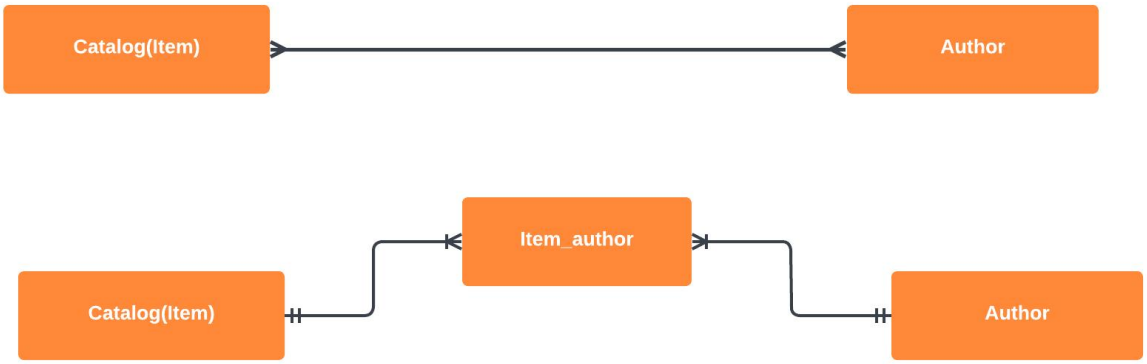


Fig 4.

If you look at the relationship between items and authors, it is a many to many relationships.

The entity item_author was created to simplify the relationship. An item in a catalogue can have multiple item author and an author can have multiple item_author.

Because we are linking the two entities with an intermediate table (entity type), we no longer have the Insert overhead, update overhead, delete overhead anomalies.

DATA TYPES USED FOR TABLES

Each column in the library database management table is required to have name and data type. As a database consultant for the library on this project, my responsibility requires deciding the type of data that will be stored inside each column in the process of creating the tables.

This data types serves as a guideline for SQL to understand the appropriate data type, which is expected inside each of the table column, this also affects how SQL interact with stored data.

We have three main data types in MySQL which includes: **String, Numeric, Date and time.**

In this project, I will be making use of all the three main data types which includes:

String Data Type

Varchar: A variable length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum column length in characters - can be from 0 to 65535

Numeric Data Type

Integer: A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.

Decimal: An unpacked floating-point number that cannot be unsigned. In the unpacked decimals, each decimal corresponds to one byte.

Date Data Type

Date: The date format is expected to be in this way YYYY-MM-DD, the range supported ranges from 1000-01-01 and 9999-12-31.

Time: The time format is expected to be in this way HH:MM: SS, the range supported ranges from -838:59:59 to 838:59:59.

```
1
2   USE master
3   GO
4   -- Create the new database if it does not exist already
5   IF NOT EXISTS (
6       SELECT [name]
7   FROM sys.databases
8   WHERE [name] = N'LibaryManagementSystem'
9   )
10  CREATE DATABASE LibaryManagementSystem
11  GO
12
13
14
15  ------ USE the LibaryManagementSystem DATABASE---------
16
17  USE LibaryManagementSystem;
18
19
20
```

Fig 5: Creating and using the database

The Fig 5 above started with the **USE master** statement, which is a way of specifying the context or environment I want the subsequent lines of codes to run, in this case the master environment.

Then the next is an **IF NOT EXISTS** statement which check the database **LibaryManagementSystem** exist by checking all the databases. Hence the statement **WHERE [name] = N'LibaryManagementSystem.**

If the database doesn't exist, the database with name LibaryManagementSystem get created.

Then USE **LibaryManagementSystem** changes the initial master environment to the new environment, **LibaryManagementSystem.**

```
21
22
23  ------ script to drop all tables if they exist---------
24
25
26  IF OBJECT_ID('member_auth', 'U') IS NOT NULL
27      DROP TABLE member_auth;
28
29  IF OBJECT_ID('member_address', 'U') IS NOT NULL
30      DROP TABLE member_address;
31
32  IF OBJECT_ID('gender', 'U') IS NOT NULL
33      DROP TABLE gender;
34
35  IF OBJECT_ID('membership_status', 'U') IS NOT NULL
36      DROP TABLE membership_status;
37
38  IF OBJECT_ID('member_fine', 'U') IS NOT NULL
39      DROP TABLE member_fine;
40
41  IF OBJECT_ID('item_status', 'U') IS NOT NULL
42      DROP TABLE item_status;
43
44  IF OBJECT_ID('item_type', 'U') IS NOT NULL
45      DROP TABLE item_type;
46
47
48  IF OBJECT_ID('payment_method', 'U') IS NOT NULL
49      DROP TABLE payment_method;
50
51
52  IF OBJECT_ID('fine_payment', 'U') IS NOT NULL
53      DROP TABLE fine_payment;
54
55  IF OBJECT_ID('author_address', 'U') IS NOT NULL
56      DROP TABLE author_address;
57
58  IF OBJECT_ID('authors', 'U') IS NOT NULL
59      DROP TABLE authors;
60
61  IF OBJECT_ID('item_author', 'U') IS NOT NULL
62      DROP TABLE item_author;
63
```

Fig 6

```
63
64
65  IF OBJECT_ID('reservations', 'U') IS NOT NULL
66      DROP TABLE reservations;
67
68  IF OBJECT_ID('reservation_status', 'U') IS NOT NULL
69      DROP TABLE reservation_status;
70
71  IF OBJECT_ID('item_reviews', 'U') IS NOT NULL
72      DROP TABLE item_reviews;
73
74  IF OBJECT_ID('author_reviews', 'U') IS NOT NULL
75      DROP TABLE author_reviews;
76
77  IF OBJECT_ID('loans', 'U') IS NOT NULL
78      DROP TABLE loans;
79
80  IF OBJECT_ID('members', 'U') IS NOT NULL
81      DROP TABLE members;
82
83  IF OBJECT_ID('catalog', 'U') IS NOT NULL
84      DROP TABLE catalog;
85
86  IF OBJECT_ID('pulisher_address', 'U') IS NOT NULL
87      DROP TABLE pulisher_address;
88
89  IF OBJECT_ID('publishers', 'U') IS NOT NULL
90      DROP TABLE publishers;
91
92
```

Fig 7

The scripts in **fig 6** and **fig 7** drops tables with selected names, by first checking if the table exist in the database.

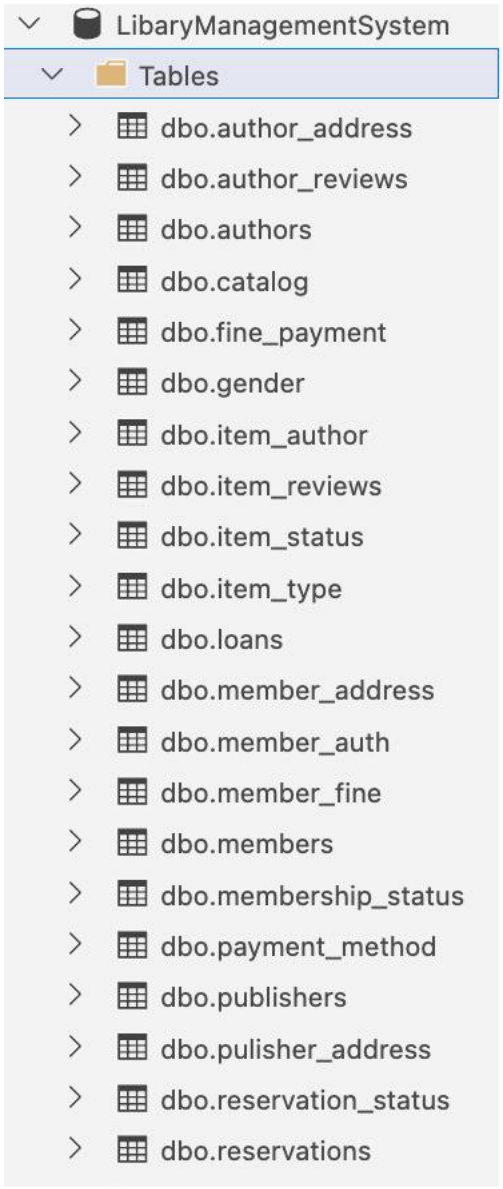


Fig 8: Inserted tables.

Fig 8 shows all the inserted tables into the database with table names. A total of 21 tables was created.

CREATING TABLES WITH T-SQL STATEMENTS

QUESTION 1.

```
----------- MEMBER'S TABLE ------------

CREATE TABLE members
(
    id INT IDENTITY(1, 1) NOT NULL,
    full_name VARCHAR(100) NOT NULL,
    date_of_birth DATE NOT NULL,
    email_address VARCHAR(255) UNIQUE NULL,
    telephone_number VARCHAR(20) UNIQUE NULL,
    PRIMARY KEY (id),
    CONSTRAINT chk_date_of_birth CHECK (date_of_birth <= GETDATE()),
    CONSTRAINT chk_email_member CHECK (email_address LIKE '%_@__%.%')
);

-------MEMBER'S ADDRESS TABLE---------

create Table member_address
(
    id INT IDENTITY(1, 1) NOT NULL,
    street_number INT NOT NULL,
    street_address VARCHAR(200) NOT NULL,
    city VARCHAR(100) NOT NULL,
    state VARCHAR(100) NOT NULL,
    member_id INT NOT NULL,
    postcode VARCHAR(50) NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (member_id) REFERENCES members(id)
);

--------- MEMBER'S AUTH TABLE -----------

CREATE TABLE member_auth
(
    id INT IDENTITY(1, 1) NOT NULL,
    username VARCHAR(50) NOT NULL UNIQUE,
    password BINARY(64) NOT NULL,
    member_id INT NOT NULL,
    FOREIGN KEY (member_id) REFERENCES members(id),
    PRIMARY KEY (id)
);
```

Fig 9.

Fig 9 shows the first 3 tables created using T-SQL statements, the member's table which is responsible for saving the member's details, the member's address table which is responsible for storing each member's address and the member's auth table which is responsible for storing user's login details.

The word auth, comes from authentication, this is a check we make against the database to verify a member's identity. The need to have a separate member's auth table is mainly for security reasons, in some applications, where caching tools like Redis are used, it wouldn't be safe to have the user's important information like password cached, as it might be exposed to a malicious act.

The section **IDENTITY (1,1)** specifies that this column is an identifier, and the SQL database will automatically generate a new value for it whenever you add a new row to your table.

The starting value and increment of the identification values are specified in the two numbers inside the parentheses. In this case, the starting value will be 1 and the increment is 1, which means that the first row inserted in a table has an id of 1, the second row would have an ID of 2, etc.

To ensure that values in columns or sets of columns are unique on all rows within the tables, a **UNIQUE** constraint is applied to the tables. This means that columns in which the value is defined as unique should not have identical values for two rows.

To make sure that columns in a table do not contain **NULL** values, we use the **'NOT NULL'** constraint.

The **check_date_of_birth** constraint in the member's table makes sure the date of birth provided is not a future date.

The **check_email_member** makes sure a valid email with **@** sign is provided, this is a low-level email validation as azure data studio doesn't completely support the use of **REGEX**, as **LIKE** doesn't work in this situation to validate email regex. An alternative way to use regex according to Microsoft documentation is by writing and importing a script.

```
--------- MEMBER'S GENDER STATUS TABLE ----------


CREATE TABLE gender
(
    id INT IDENTITY(1, 1) NOT NULL,
    gender VARCHAR(50) NOT NULL,
    member_id INT NOT NULL,
    FOREIGN KEY (member_id) REFERENCES members(id),
    PRIMARY KEY (id)
);


---------- MEMBER'S MEMBERSHIP STATUS TABLE ---------


CREATE TABLE membership_status
(
    id INT IDENTITY(1, 1) NOT NULL,
    membership_start_date DATE NOT NULL,
    membership_end_date DATE NULL,
    membership_suspended_date DATE NULL,
    member_id INT NOT NULL,
    FOREIGN KEY (member_id) REFERENCES members(id),
    PRIMARY KEY (id)
);


---------- MEMBER'S FINE TABLE ------------


CREATE TABLE member_fine
(
    id INT IDENTITY(1, 1) NOT NULL,
    member_id INT NOT NULL,
    total_fines DECIMAL(10,2) NOT NULL DEFAULT 0.00,
    amount_paid DECIMAL(10,2) NOT NULL DEFAULT 0.00,
    outstanding_balance DECIMAL(10,2) NOT NULL DEFAULT 0.00,
    PRIMARY KEY (id),
    FOREIGN KEY (member_id) REFERENCES members(id)
);
```

Fig 10.

```
---------------- PUBLISHER'S TABLE ----------------

CREATE TABLE publishers
(
    id INT IDENTITY(1, 1) NOT NULL,
    publisher_name VARCHAR(50) NOT NULL,
    number_of_publications INT,
    telephone_number VARCHAR(20) UNIQUE NOT NULL,
    email VARCHAR(50) UNIQUE NOT NULL,
    PRIMARY KEY (id),
    CONSTRAINT publisher_name_unique UNIQUE (publisher_name),
    CONSTRAINT chk_email_publisher CHECK (email LIKE '%_@__%.%')
);

---------------- PUBLISHER'S ADDRESS TABLE ----------------

CREATE TABLE pulisher_address
(
    id INT IDENTITY(1, 1) NOT NULL,
    street_number INT NOT NULL,
    publisher_id INT NOT NULL,
    street_address VARCHAR(200) NOT NULL,
    city VARCHAR(100) NOT NULL,
    state VARCHAR(100) NOT NULL,
    postcode VARCHAR(50) NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (publisher_id) REFERENCES publishers (id)
);

---------- CATALOG TABLE ------------

CREATE TABLE catalog
(
    item_id INT IDENTITY(1, 1) NOT NULL,
    item_title VARCHAR(255) NOT NULL,
    year_of_publication DATE NOT NULL,
    date_added DATE NOT NULL,
    publisher_id INT NOT NULL,
    FOREIGN KEY (publisher_id) REFERENCES publishers(id),
    PRIMARY KEY (item_id)
);
```

Fig 11.

```
-------- CATALOG ITEM'S STATUS TABLE  ------------

CREATE TABLE item_status
(
    id INT IDENTITY(1, 1) NOT NULL,
    item_id INT NOT NULL,
    item_status_value VARCHAR(50) NOT NULL,
    item_status_date DATE NOT NULL,
    FOREIGN KEY (item_id) REFERENCES catalog(item_id),
    PRIMARY KEY (id)
);

---------- CATALOG ITEM TYPE TABLE -----------

--ISBN CONSTRAINT CHECK : -- ISBN CAN EITHER BE 10(ISBN-10) OR 13(ISBN-13) DIGITS NO
-- item_type_constraint : When item type is a book, item_isbn shouldn't be null and the same implies for dvd

CREATE TABLE item_type
(
    id INT IDENTITY(1, 1) NOT NULL,
    item_id INT NOT NULL,
    item_type VARCHAR(50) NOT NULL,
    item_isbn VARCHAR(50) NULL,
    item_issn VARCHAR(50) NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (item_id) REFERENCES catalog(item_id),
    CONSTRAINT item_type_constraint  CHECK ( (item_type = 'book' AND item_isbn IS NOT NULL AND item_issn IS NULL)
        OR (item_type = 'dvd' AND item_issn IS NOT NULL AND item_isbn IS NULL))
);


--------- MEMBER'S LOAN TABLE ------------

CREATE TABLE loans
(
    id INT IDENTITY(1, 1) NOT NULL,
    member_id INT NOT NULL,
    item_id INT NOT NULL,
    total_fines DECIMAL(10,2) NOT NULL DEFAULT 0.00,
    date_taken_out DATE NOT NULL,
    date_due_back DATE NOT NULL,
    date_returned DATE NULL,
    status VARCHAR(20) CHECK (status IN('Returned', 'Not-Returned')) DEFAULT 'Not-Returned' NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (member_id) REFERENCES members(id),
    FOREIGN KEY (item_id) REFERENCES catalog(item_id)
);
```

Fig 12.

```sql
--------- MEMBER'S FINE PAYMENT TABLE ------------
CREATE TABLE fine_payment
(
    id INT IDENTITY(1, 1) NOT NULL,
    member_id INT NOT NULL,
    time TIME NOT NULL,
    loan_id INT NOT NULL,
    amount_paid DECIMAL(10,2) NOT NULL DEFAULT 0.00,
    PRIMARY KEY (id),
    FOREIGN KEY (member_id) REFERENCES members(id),
    FOREIGN KEY (loan_id) REFERENCES loans(id)
);

---------- MEMBER'S FINE PAYMENT METHOD TABLE -----------
CREATE TABLE payment_method
(
    id INT IDENTITY(1, 1) NOT NULL,
    fine_payment_id INT NOT NULL,
    fine_payment_method VARCHAR(50) NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (fine_payment_id) REFERENCES fine_payment(id)
);

--------- AUTHOR'S TABLE ------------
CREATE TABLE authors
(
    id INT IDENTITY(1, 1) NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    no_of_books_written INT DEFAULT 0,
    date_of_birth DATE NOT NULL,
    email_address VARCHAR(255) UNIQUE NULL,
    telephone_number VARCHAR(20) UNIQUE NULL,
    PRIMARY KEY (id),
    CONSTRAINT chk_date_of_birth_author CHECK ( date_of_birth <= GETDATE()),
    CONSTRAINT chk_email_author CHECK (email_address LIKE '%_@__%.%')
);
```

Fig 13.

```
-------AUTHOR'S ADDRESS TABLE---------

create Table author_address
(
    id INT IDENTITY(1, 1) NOT NULL,
    street_number INT NOT NULL,
    street_address VARCHAR(200) NOT NULL,
    city VARCHAR(100) NOT NULL,
    state VARCHAR(100) NOT NULL,
    author_id INT NOT NULL,
    postcode VARCHAR(50) NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (author_id) REFERENCES authors(id)
);

---------- ITEM-AUTHOR'S TABLE -----------

CREATE TABLE item_author
(
    id INT IDENTITY(1, 1) NOT NULL,
    item_id INT NOT NULL,
    author_id INT NOT NULL,
    FOREIGN KEY (author_id) REFERENCES authors(id),
    FOREIGN KEY (item_id) REFERENCES catalog(item_id),
    PRIMARY KEY (id)
);
```

Fig 14.

ADDED TABLES T-SQL STATEMENT

QUESTION 7.

```
----------------------ADDED TABLES ----------------------------

----------------- ITEM RESERVATION'S TABLE -------------------
CREATE TABLE reservations
(
    id INT IDENTITY(1, 1) NOT NULL,
    item_id INT NOT NULL,
    reservation_date DATE NOT NULL,
    member_id INT NOT NULL,
    FOREIGN KEY (member_id) REFERENCES members(id),
    FOREIGN KEY (item_id) REFERENCES catalog(item_id),
    PRIMARY KEY (id)
);

------------------ ITEM RESERVATION'S STATUS ------------------
CREATE TABLE reservation_status
(
    id INT IDENTITY(1, 1) NOT NULL,
    reservation_id INT NOT NULL,
    status VARCHAR(100) NOT NULL,
    FOREIGN KEY (reservation_id) REFERENCES reservations(id),
    PRIMARY KEY (id)
);

------------------ ITEM REVIEW'S TABLE -----------------
--item_rating_check constrainst make sure the review's rating falls between the range 0f 1-5
CREATE TABLE item_reviews
(
    id INT IDENTITY(1, 1) NOT NULL,
    item_id INT NOT NULL,
    rating INT NOT NULL,
    review_comment VARCHAR(1000),
    FOREIGN KEY (item_id) REFERENCES catalog(item_id),
    PRIMARY KEY (id),
    CONSTRAINT item_rating_check CHECK (rating >= 1 AND rating <= 5),
);

------------------- AUTHOR'S REVIEW TABLE -----------------
CREATE TABLE author_reviews
(
    id INT IDENTITY(1, 1) NOT NULL,
    author_id INT NOT NULL,
    rating INT NOT NULL,
    review_comment VARCHAR(1000),
    FOREIGN KEY (author_id) REFERENCES authors(id),
    PRIMARY KEY (id),
    CONSTRAINT author_rating_check CHECK (rating >= 1 AND rating <= 5),
);
```

Fig 15.

MEMBER'S PROCEDURES

```
---------------------  STORE PROCEDURE TO INSERT A NEW MEMBER INTO THE MEMBER'S TABLE -----------------------------------
GO
CREATE PROCEDURE addNewMember
    @full_name VARCHAR(100),
    @date_of_birth DATE,
    @email_address VARCHAR(255) = NULL,
    @telephone_number VARCHAR(20) = NULL,
    @id INT OUTPUT
AS
BEGIN
    INSERT INTO members
        (full_name, date_of_birth, email_address, telephone_number)
    OUTPUT
    inserted.id
    VALUES
        (@full_name, @date_of_birth, @email_address, @telephone_number)

    SET @id = SCOPE_IDENTITY();
END
GO

---------------------  STORE PROCEDURE TO INSERT A NEW ADDRESS INTO THE ADDRESS'S TABLE -----------------------------------
GO
CREATE PROCEDURE addNewAddress
    @street_number INT,
    @member_id INT,
    @street_address VARCHAR(200),
    @city VARCHAR(100),
    @state VARCHAR(100),
    @postcode VARCHAR(50)
AS
BEGIN
    INSERT INTO member_address
        (street_number, member_id, street_address, city, state, postcode)
    VALUES
        (@street_number, @member_id, @street_address, @city, @state, @postcode)
END
GO

 ----------------------  STORE PROCEDURE TO INSERT A MEMBER'S GENDER STATUS INTO THE MEMBER'S GENDER STATUS TABLE --
 GO
 CREATE PROCEDURE addMemberGender
     @gender VARCHAR(50),
     @member_id INT
 AS
 BEGIN
     INSERT INTO gender
         (gender)
     VALUES
         (@gender)
 END
 GO

 ---------------------  STORE PROCEDURE TO MEMBER'S AUTH TO MEMBER'S AUTH TABLE -----------------------------------
 GO
 CREATE PROCEDURE addNewMemberAuth
     @username VARCHAR(50),
     @password BINARY(255),
     @member_id INT
 AS
 BEGIN
     INSERT INTO member_auth
         (username, password, member_id)
     VALUES
         (@username, @password, @member_id)
 END
 GO
```

Fig 16.

```
---------------------  STORE PROCEDURE TO INSERT A MEMBERSHIP STATUS INTO THE MEMBERSHIP STATUS TABLE ------------

GO
CREATE PROCEDURE addMemberShipStatus
    @membership_start_date DATE,
    @membership_end_date DATE = NULL,
    @membership_suspended_date DATE = NULL,
    @member_id INT
AS
BEGIN
    INSERT INTO membership_status
        (membership_start_date, membership_end_date, membership_suspended_date, member_id)
    VALUES
        (@membership_start_date, @membership_end_date, @membership_suspended_date, @member_id)
END
GO



------------------------ STORE PROCEDURE TO UDATE MEMBER'S DETAIL ----------------------------
GO
CREATE PROCEDURE updateMemberDetails
    @member_id INT,
    @full_name VARCHAR(100),
    @date_of_birth DATE,
    @email_address VARCHAR(255),
    @telephone_number VARCHAR(20)
AS
BEGIN
    UPDATE members
    SET full_name = @full_name, @telephone_number = @telephone_number, email_address = @email_address
    WHERE id = @member_id
END
GO
------------------------ STORE PROCEDURE TO UDATE MEMBER'S DETAIL ----------------------------
GO
CREATE PROCEDURE updateMemberStatus
    @member_id INT,
    @membership_start_date DATE,
    @membership_end_date DATE,
    @membership_suspended_date DATE
AS
BEGIN
    UPDATE membership_status
    SET membership_start_date = @membership_start_date, membership_end_date = @membership_end_date, membership_suspended_date = @membership_suspended_date
    WHERE member_id = @member_id
END
GO
```

Fig 17.

JOINING ALL MEMBER'S PROCEDURES USING TRANSACTIONS.

```
-----------------------Populating the tables with the create procedures----------------------
--------------------------ALL TRANSACTIONS PROCEDURES-------------------------------------------

------------------------------ JOINING ALL THE MEMBER'S PROCEDURES USING TRANSACTIONS -------------------

GO
CREATE PROCEDURE addNewMemberWithAddressAndAuthAndGenderAndMemberShipStatus
    @full_name VARCHAR(100),
    @date_of_birth DATE,
    @email_address VARCHAR(255) = NULL,
    @telephone_number VARCHAR(20) = NULL,
    @street_number INT,
    @street_address VARCHAR(200),
    @city VARCHAR(100),
    @state VARCHAR(100),
    @postcode VARCHAR(50),
    @gender VARCHAR(50),
    @username VARCHAR(50),
    @password BINARY(255),
    @membership_start_date DATE,
    @membership_end_date DATE = NULL,
    @membership_suspended_date DATE = NULL
AS
BEGIN
    BEGIN TRANSACTION;
    DECLARE @id INT;

    EXEC addNewMember
        @full_name = @full_name,
        @date_of_birth = @date_of_birth,
        @email_address = @email_address,
        @telephone_number = @telephone_number,
        @id = @id OUTPUT;

    SET @id = @id

    EXEC addNewAddress
        @street_number = @street_number,
        @member_id = @id,
        @street_address = @street_address,
        @city = @city,
        @state = @state,
        @postcode = @postcode;

     EXEC addNewMemberAuth
         @username = @username,
         @password = @password,
         @member_id = @id;

     EXEC addMemberShipStatus
         @member_id = @id,
         @membership_start_date = @membership_start_date,
         @membership_end_date = @membership_end_date,
         @membership_suspended_date = @membership_suspended_date;

    COMMIT TRANSACTION;
END
GO
```

Fig 18.

JOINING ALL CATALOGUE RELATED PROCEDURES USING TRANSACTIONS.

```
-------- JOINING ALL CATALOGUE RELATED PROCEDURES --------------
-- Publisher, publisher's address, catalog,
--item status, item type, author, author's address, item_author

GO
CREATE PROCEDURE insertAllCatalogueAssociatedTables
    -------publisher-----------
    @publisher_name VARCHAR(50),
    @publisher_telephone_number VARCHAR(20),
    @publisher_email VARCHAR(50),

    ------publisher's address-----
    @pulisher_street_number INT,
    @pulisher_street_address VARCHAR(200),
    @pulisher_city VARCHAR(100),
    @pulisher_state VARCHAR(100),
    @pulisher_postcode VARCHAR(50),

    --------author--------
    @author_first_name VARCHAR(100),
    @author_last_name VARCHAR(100),
    @author_date_of_birth DATE,
    @author_email_address VARCHAR(255),
    @author_telephone_number VARCHAR(20),

    -------author's address------
    @author_street_number INT,
    @author_street_address VARCHAR(200),
    @author_city VARCHAR(100),
    @author_state VARCHAR(100),
    @author_postcode VARCHAR(50),

    -------catalogue--------
    @item_title VARCHAR(255),
    @year_of_publication DATE,
    @date_added DATE,

    -------item type--------
    @item_type VARCHAR(50),
    @item_isbn VARCHAR(50) = NULL,
    @item_issn VARCHAR(50) = NULL,

    ------item status-------
    @item_status_value VARCHAR(50),
    @item_status_date DATE
AS
BEGIN
    BEGIN TRANSACTION;
    DECLARE @publisher_id INT;
    DECLARE @author_id INT;
    DECLARE @item_id INT;
```

```
    EXEC insertPublishers
        @publisher_name = @publisher_name,
        @telephone_number = @publisher_telephone_number,
        @email = @publisher_email,
        @publisher_id = @publisher_id OUTPUT;

        SET @publisher_id = @publisher_id

    EXEC addNewPublisherAddress
        @street_number = @pulisher_street_number,
        @street_address = @pulisher_street_address,
        @city = @pulisher_city,
        @state = @pulisher_state,
        @postcode = @pulisher_postcode,
        @publisher_id = @publisher_id
    EXEC insertItemIntoCatalog
        @item_title =  @item_title,
        @year_of_publication = @year_of_publication,
        @date_added = @date_added,
        @publisher_id = @publisher_id,
        @item_id = @item_id OUTPUT;

         SET @item_id = @item_id

    EXEC insertIntoItemType
        @item_id = @item_id,
        @item_type = @item_type,
        @item_isbn  = @item_isbn,
        @item_issn  = @item_issn
    EXEC insertIntoItemStatus
        @item_id = @item_id,
        @item_status_value = @item_status_value,
        @item_status_date = @item_status_date
    EXEC addNewAuthor
        @first_name = @author_first_name,
        @last_name = @author_last_name,
        @date_of_birth  = @author_date_of_birth,
        @email_address = @author_email_address,
        @telephone_number = @author_telephone_number,
        @author_id = @author_id OUTPUT;

         SET @author_id = @author_id

    EXEC addAuthorAddress
        @street_number = @author_street_number,
        @street_address = @author_street_address,
        @city = @author_city,
        @state = @author_state,
        @postcode = @author_postcode,
        @author_id = @author_id
    EXEC addNewItemAuthor
        @item_id = @item_id,
        @author_id = @author_id


    COMMIT TRANSACTION;
END
GO
```

Fig 19.

CATALOGUE, LOAN, PAYMENT AND AUTHOR PROCEDURES

```
-- ----------------------  STORE PROCEDURE TO INSERT A NEW ITEM INTO A CATALOG'S TABLE ----------------------------------

CREATE PROCEDURE insertItemIntoCatalog
    @item_title VARCHAR(255),
    @year_of_publication DATE,
    @date_added DATE,
    @publisher_id INT,
    @item_id INT OUTPUT
AS
BEGIN
    DECLARE @InsertedRows TABLE (item_id INT);

    INSERT INTO catalog
        (item_title, year_of_publication, date_added, publisher_id)
        OUTPUT inserted.item_id INTO @InsertedRows
    VALUES
        (@item_title, @year_of_publication, @date_added, @publisher_id);

    SELECT @item_id = item_id FROM @InsertedRows;
END
GO


----------------------  STORE PROCEDURE TO INSERT A ITEM TYPE IN ITEM TYPE TABLE ----------------------------------

GO
CREATE PROCEDURE insertIntoItemType
    @item_id INT,
    @item_type VARCHAR(50),
    @item_isbn VARCHAR(50) = NULL,
    @item_issn VARCHAR(50) = NULL
AS
BEGIN
    INSERT INTO item_type
        (item_id, item_type, item_isbn, item_issn)
    VALUES
        (@item_id, @item_type, @item_isbn, @item_issn)
END
GO

----------------------  STORE PROCEDURE TO INSERT A ITEM STATUS INTO A ITEM STATUS TABLE ------------------------------

GO
CREATE PROCEDURE insertIntoItemStatus
    @item_id INT,
    @item_status_value VARCHAR(50),
    @item_status_date DATE
AS
BEGIN
    INSERT INTO item_status
        (item_id, item_status_value, item_status_date)
    VALUES
        (@item_id, @item_status_value, @item_status_date)
END
GO
```

Fig 20.

```
GO
CREATE PROCEDURE insertIntoLoans
    @item_id INT,
    @date_taken_out DATE,
    @date_due_back DATE,
    @date_returned DATE = NULL,
    @status VARCHAR(20) = 'Not-Returned'
AS
BEGIN
    INSERT INTO loans
        (item_id, date_taken_out, date_due_back, date_returned, status)
    VALUES
        (@item_id, @date_taken_out, @date_due_back, @date_returned, @status)
END
GO
------------ create procedure for inserting payment fine into fine_payment table -------------

GO
CREATE PROCEDURE insertFine_payment
    @amount_paid DECIMAL(10,2) = 0.00,
    @member_id INT,
    @loan_id INT
AS
BEGIN
    INSERT INTO fine_payment
        (amount_paid, member_id, loan_id)
    VALUES
        (@amount_paid, @member_id, @loan_id)
END
GO

------------ create procedure for inserting payment method into payment method table ----------------

GO
CREATE PROCEDURE insertpayment_method
    @fine_payment_id INT,
    @fine_payment_method VARCHAR(50)
AS
BEGIN
    INSERT INTO payment_method
        (fine_payment_id, fine_payment_method)
    VALUES
        (@fine_payment_id, @fine_payment_method)
END
GO
```

Fig 21

```
------------------------- STORE PROCEDURE TO UDATE LOAN TABLE -----------------------------
GO
CREATE PROCEDURE loanItemStatus
    @loan_id INT,
    @date_taken_out DATE,
    @date_due_back DATE,
    @date_returned DATE = NULL,
    @status VARCHAR(20)
AS
BEGIN
    UPDATE loans
    SET date_taken_out = @date_taken_out, date_due_back = @date_due_back, date_returned = @date_returned, status = @status
    WHERE id = @loan_id
END
GO

---------------------  STORE PROCEDURE TO INSERT A NEW AUTHOR INTO THE AUTHOR'S TABLE ------------------------------------
GO
CREATE PROCEDURE addNewAuthor
    @first_name VARCHAR(100),
    @last_name VARCHAR(100),
    @date_of_birth DATE,
    @email_address VARCHAR(255),
    @telephone_number VARCHAR(20)
AS
BEGIN
    INSERT INTO authors
        (first_name, last_name, date_of_birth, email_address, telephone_number)
    VALUES
        (@first_name, @last_name, @date_of_birth, @email_address, @telephone_number)
END
GO

----------------------  STORE PROCEDURE TO INSERT AUTHOR ADDRESS INTO THE AUTHOR'S ADDRESS TABLE ------------------------------------

GO
CREATE PROCEDURE addAuthorAddress
    @street_number INT,
    @author_id INT,
    @street_address VARCHAR(200),
    @city VARCHAR(100),
    @state VARCHAR(100),
    @postcode VARCHAR(50)
AS
BEGIN
    INSERT INTO author_address
        (street_number, author_id, street_address, city, state, postcode)
    VALUES
        (@street_number, @author_id, @street_address, @city, @state, @postcode)
END
GO
```

Fig 22.

```
-------------------------- STORE PROCEDURE TO UDATE AUTHOR'S DETAIL -----------------------------

GO
CREATE PROCEDURE updateAuthorDetails
    @author_id INT,
    @first_name VARCHAR(100),
    @last_name VARCHAR(100),
    @date_of_birth DATE,
    @email_address VARCHAR(255),
    @telephone_number VARCHAR(20)
AS
BEGIN
    UPDATE authors
    SET first_name = @first_name, last_name = @last_name, @telephone_number = @telephone_number, email_address = @email_address
    WHERE id = @author_id
END
GO



----------------------  STORE PROCEDURE TO INSERT AN ITEM AND AUTHOR INTO THE ITEM-AUTHOR'S TABLE -------------------------------

GO
CREATE PROCEDURE addNewItemAuthor
    @item_id INT,
    @author_id INT
AS
BEGIN
    INSERT INTO item_author
        (item_id, author_id)
    VALUES
        (@item_id, @author_id)
END
GO
```

Fig 23.

ADDED PROCEDURES.

QUESTION 7.

```
------------------------ ADDED TABLES POCEDURE -------------------------
---------- MAKE RESERVATION PROCEDURE ----------

GO
CREATE PROCEDURE makeReservation
    @item_id INT,
    @reservation_date DATE,
    @member_id INT
AS
BEGIN
    INSERT INTO reservations
        (item_id, reservation_date, member_id)
    VALUES
        (@item_id, @reservation_date, @member_id)
END
GO

------------- SET RESERVATION STATUS ROCEDURE ---------
GO
CREATE PROCEDURE insertItemReservationStatus
    @reservation_id INT,
    @status VARCHAR(100)
AS
BEGIN
    INSERT INTO reservation_status
        (reservation_id, status)
    VALUES
        (@reservation_id, @status)
END
GO
-------------- UPDATE RESERVATION STATUS --------------
GO
CREATE PROCEDURE updateReservationStatus
    @reservation_id INT,
    @status VARCHAR(100)
AS
BEGIN
    UPDATE reservation_status
    SET status = @status
    WHERE reservation_id = @reservation_id
END
GO
```

Fig 24.

```
--------- INSERT ITEM REVIEW PROCEDURE -----------------
GO
CREATE PROCEDURE insertItemReview
    @item_id INT,
    @rating INT,
    @review_comment VARCHAR(1000)
AS
BEGIN
    INSERT INTO item_reviews
        (item_id, rating, review_comment)
    VALUES
        (@item_id, @rating, @review_comment)
END
GO

------------ INSERT AUTHOR REVIEW PROCEDURE -----------
GO
CREATE PROCEDURE insertAuthorReview
    @item_id INT,
    @rating INT,
    @review_comment VARCHAR(1000)
AS
BEGIN
    INSERT INTO author_reviews
        (author_id, rating, review_comment)
    VALUES
        (@item_id, @rating, @review_comment)
END
GO
------------ INSERT PUBLISHER PROCEDURE -----------
GO
CREATE PROCEDURE insertPublishers
    @publisher_name VARCHAR(50),
    @telephone_number VARCHAR(20),
    @email VARCHAR(50)
AS
BEGIN
    INSERT INTO publishers
        (publisher_name, telephone_number, email)
    VALUES
        (@publisher_name, @telephone_number, @email)
END
GO


------------ INSERT PUBLISHER ADDRESS PROCEDURE -----------

GO
CREATE PROCEDURE addNewPublisherAddress
    @street_number INT,
    @street_address VARCHAR(200),
    @publisher_id INT,
    @city VARCHAR(100),
    @state VARCHAR(100),
    @postcode VARCHAR(50)
AS
BEGIN
    INSERT INTO pulisher_address
        (street_number, street_address, publisher_id, city, state, postcode)
    VALUES
        (@street_number, @street_address, @publisher_id, @city, @state, @postcode)
END
GO
```

Fig 25.

VIEWS, FUNCTIONS AND TRIGGERS

```
-------- FUNCTION TO: search the catalog by title and sorts the results by publication date, while also joining the item_type and item_status tables ---
CREATE FUNCTION search_catalog_for_item_by_title(@title VARCHAR(255))
RETURNS TABLE
AS
RETURN (
    SELECT TOP 100 PERCENT
    c.item_id, c.item_title, t.item_isbn, t.item_issn, s.item_status_value, s.item_status_date, c.year_of_publication, t.item_type
FROM catalog c
    JOIN item_status s ON c.item_id = s.item_id
    JOIN item_type t ON c.item_id = t.item_id
WHERE c.item_title LIKE '%' + @title + '%'
ORDER BY c.year_of_publication DESC
);
GO


SELECT * FROM search_catalog_for_item_by_title('The Lord of the Rings');


-- ----- Return a full list of all items currently on loan which have a due date of less than or equals five days from the current date ----------

GO
CREATE PROCEDURE itemsOnLoanWhichDueIn5DaysOrLess
AS
BEGIN

    DECLARE @Today DATE
    SET @Today = GETDATE()

    SELECT c.item_title, c.year_of_publication, t.item_type,
          s.item_status_value, l.date_taken_out, l.date_due_back
    FROM loans AS l
        INNER JOIN catalog AS c ON l.item_id = c.item_id
        INNER JOIN item_type AS t ON t.item_id = c.item_id
        INNER JOIN item_status AS s ON s.item_id = c.item_id
    WHERE l.status = 'Not-Returned'
        AND l.date_due_back BETWEEN @Today AND DATEADD(DAY, 5, @Today)
    ORDER BY l.date_due_back ASC;
END
GO

--  ---- the loan history, showing all previous and currentloans, and including details of the item borrowed, borrowed date ---,
--- due date and any associated fines for each loan ------

GO
CREATE VIEW loan_history
AS
    SELECT c.item_title, it.item_type, it.item_isbn, it.item_issn, ist.item_status_value, l.date_taken_out, l.date_due_back, l.date_returned,
        l.status, f.amount_paid
    FROM catalog c
        INNER JOIN item_type it ON c.item_id = it.item_id
        INNER JOIN item_status ist ON c.item_id = ist.item_id
        INNER JOIN loans l ON c.item_id = l.item_id
        LEFT JOIN fine_payment f ON l.id = f.loan_id
GO

-- ------ select query which allows the library toidentify the total number of loans made on a specified date ---------------

SELECT COUNT(*) as total_loans
FROM loans
WHERE date_taken_out = '2023-04-13'


----- trigger that update the current status of an item automatically to Available when the book is returned ---------
GO
CREATE TRIGGER update_item_status
ON loans
AFTER UPDATE
AS
BEGIN
    UPDATE item_status
  SET item_status_value = 'Available'
  FROM item_status
        INNER JOIN inserted ON item_status.item_id = inserted.item_id
  WHERE inserted.status = 'Returned';
END;
GO
```

column item_issn(varchar, null)

Fig 26.

ADDED TRIGGERS

```
----------------------------ADDED TRIGGERS-----------------------------------

-----trigger that calculate numbers of book written by an author------
GO
CREATE TRIGGER no_of_books_written
ON item_author
AFTER INSERT
AS
BEGIN
    UPDATE authors
  SET no_of_books_written = no_of_books_written + 1
  WHERE id = (SELECT author_id
    FROM inserted)
END
GO


-----trigger that calculate publisher numbers of publication------
GO
CREATE TRIGGER update_number_of_publications
ON catalog
AFTER INSERT
AS
BEGIN
    UPDATE publishers
  SET number_of_publications = number_of_publications + 1
  WHERE id = (SELECT publisher_id
    FROM inserted)
END
GO


----- trigger that add total fines to the loan, every number of days after due when an item is still yet to be returned --
GO
CREATE TRIGGER add_overdue_fee
ON loans
AFTER UPDATE
AS
BEGIN
    DECLARE @member_id INT
    DECLARE @total_fines DECIMAL(10,2)
    DECLARE @outstanding_balance DECIMAL(10,2)
    DECLARE @date_due_back DATE
    DECLARE @status VARCHAR(20)

    SELECT @member_id = member_id, @total_fines = total_fines
    FROM inserted

    IF (DATEDIFF(day, @date_due_back, GETDATE()) > 0 AND @status = 'Not-Returned')
 BEGIN
        DECLARE @new_due_date DATE
        SET @new_due_date = GETDATE()
        SET @total_fines = @total_fines + 10.00
        SET @date_due_back = @new_due_date

        UPDATE member_fine SET total_fines = total_fines + 10.00,
        outstanding_balance = outstanding_balance + 10.00 WHERE member_id = @member_id
    END
END
GO
```

Fig 27.

```sql
-- ------------ Trigger that substract the total fine from loan table and set the status as returned
-----  and inturns update total member's fine table when fine payment is made -------
GO
CREATE TRIGGER update_fine_payment
ON fine_payment
AFTER INSERT
AS
BEGIN
    DECLARE @member_id INT, @loan_id INT, @amount_paid DECIMAL(10,2);  column amount_paid(decimal, not null)
    SELECT @member_id = member_id, @loan_id = loan_id, @amount_paid = amount_paid
    FROM inserted;

    UPDATE loans
  SET total_fines = total_fines - @amount_paid,
      date_returned = GETDATE(),
      status = 'Returned'
  WHERE id = @loan_id;

    UPDATE member_fine
  SET amount_paid = amount_paid + @amount_paid,
      outstanding_balance = total_fines - amount_paid
  WHERE member_id = @member_id;
END
GO
--- a function that search through the author, return their books, along with the ratings(both author and book ratinngs) sorted by year of publication
GO
CREATE PROCEDURE search_books_by_author
    @first_name VARCHAR(100),
    @last_name VARCHAR(100)
AS
BEGIN
    SELECT c.item_title, c.year_of_publication, ia.author_id, ir.rating AS item_rating, ar.rating AS author_rating
    FROM catalog c
        INNER JOIN item_author ia ON c.item_id = ia.item_id
        INNER JOIN authors a ON a.id = ia.author_id
        LEFT JOIN item_reviews ir ON c.item_id = ir.item_id
        LEFT JOIN author_reviews ar ON a.id = ar.author_id
        INNER JOIN item_type it ON c.item_id = it.item_id
        INNER JOIN item_status ist ON c.item_id = ist.item_id
    WHERE a.first_name LIKE '%' + @first_name + '%'
        AND a.last_name LIKE '%' + @last_name + '%'
    ORDER BY c.year_of_publication ASC
END
```

Fig 28.

ADDED TABLES, PROCEDURES AND TRIGGERS.

QUESTION 7: Explanation of functionalities of added features

Based on the entity relationship diagram, I was able to add some **additional database tables** which I feel from my designing thinking process will benefit the system.

Reservation table: This allows the members to make reservations for items in the catalogue.

Author's review table: This enables member (anonymous) to review and rate authors based on their work.

Item's review table: This enables library member (anonymous) to review and rate catalogue items

Publisher's table: This table is to keep track of publishers for each item in the catalogue.

The **added procedures** are shown in **fig 23** and **24**, which includes the procedures for inserting values into the reservation's, author's, item's review and publisher table.

Fig **26** and **27** shows the system **additional triggers**.

No of books written: This trigger automatically increments the numbers of book written by an author on insert to the item_author's table, which is an intermediary table between item and author. This prevents us from manually doing the increment.

Update number of publication: This trigger automatically increments the numbers of book published by a publisher on item insert into the catalogue.

Add overdue fee: This trigger adds overdue fees of loan item that are due, if the loan item status remains not returned and this also update the member's fine table which keeps record of user's total loan fines and payment.

Update fine payment: This trigger calculates the fine and member's fine payment on every payment towards any loan fine.

Search books by author: This displays all author's review and books written by an author, by searching for the author's name

INSERTING RECORDS TO TABLES

QUESTION 6 AND QUESTION 2C.

```
----------------INSERTING DATA----------------------------


EXEC insertAllCatalogueAssociatedTables
    @publisher_name = 'ABC Publisher',
    @publisher_telephone_number = '555-1234',
    @publisher_email = 'info@abcpublisher.com',
    @pulisher_street_number = 123,
    @pulisher_street_address = '123 Main St',
    @pulisher_city = 'Vegas',
    @pulisher_state = 'CA',
    @pulisher_postcode = '12345',
    @item_title = 'The Great Gatsby',
    @year_of_publication = '2022-01-01',
    @date_added = '2022-01-15',
    @item_type = 'book',
    @item_isbn = '978-1-2345-6789-0',
    @item_status_value = 'Available',
    @item_status_date = '2022-01-15',
    @author_first_name = 'John',
    @author_last_name = 'Doe',
    @author_date_of_birth = '1980-01-01',
    @author_email_address = 'john@example.com',
    @author_telephone_number = '898-567',
    @author_street_number = 456,
    @author_street_address = '456 Main St',
    @author_city = 'Luton',
    @author_state = 'CA',
    @author_postcode = '12345';
```

Fig 29: Inserting catalogue's data and the catalogue's related data using the **insertAllCatalogueAssociatedTables** procedure from **fig 19**

```sql
SELECT full_name, date_of_birth, email_address,
    telephone_number, street_address,
    street_number, city, [state], postcode,
    username, [password], gender, membership_start_date, membership_end_date
    membership_suspended_date
FROM members m
JOIN member_address ma ON m.id = ma.member_id
JOIN member_auth mauth ON m.id = mauth.member_id
JOIN gender g ON m.id = g.member_id
JOIN membership_status ms ON m.id = ms.member_id;
```

Results Messages

	full_name	date_of_birth	email_address	telephone_number	street_address	street_number	city	state	postcode	username	password	gender
1	John Smith	1990-05-12	john.smith@example.com	123-456-7890	Main St	123	Anytown	NY	12345	johnsmith	0x1E8C5F20D86431ED4D4C7A5…	Male
2	Sarah Johnson	1985-11-02	sarah.johnson@example.com	555-555-5555	Broadway	456	Anytown	CA	54321	sjohnson	0x1E8C5F20D86431ED4D4C7A5…	Female
3	Michael Lee	1987-09-20	michael.lee@example.com	777-777-7777	Park Ave	789	Anytown	FL	98765	mlee	0x1E8C5F20D86431ED4D4C7A5…	Male
4	Emily Wilson	1995-02-14	emily.wilson@example.com	NULL	Main St	456	Anytown	TX	67890	ewilson	0x1E8C5F20D86431ED4D4C7A5…	Female
5	David Brown	1993-11-22	david.brown@example.com	555-9876	Park Ave	789	Chicago	IL	60601	davidbrown	0x1E8C5F20D86431ED4D4C7A5…	Male
6	Karen Taylor	1978-06-05	karen.taylor@example.com	555-6789	State St	321	Boston	MA	02101	karentaylor	0x1E8C5F20D86431ED4D4C7A5…	Female
7	Mila Kunis	1983-08-14	mila.kunis@example.com	555-555-1234	Main St.	123	Los Angeles	California	90001	milak	0x7C5A2A474053F20BB0CCED3…	Female
8	Saoirse Ronan	1994-04-12	saoirse.ronan@example.com	555-555-5678	Oak St.	456	New York	New York	10001	saoirser	0xD29EAE99FCFDB6371EE0EBF…	Female
9	Tom Hardy	1977-09-15	tom.hardy@example.com	555-555-9101	Park Ave.	789	London		EC1Y 8SY	tomh	0x67B7B2F78BB9E9D0B8C0399…	Male
10	Emma Stone	1988-11-06	emma.stone@example.com	555-555-1212	1st St.	101	Los Angeles	California	90001	emmast	0x1E8C5F20D86431ED4D4C7A5…	Female
11	Robert Davis	1980-12-05	robert.davis@gmail.com	666-9876	Elm St	789	New York	NY	10001	robert.dav…	0x1E8C5F20D86431ED4D4C7A5…	Male
12	Emily Chen	1995-08-11	emily.chen@gmail.com	555-4321	Cedar St	246	Boston	MA	02108	emily.chen	0x1E8C5F20D86431ED4D4C7A5…	Female

Fig 30: Inserted member's information with join statement

```sql
EXEC insertAllCatalogueAssociatedTables
@publisher_name = 'Pride Publishing',
@publisher_telephone_number = '0985-5678',
@publisher_email = 'pridel@xyzpublishing.com',
@pulisher_street_number = 561,
@pulisher_street_address = '430 morrison St',
@pulisher_city = 'Buton',
@pulisher_state = 'BT',
@pulisher_postcode = '9876',
@item_title = 'Pride and Prejudice',
@year_of_publication = '2022-02-22',
@date_added = '2023-04-18',
@item_type = 'book',
@item_isbn = '1234-5678',
@item_status_value = 'not available',
@item_status_date = '2022-02-15',
@author_first_name = 'camila',
@author_last_name = 'camilo',
@author_date_of_birth = '1985-01-01',
@author_email_address = 'camilo@example.com',
@author_telephone_number = '088-699',
@author_street_number = 456,
@author_street_address = '450 Jones St',
@author_city = 'London',
@author_state = 'LD',
@author_postcode = '1299';
```

Fig 31: inserting item and all item related objects using **insertAllCatalogueAssociatedTables** procedure from **fig 19**

```
--- join query from all catalogue related items -----
SELECT
    pub.id, pub.publisher_name, pub.number_of_publications,
    pub.telephone_number, pub.email, pub_address.street_number,
    pub_address.publisher_id, pub_address.street_address, pub_address.city,
    pub_address.state, pub_address.postcode, cat.item_id,
    cat.item_title, cat.year_of_publication, cat.date_added,
    cat.publisher_id, item_stat.item_id, item_stat.item_status_value,
    item_stat.item_status_date, item_typ.item_id, item_typ.item_type,
    item_typ.item_isbn, item_typ.item_issn, aut.id,
    aut.first_name, aut.last_name, aut.no_of_books_written,
    aut.date_of_birth, aut.email_address, aut.telephone_number,
    aut_address.street_number, aut_address.street_address,
    aut_address.city, aut_address.state, aut_address.postcode, item_aut.id
FROM
    publishers AS pub
    JOIN pulisher_address AS pub_address ON pub.id = pub_address.publisher_id
    JOIN catalog AS cat ON pub.id = cat.publisher_id
    JOIN item_status AS item_stat ON cat.item_id = item_stat.item_id
    JOIN item_type AS item_typ ON cat.item_id = item_typ.item_id
    JOIN item_author AS item_aut ON cat.item_id = item_aut.item_id
    JOIN authors AS aut ON item_aut.author_id = aut.id
    JOIN author_address AS aut_address ON aut.id = aut_address.author_id;
```

esults Messages

id	publisher_name	number_of_publications	telephone_number	email	street_number	publisher_id	street_address	city	state	postcode	item_id	item_ti
1	ABC Publisher	1	555-1234	info@abcpublisher.com	123	1	123 Main St	Vegas	CA	12345	1	The Gr
2	IJK Publisher	1	555-1299	ijk@abcpublisher.com	123	2	123 Main St	Seatle	CA	12345	2	The Ma
3	XYZ Publishing	1	555-5678	unique@xyzpublishing.com	456	3	456 Main St	bolton	CA	12345	3	The Bo
4	UTC Publishing	1	585-5678	utce@xyzpublishing.com	470	4	456 Peter St	liverpool	TX	9876	4	The Fa
5	DBOOK Publishing	1	444-5678	dbooke@xyzpublishing.com	560	5	458 Johnson St	Newcastle	NW	9876	5	Gone G
6	LABEL Publishing	1	005-5678	myemail@xyzpublishing.com	561	6	430 morrison St	Buton	BT	9876	6	Angels
7	WINNERS Publishing	1	885-5678	jobmail@xyzpublishing.com	589	7	430 morrison St	Buton	BT	9876	7	Game o
8	Movie Publishing	1	266-5678	mymovie@xyzpublishing.com	561	8	430 morrison St	Buton	BT	9876	8	The Lo
9	Pride Publishing	1	0985-5678	pridel@xyzpublishing.com	561	9	430 morrison St	Buton	BT	9876	9	Pride
12	QDOT Publishing	1	500-5678	qdot@xyzpublishing.com	561	12	430 morrison St	Newcastle	NW	9876	12	The Ca
13	ikigai Publishing	1	777-5678	ikigai@xyzpublishing.com	561	13	430 morrison St	Tokyo	TK	9876	13	Ikigai

```
    JOIN authors AS aut ON item_aut.author_id = aut.id
    JOIN author_address AS aut_address ON aut.id = aut_address.author_id;
```

Results Messages

year_of_publication	date_added	publisher_id	item_id	item_status_value	item_status_date	item_id	item_type	item_isbn	item_issn	id	first_name	last_name	no_of_b
2022-01-01	2022-01-15	1	1	Available	2022-01-15	1	book	978-1-2345-6789-0	NULL	1	John	Doe	1
2022-01-01	2022-01-15	2	2	Available	2022-01-15	2	book	978-1-2345-6789-0	NULL	2	Peter	harrison	1
2022-02-01	2022-02-15	3	3	Available	2022-02-15	3	dvd	NULL	1234-5678	3	peter	pan	1
2023-02-11	2022-04-16	4	4	Available	2022-02-15	4	dvd	NULL	1234-5678	4	samson	sunday	1
2022-02-22	2023-04-18	5	5	Available	2022-02-15	5	book	1234-5678	NULL	5	abiola	moshood	1
2022-02-22	2023-04-18	6	6	Available	2022-02-15	6	book	1234-5678	NULL	6	paul	jack	1
2022-02-22	2023-04-18	7	7	Available	2022-02-15	7	book	1234-5678	NULL	7	john	snow	1
2022-02-22	2023-04-18	8	8	Available	2022-02-15	8	book	1234-5678	NULL	8	saka	bukayo	1
2022-02-22	2023-04-18	9	9	not available	2022-02-15	9	book	1234-5678	NULL	9	camila	camilo	1
2022-02-22	2023-04-18	12	12	Available	2022-02-15	12	book	1234-5678	NULL	12	catcher	rey	1
2022-02-22	2023-04-18	13	13	not available	2022-02-15	13	book	1234-5678	NULL	13	mimi	aiko	1

item_issn	id	first_name	last_name	no_of_books_written	date_of_birth	email_address	telephone_number	street_number	street_address	city	state	postcode	id
NULL	1	John	Doe	1	1980-01-01	john@example.com	898-567	456	456 Main St	Luton	CA	12345	1
NULL	2	Peter	harrison	1	1980-01-01	peter@example.com	909-567	456	456 Main St	Lacanshire	CA	12345	2
1234-5678	3	peter	pan	1	1985-01-01	pan@example.com	456-678	456	456 Main St	Alabama	CA	12345	3
1234-5678	4	samson	sunday	1	1985-01-01	sun@example.com	1274-678	456	459 Main St	shicago	SH	12387	4
NULL	5	abiola	moshood	1	1985-01-01	abiola@example.com	555-678	456	450 Jones St	London	LD	1299	5
NULL	6	paul	jack	1	1985-01-01	jack@example.com	456-699	456	450 Jones St	London	LD	1299	6
NULL	7	john	snow	1	1985-01-01	snow@example.com	000-699	456	450 Jones St	London	LD	1299	7
NULL	8	saka	bukayo	1	1985-01-01	saka@example.com	299-699	456	450 Jones St	London	LD	1299	8
NULL	9	camila	camilo	1	1985-01-01	camilo@example.com	088-699	456	450 Jones St	London	LD	1299	9
NULL	12	catcher	rey	1	1985-01-01	rey@example.com	533-600	456	450 Jones St	London	LD	1299	12
NULL	13	mimi	aiko	1	1985-01-01	aiko@example.com	444-600	456	450 Jones St	Tokyo	TK	1299	13

Fig 32: Bringing All the **catalogue's item related tables together** based on relationships using **join**. This table is a result of joining the **item table** with the **publisher's table** and publisher's table with the **publisher's address**, and the **item's** table with the **item_author** table, and with the **author's** table and **author's address** table.

QUESTION 2A

```
-------------------------FUNTIONS, VIEWS, AND TRIGGERS ----------------------------------

-------- FUNCTION TO: search the catalog by title and sorts the results by publication date, while also joining the item_type and item_status tables ----

CREATE FUNCTION search_catalog_for_item_by_title(@title VARCHAR(255))
RETURNS TABLE
AS
RETURN (
    SELECT TOP 100 PERCENT
    c.item_id, c.item_title, t.item_isbn, t.item_issn, s.item_status_value, s.item_status_date, c.year_of_publication, t.item_type
FROM catalog c
    JOIN item_status s ON c.item_id = s.item_id
    JOIN item_type t ON c.item_id = t.item_id
WHERE c.item_title LIKE '%' + @title + '%'
ORDER BY c.year_of_publication DESC
);
GO


SELECT * FROM search_catalog_for_item_by_title('The Lord of the Rings');
```

lts Messages

item_id	item_title	item_isbn	item_issn	item_status_value	item_status_date	year_of_publication	item_type
8	The Lord of the Rings	1234-5678	*NULL*	Available	2022-02-15	2022-02-22	book

Fig 33: Search catalogue by title.

```
1310  --- loan items query ----------
1311  SELECT full_name,  total_fines, date_taken_out,
1312        status, item_title, date_due_back, date_returned
1313  FROM
1314  loans l
1315  JOIN members m ON l.member_id = m.id
1316  JOIN catalog ca ON l.item_id = ca.item_id;
1317
```

Results Messages

	full_name	total_fines	date_taken_out	status	item_title	date_due_back	date_returned
1	John Smith	0.00	2023-04-10	Not-Returned	The Great Gatsby	2023-04-24	*NULL*
2	Sarah Johnson	0.00	2023-04-12	Not-Returned	The Maze Runner	2023-04-25	*NULL*
3	John Smith	0.00	2023-04-15	Not-Returned	The Book Thief	2023-04-23	*NULL*
4	Michael Lee	0.00	2023-04-17	Not-Returned	The Fault in Ou…	2023-04-26	*NULL*
5	John Smith	0.00	2023-04-20	Not-Returned	Gone Girl	2023-04-24	*NULL*
6	John Smith	0.00	2023-04-22	Not-Returned	Angels and Demo…	2023-04-26	*NULL*
7	Karen Taylor	0.00	2023-04-25	Not-Returned	Game of thrones	2023-04-22	*NULL*
8	John Smith	0.00	2023-04-27	Not-Returned	The Lord of the…	2023-04-24	*NULL*
9	David Brown	0.00	2023-04-30	Not-Returned	Pride and Preju…	2023-05-25	*NULL*
10	Emily Wilson	0.00	2023-05-01	Not-Returned	Ikigai	2023-05-08	*NULL*

Fig 34: Table showing items on loan.

QUESTION 2B.

```
1018
1019
1020  -- ----- Return a full list of all items currently on loan which have a due date of less than or equals five days from the current date ----------
1021
1022  GO
1023  CREATE PROCEDURE itemsOnLoanWhichDueIn5DaysOrLess
1024  AS
1025  BEGIN
1026
1027      DECLARE @Today DATE
1028      SET @Today = GETDATE()
1029
1030      SELECT c.item_title, c.year_of_publication, t.item_type,
1031          s.item_status_value, l.date_taken_out, l.date_due_back
1032      FROM loans AS l
1033          INNER JOIN catalog AS c ON l.item_id = c.item_id
1034          INNER JOIN item_type AS t ON t.item_id = c.item_id
1035          INNER JOIN item_status AS s ON s.item_id = c.item_id
1036      WHERE l.status = 'Not-Returned'
1037          AND l.date_due_back BETWEEN @Today AND DATEADD(DAY, 5, @Today)
1038      ORDER BY l.date_due_back ASC;
1039  END
1040  GO
1041
1042
1043  EXEC itemsOnLoanWhichDueIn5DaysOrLess;
1044
```

Results Messages

	item_title	year_of_publication	item_type	item_status_value	date_taken_out	date_due_back
1	The Book Thief	2022-02-01	dvd	Available	2023-04-15	2023-04-23
2	The Great Gatsby	2022-01-01	book	Available	2023-04-10	2023-04-24
3	The Lord of the Rings	2022-02-22	book	Available	2023-04-27	2023-04-24
4	Gone Girl	2022-02-22	book	Available	2023-04-20	2023-04-24
5	The Maze Runner	2022-01-01	book	Available	2023-04-12	2023-04-25
6	The Fault in Our Stars	2023-02-11	dvd	Available	2023-04-17	2023-04-26
7	Angels and Demons	2022-02-22	book	Available	2023-04-22	2023-04-26
8	Angels and Demons	2022-02-22	book	Available	2023-04-22	2023-04-26
9	Angels and Demons	2022-02-22	book	Available	2023-04-22	2023-04-26

Fig 35: Table showing items on loan which are due in **5days or less**, with today's date **2023-04-21**.

QUESTION 5

```
987
988   -- ------ select query which allows the library toidentify the total number of loans made on a specified date ----------
989
990   SELECT COUNT(*) as total_loans
991   FROM loans
992   WHERE date_taken_out = '2023-04-22'
993
```

sults Messages

total_loans
1

Fig 36: Total numbers of loans taken out on **2023-04-22**

QUESTION 3

```
967
968   --  ---- the loan history, showing all previous and currentloans, and including details of the item borrowed, borrowed date ---,
969   --- due date and any associated fines for each loan ------
970
971   GO
972   CREATE VIEW loan_history
973   AS
974       SELECT c.item_title, it.item_type, it.ite  column date_returned(date, null)  tus_value,
975           l.date_taken_out, l.date_due_back, l.date_returned,
976           l.status, l.total_fines, COALESCE(SUM(f.amount_paid), 0.00) AS total_amount_paid_fine_on_item_loan
977       FROM catalog c
978           INNER JOIN item_type it ON c.item_id = it.item_id
979           INNER JOIN item_status ist ON c.item_id = ist.item_id
980           INNER JOIN loans l ON c.item_id = l.item_id
981           LEFT JOIN fine_payment f ON l.id = f.loan_id
982       GROUP BY c.item_title, it.item_type, it.item_isbn, it.item_issn, ist.item_status_value,
983           l.date_taken_out, l.date_due_back, l.date_returned, l.status, l.total_fines;
984   GO
985
986   SELECT * FROM loan_history;
987
988   -- ------ select query which allows the library toidentify the total number of loans made on a specified date ---------------
```

Results Messages

	item_title	item_type	item_isbn	item_issn	item_status_value	date_taken_out	date_due_back	date_
1	Angels and Demons	book	1234-5678	NULL	Available	2023-04-22	2023-04-26	NULL
2	Game of thrones	book	1234-5678	NULL	Available	2023-04-25	2023-04-22	NULL
3	Gone Girl	book	1234-5678	NULL	Available	2023-04-20	2023-04-24	NULL
4	Ikigai	book	1234-5678	NULL	not available	2023-05-01	2023-05-08	NULL
5	Pride and Prejudice	book	1234-5678	NULL	not available	2023-04-30	2023-05-25	NULL
6	The Book Thief	dvd	NULL	1234-5678	Available	2023-04-15	2023-04-23	NULL
7	The Fault in Our Stars	dvd	NULL	1234-5678	Available	2023-04-17	2023-04-26	NULL
8	The Great Gatsby	book	978-1-2345-6789-0	NULL	Available	2023-04-10	2023-04-24	NULL
9	The Lord of the Rings	book	1234-5678	NULL	Available	2023-04-27	2023-04-24	NULL
10	The Maze Runner	book	978-1-2345-6789-0	NULL	Available	2023-04-12	2023-04-25	NULL

status	total_fines	total_amount_paid_fine_on_item_loan
Not-Returned	0.00	0.00
Not-Returned	0.00	0.00
Not-Returned	0.00	0.00
Not-Returned	0.00	0.00
Not-Returned	0.00	0.00
Not-Returned	0.00	0.00
Not-Returned	0.00	0.00
Not-Returned	0.00	0.00
Not-Returned	0.00	0.00
Not-Returned	0.00	0.00

Fig 36: Loan histories with the total due fines and total amount paid from the fine.

QUESTION 4

```
GO
CREATE TRIGGER update_item_status
ON loans
AFTER UPDATE
AS
BEGIN
    IF UPDATE (date_returned)
    BEGIN
    UPDATE item_status
  SET item_status_value = 'Available'
  FROM item_status
       INNER JOIN inserted ON item_status.item_id = inserted.item_id
  WHERE inserted.date_returned IS NOT NULL;
  END
END;
GO

EXEC loanItemStatus
    @loan_id = 19,
    @date_taken_out = '2023-04-20',
    @date_due_back = '2023-05-20',
    @date_returned = '2023-04-21',
    @status = 'Returned';

SELECT * from item status
```

esults Messages

id	item_id	item_status_value	item_status_date
1	1	Available	2022-01-15
2	2	Available	2022-01-15
3	3	Available	2022-02-15
4	4	Available	2022-02-15
5	5	Available	2022-02-15
6	6	Available	2022-02-15
7	7	Available	2022-02-15
8	8	Available	2022-02-15
9	9	Available	2022-02-15
11	11	Available	2022-02-15
12	12	Available	2022-02-15
13	13	not available	2022-02-15

Fig 37: Loan with **loan ID of 19** and **item ID of 9** with **item status of not available** is now back at **available** once the item **table date_returned** column is **updated on** item returned.

QUESTION 2D

```
---------- single update procedure ------------------
CREATE PROCEDURE updateMemberInformation
    @member_id INT,
    @full_name VARCHAR(100),
    @date_of_birth DATE,
    @email_address VARCHAR(255),
    @telephone_number VARCHAR(20),
    @membership_start_date DATE,
    @membership_end_date DATE,
    @membership_suspended_date DATE,
    @street_number INT,
    @street_address VARCHAR(200),
    @city VARCHAR(100),
    @state VARCHAR(100),
    @postcode VARCHAR(50)
AS
BEGIN
    BEGIN TRANSACTION
    -- Update the full name, telephone number, email address in members table ---
    ---By calling the UPDATE members status procedure--------
    UPDATE members
    SET full_name = @full_name, telephone_number = @telephone_number,
    email_address = @email_address
    WHERE id = @member_id

    -- Update the memebership start date, membership end date and membership suspended date in membership_status table --
    ---By calling the UPDATE membership status procedure--------
    UPDATE membership_status
    SET membership_start_date = @membership_start_date,
    membership_end_date = @membership_end_date,
    membership_suspended_date = @membership_suspended_date
    WHERE member_id = @member_id

    -- Update member_address table by updating the street number, street address, city and state-----
    ----- by calling the member_address procedure---
    UPDATE member_address
    SET
    street_number = @street_number,
    street_address = @street_address,
    city = @city,
    state = @state,
    postcode = @postcode
    WHERE member_id = @member_id

    -- Roll back all the changes in case of an error -----
    IF @@ERROR <> 0
    BEGIN
        ROLLBACK TRANSACTION
        RETURN
    END
    -- If the operation was successful, commit all changes------
    COMMIT TRANSACTION
END
```

Fig 38: Update all member details procedure

QUESTION 2D

```
2233
2234    ----------------------UPDATE MEMBER'S DETAILS-----------
2235
2236    EXEC updateMemberInformation
2237        @member_id = 12,
2238        @full_name = 'Mrs Emily chen',
2239        @date_of_birth = '1992-06-30',
2240        @email_address = 'Alan@Salford.co.uk',
2241        @telephone_number = '555-4321',
2242        @membership_start_date = '2022-04-01',
2243        @membership_end_date = '2022-12-31',
2244        @membership_suspended_date = NULL,
2245        @street_number = 246,
2246        @street_address = 'Cedar St',
2247        @city = 'Boston',
2248        @state = 'VA',
2249        @postcode = '300'
2250
```

Results Messages

id	full_name	date_of_birth	email_address	telephone_number
1	John Smith	1990-05-12	john.smith@example.com	123-456-7890
2	Sarah Johns…	1985-11-02	sarah.johnson@example…	555-555-5555
3	Michael Lee	1987-09-20	michael.lee@example.c…	777-777-7777
4	Emily Wilson	1995-02-14	emily.wilson@example.…	*NULL*
5	David Brown	1993-11-22	david.brown@example.c…	555-9876
6	Karen Taylor	1978-06-05	karen.taylor@example.…	555-6789
7	Mila Kunis	1983-08-14	mila.kunis@example.com	555-555-1234
8	Saoirse Ron…	1994-04-12	saoirse.ronan@example…	555-555-5678
9	Tom Hardy	1977-09-15	tom.hardy@example.com	555-555-9101
10	Emma Stone	1988-11-06	emma.stone@example.com	555-555-1212
11	Robert Davis	1980-12-05	robert.davis@gmail.com	666-9876
12	Mrs Emily c…	1995-08-11	Alan@Salford.co.uk	555-4321
13	Jonathan da…	1990-05-12	jonathan.smith@exampl…	000-456-7890

Fig 39: Updating member with id of 12 details

```
2104
2105
2106    ---item reservation----
2107    EXEC makeReservation 1, '2023-04-22', 1
2108    EXEC makeReservation 2, '2023-04-23', 2
2109    EXEC makeReservation 1, '2023-04-24', 3
2110    EXEC makeReservation 3, '2023-04-25', 4
2111    EXEC makeReservation 4, '2023-04-26', 5
2112    EXEC makeReservation 2, '2023-04-27', 6
2113    EXEC makeReservation 5, '2023-04-28', 7
2114    EXEC makeReservation 3, '2023-04-29', 8
2115    EXEC makeReservation 4, '2023-04-30', 9
2116    EXEC makeReservation 5, '2023-05-01', 10
2117
2118
2119    ----item reservation status---
2120    EXEC insertItemReservationStatus 1, 'Pending'
2121    EXEC insertItemReservationStatus 2, 'Approved'
2122    EXEC insertItemReservationStatus 3, 'Canceled'
2123    EXEC insertItemReservationStatus 4, 'Pending'
2124    EXEC insertItemReservationStatus 5, 'Approved'
2125    EXEC insertItemReservationStatus 6, 'Canceled'
2126    EXEC insertItemReservationStatus 7, 'Pending'
2127    EXEC insertItemReservationStatus 8, 'Approved'
2128    EXEC insertItemReservationStatus 9, 'Canceled'
2129    EXEC insertItemReservationStatus 10, 'Pending'
2130
2131    SELECT
2132        reservation_date,
2133        full_name,
2134        [status],
2135        item_title,
2136        item_type
2137    FROM reservations r
2138        JOIN members m ON r.member_id = m.id
2139        JOIN [catalog] c ON c.item_id = c.item_id
2140        JOIN reservation_status rs ON r.id = rs.reservation_id
2141        JOIN item_type it ON it.item_id = c.item_id
2142
```

Results Messages

	reservation_date	full_name	status	item_title	item_type
1	2023-04-22	John Smith	Pending	The Great Gatsby	book
2	2023-04-22	John Smith	Pending	The Maze Runner	book
3	2023-04-22	John Smith	Pending	The Book Thief	dvd
4	2023-04-22	John Smith	Pending	The Fault in Our Stars	dvd
5	2023-04-22	John Smith	Pending	Gone Girl	book
6	2023-04-22	John Smith	Pending	Angels and Demons	book
7	2023-04-22	John Smith	Pending	Game of thrones	book
8	2023-04-22	John Smith	Pending	The Lord of the Rings	book
9	2023-04-22	John Smith	Pending	Pride and Prejudice	book

Fig 40: Making item reservation and fetching all reservations.

```
---insert item review---

EXEC insertItemReview  6, 4, 'Great book!'
EXEC insertItemReview  7, 5, 'Excellent read!'
EXEC insertItemReview  8, 3, 'Average book'
EXEC insertItemReview 9, 2, 'Not impressed with the plot'
EXEC insertItemReview  10, 4, 'Good characters and pacing'
EXEC insertItemReview  11, 4, 'Satisfied with the story'
EXEC insertItemReview  12, 1, 'Terrible book'
EXEC insertItemReview 3, 5, 'Impressed with the writing style'
EXEC insertItemReview  4, 2, 'Disappointing ending'
EXEC insertItemReview 5, 3, 'Could be better'


--author's review-----

EXEC insertAuthorReview 1, 4, 'Great author!'
EXEC insertAuthorReview 2, 5, 'Excellent writing skills!'
EXEC insertAuthorReview 1, 3, 'Average author'
EXEC insertAuthorReview 3, 2, 'Not impressed with the storytelling'
EXEC insertAuthorReview 4, 4, 'Good style and approach'
EXEC insertAuthorReview 2, 4, 'Satisfied with the book'
EXEC insertAuthorReview 5, 1, 'Terrible author'
EXEC insertAuthorReview 3, 5, 'Impressed with the creativity'
EXEC insertAuthorReview 4, 2, 'Disappointing works'
EXEC insertAuthorReview 5, 3, 'Could be better'
```

```
--- a function that search through the author, return their books, along with the average ratings(both author and book ratinngs) sorted by year of publication
GO
CREATE PROCEDURE search_books_by_author
    @first_name VARCHAR(100),
    @last_name VARCHAR(100)
AS
BEGIN
    SELECT c.item_title, c.year_of_publication, ia.author_id, ir.review_comment AS item_reviews, ar.review_comment AS author_reviews,
           (SELECT AVG(rating) FROM item_reviews WHERE item_id = c.item_id) AS item_rating,
           (SELECT AVG(rating) FROM author_reviews WHERE author_id = ia.author_id) AS author_rating
    FROM catalog c
        INNER JOIN item_author ia ON c.item_id = ia.item_id
        INNER JOIN authors a ON a.id = ia.author_id
        LEFT JOIN item_reviews ir ON c.item_id = ir.item_id
        LEFT JOIN author_reviews ar ON a.id= ar.author_id
        INNER JOIN item_type it ON c.item_id = it.item_id
        INNER JOIN item_status ist ON c.item_id = ist.item_id
    WHERE a.first_name LIKE '%' + @first_name + '%'
        AND a.last_name LIKE '%' + @last_name + '%'
    ORDER BY c.year_of_publication ASC
END
GO

EXEC search_books_by_author 'Peter', 'harrison';
```

Results Messages

	item_title	year_of_publication	author_id	item_reviews	author_reviews	item_rating	author_rating
1	The Maze Runner	2022-01-01	2	Excellent read!	Excellent writing skills!	4	4
2	The Maze Runner	2022-01-01	2	Excellent read!	Satisfied with the book	4	4
3	The Maze Runner	2022-01-01	2	Satisfied with the story	Excellent writing skills!	4	4
4	The Maze Runner	2022-01-01	2	Satisfied with the story	Satisfied with the book	4	4

Fig 41: Searching for books by author along with the average item rating and author's rating.

```
40
41  ----------------------------ADDED TRIGGERS-----------------------------------
42
43  -----trigger that calculate numbers of book written by an author------
44
45  GO
46  CREATE TRIGGER no_of_books_written
47  ON item_author
48  AFTER INSERT
49  AS
50  BEGIN
51      UPDATE authors
52    SET no_of_books_written = no_of_books_written + 1
53    WHERE id = (SELECT author_id
54      FROM inserted)
55  END
56  GO
57
58
59  -----trigger that calculate publisher numbers of publication------
60
61  GO
62  CREATE TRIGGER update_number_of_publications
63  ON catalog
64  AFTER INSERT
65  AS
66  BEGIN
67      UPDATE publishers
68    SET number_of_publications =  number_of_publications + 1
69    WHERE id = (SELECT publisher_id
70      FROM inserted)
71  END
72  GO
73
74
```

FIG 42: This trigger automatically counts the numbers of book written and published by an author and publishers, please see **fig 32**

	id	member_id	item_id	total_fines	date_taken_out	date_due_back	date_returned	status
1	11	1	1	2000.00	2023-04-10	2023-04-24	*NULL*	Not-Returned
2	12	2	2	280.00	2023-04-12	2023-04-25	*NULL*	Not-Returned
3	13	1	3	280.00	2023-04-15	2023-04-23	*NULL*	Not-Returned
4	14	3	4	650.00	2023-04-17	2023-04-26	*NULL*	Not-Returned
5	15	1	5	450.00	2023-04-20	2023-04-24	*NULL*	Not-Returned
6	16	1	6	650.00	2023-04-22	2023-04-26	*NULL*	Not-Returned
7	17	6	7	800.00	2023-04-25	2023-04-22	*NULL*	Not-Returned
8	18	1	8	450.00	2023-04-27	2023-04-24	*NULL*	Not-Returned
9	19	5	9	950.00	2023-04-20	2023-05-20	2023-04-21	Returned
10	20	4	13	950.00	2023-05-01	2023-05-08	*NULL*	Not-Returned
11	21	1	11	800.00	2023-04-22	2023-04-26	*NULL*	Not-Returned
12	22	1	6	550.00	2023-04-22	2023-04-26	*NULL*	Not-Returned
13	23	1	6	350.00	2023-04-22	2023-04-26	*NULL*	Not-Returned

FIG 43: Updating the **total_fines** column to test payment functionality and total loan histories.

The next step is to make a fine payment for **member_id** of 6 who also have the general system fine details below.

```
2144
2145
2146    select * from member_fine where member_id = 6;
```

Results Messages

	id	member_id	total_fines	amount_paid	outstanding_balance
1	7	6	800.00	0.00	800.00

This table calculates all general loan fines in the system relating to an individual user, the next step is to make a payment of **300 pounds** towards the loan fine payment.

esults Messages

	id	member_id	item_id	total_fines	date_taken_out	date_due_back	date_returned	status
	11	1	1	2000.00	2023-04-10	2023-04-24	*NULL*	Not-Returned
	12	2	2	280.00	2023-04-12	2023-04-25	*NULL*	Not-Returned
	13	1	3	280.00	2023-04-15	2023-04-23	*NULL*	Not-Returned
	14	3	4	650.00	2023-04-17	2023-04-26	*NULL*	Not-Returned
	15	1	5	450.00	2023-04-20	2023-04-24	*NULL*	Not-Returned
	16	1	6	650.00	2023-04-22	2023-04-26	*NULL*	Not-Returned
	17	6	7	500.00	2023-04-25	2023-04-22	2023-04-21	Returned
	18	1	8	450.00	2023-04-27	2023-04-24	*NULL*	Not-Returned
	19	5	9	950.00	2023-04-20	2023-05-20	2023-04-21	Returned

```
2148
2149
2150    EXEC insertFine_payment @amount_paid = 300.00, @member_id = 6, @loan_id = 17, @time = '12:30:00'
2151    select * from member_fine where member_id = 6;
```

Results Messages

	id	member_id	total_fines	amount_paid	outstanding_balance
1	7	6	800.00	300.00	500.00

After making a payment of **300 pounds**, the amount get deducted from the initial **800 pounds**, and there by affecting the **general system calculations for all loans by a single member**, the total_fines, which is the **total amount of fines** histories for member with the id of 6, the amount paid is the **total amount paid** on loans fine and **outstanding balance** is the total amount left to be paid on all loans in the system.

```
-- ------------ Trigger that substract the total fine from loan table and set the status as returned
-----  and inturns update total member's fine table when fine payment is made -------
GO
CREATE TRIGGER update_fine_payment
ON fine_payment
AFTER INSERT
AS
BEGIN
    DECLARE @member_id INT, @loan_id INT, @amount_paid DECIMAL(10,2);
    SELECT @member_id = member_id, @loan_id = loan_id, @amount_paid = amount_paid
    FROM inserted;

    UPDATE loans
  SET total_fines = total_fines - @amount_paid,
      date_returned = GETDATE(),
      status = 'Returned'
  WHERE id = @loan_id;

    UPDATE member_fine
  SET amount_paid = amount_paid + @amount_paid,
      outstanding_balance = total_fines - @amount_paid
  WHERE member_id = @member_id;
END
GO
```

Fig 44: This is the trigger which is responsible for doing the above payment logic calculation after every successful fine payment.

```
----- trigger that add total fines to the loan, every number of days after due when an item is still yet to be returned ----
-- ---- SET @date_due_back = @new_due_date -----
-- --- THIS PREVENT THE TRIGGER FROM RUNNING MULTIPLE TIMES THERE BY GIVING INVALID CALCULATION OF FINE, OR BECOMES AN INVINITE LOOP--

GO
CREATE TRIGGER add_overdue_fee
ON loans
AFTER UPDATE
AS
BEGIN
    DECLARE @member_id INT
    DECLARE @total_fines DECIMAL(10,2)
    DECLARE @outstanding_balance DECIMAL(10,2)
    DECLARE @date_due_back DATE
    DECLARE @status VARCHAR(20)

    SELECT @member_id = member_id, @total_fines = total_fines
    FROM inserted

    IF (DATEDIFF(day, @date_due_back, GETDATE()) > 0 AND @status = 'Not-Returned')
  BEGIN
        DECLARE @new_due_date DATE
        SET @new_due_date = GETDATE()
        SET @total_fines = @total_fines + 10.00
        SET @date_due_back = @new_due_date

        UPDATE member_fine SET total_fines = total_fines + 10.00,
        outstanding_balance = outstanding_balance + 10.00 WHERE member_id = @member_id
    END
END
GO
```

Fig 45: This adds up the overdue fines in the system 24 hours. This would be best placed as a cronjob on the library management system backend server.

ADVICE AN GUIDANCE

Database integrity and concurrency: The ability for multiple users to impact several transactions on the database is called data concurrency.
At the very least, data concurrency allows a lot of users to access it at the same time. The possibility to offer simultaneous operation is a rare feature in databases.

The accuracy, completeness and consistency of the information is a major aspect of data integrity. Data integrity is also about data security when it comes to regulatory compliance.

The type of database data integrity includes physical and logical integrity, physical integrity protects data accuracy. A logical accuracy ensures that the data remains constant and is frequently used in different ways within a relational database.

The use of primary keys as a unique identifier to prevent repetition of data more than it's expected is one of the ways database integrities is established another way is also using constraints in various tables. The need to keep this methodology going forward is advised as a database consultant.

Other steps the library should put in place in maintaining data integrity includes:

- Frequent input validation of input data
- Implementation of access controls
- Adopting security best practices
- Frequent backup of the data

Locked-Based Protocols, 2nd Phase Locking and Timestamping Protocols, Validation Protocols are among the protocols to be adopted as part of the currency control protocol for the library's database.

Database security: The protection of data from accidental, deliberate loss, destruction, or misuse.

As a database consultant, it's advisable for the library to consider the use of database privileges, by granting privileges to developers working or maintaining the database, other ways of maintaining database security include authentication and authorization.

The library should consider creating users and granting them privileges, the library can decide to grant certain privileged to certain developers on the team.

Privileges should be granted on the views, procedures, and functions created in our database to ensure security.

Database backup and Recovery: It is important that your database be backed in case data gets corrupted or lost for any reason. If this backup is used, it may be possible to restore a database as it would have been prior to the failure.

Database backup is to say that a duplicate of database information and data shall be generated and kept on the Backup server only for reasons of safety.

Reasons that could result in the library database failure includes:

- Human error
- Hardware failure
- Catastrophic event

Backups methods that the library could adopt includes:

- Full backup
- Transaction logs
- Differential backup

Database Recovery Methods

There are two methods that are primarily used for database recovery that the library can adopt. These are:

- Log-based recovery
- Shadow paging

CONCLUSION

Following the **SDLC**, I have been able to meet all the requirement and functionality of the library's database. And I have been able to build on the requirements to add more additional functionalities which I believe its going to improve the system.

I was able to fulfil all the client's project requirement by providing an appropriate view, triggers, functions and select statements and documenting all my design process. I was also able to design the database using the 3NF and reducing the possibilities of redundancy and anomalies.

I also provided some advice as a database consultant on database backup and recovery, database security with data integrity and concurrency which the library can adopt to improve the system.

REFRENCES

- Stackify article on SDLC (https://stackify.com/what-is-sdlc/)
- SQL data types(https://www.w3schools.com/mysql/mysql_datatypes.asp)
- Database backup and recovery (https://www.tutorialspoint.com/Database-Backup-and-Recovery)
- Data integrity and concurrency (Advance database coursework module)
- A comprehensive guide to using SQL ENUM and why it's bad (https://www.mysqltutorial.org/mysql-enum/)
- Article on Entity relationship diagram (https://www.lucidchart.com/pages/er-diagrams)

TASK TWO.

INTRODUCTION.

For this task, I was provided with three CSV files, these files are an excerpt from a larger file which is a dataset released every month by the national health service NHS in England.

The contents of this file enlighten me on the prescriptions issued in England specifically in a location called Bolton.

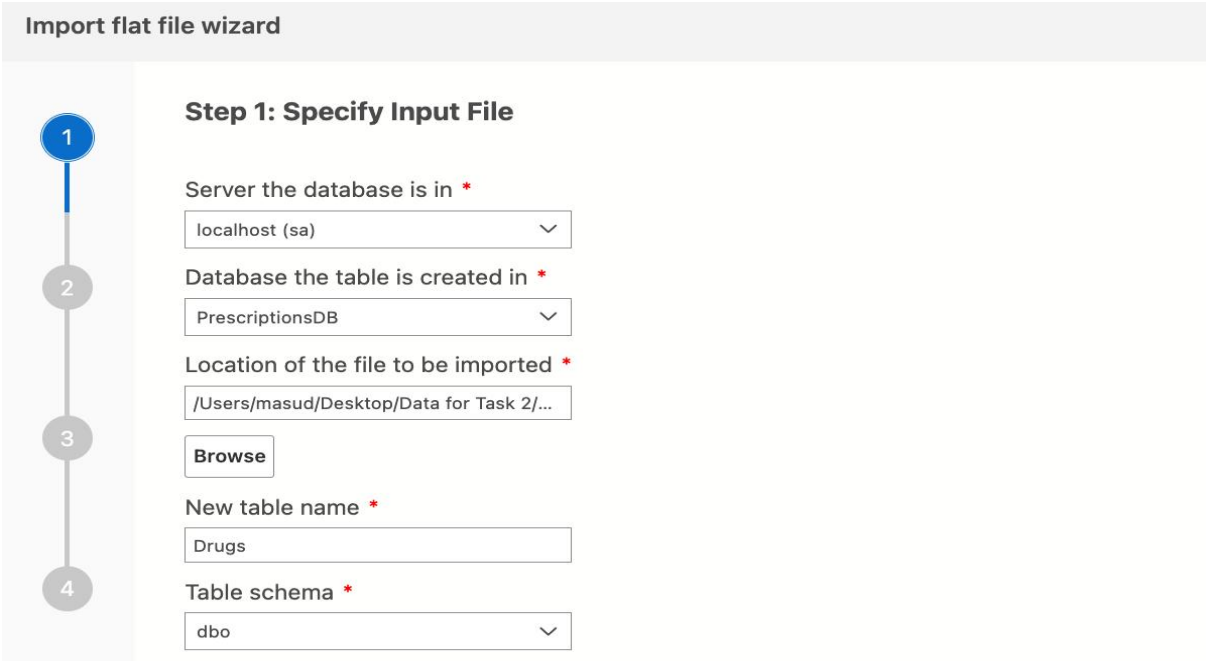
These three files include the **Medical_practice.csv** file, which is a record of names and addresses of medical practices which have prescribed medication within Bolton. The **PRACTICE_CODE** column provides a unique identifier for each practice, which makes it easier to identify it as a primary key column.

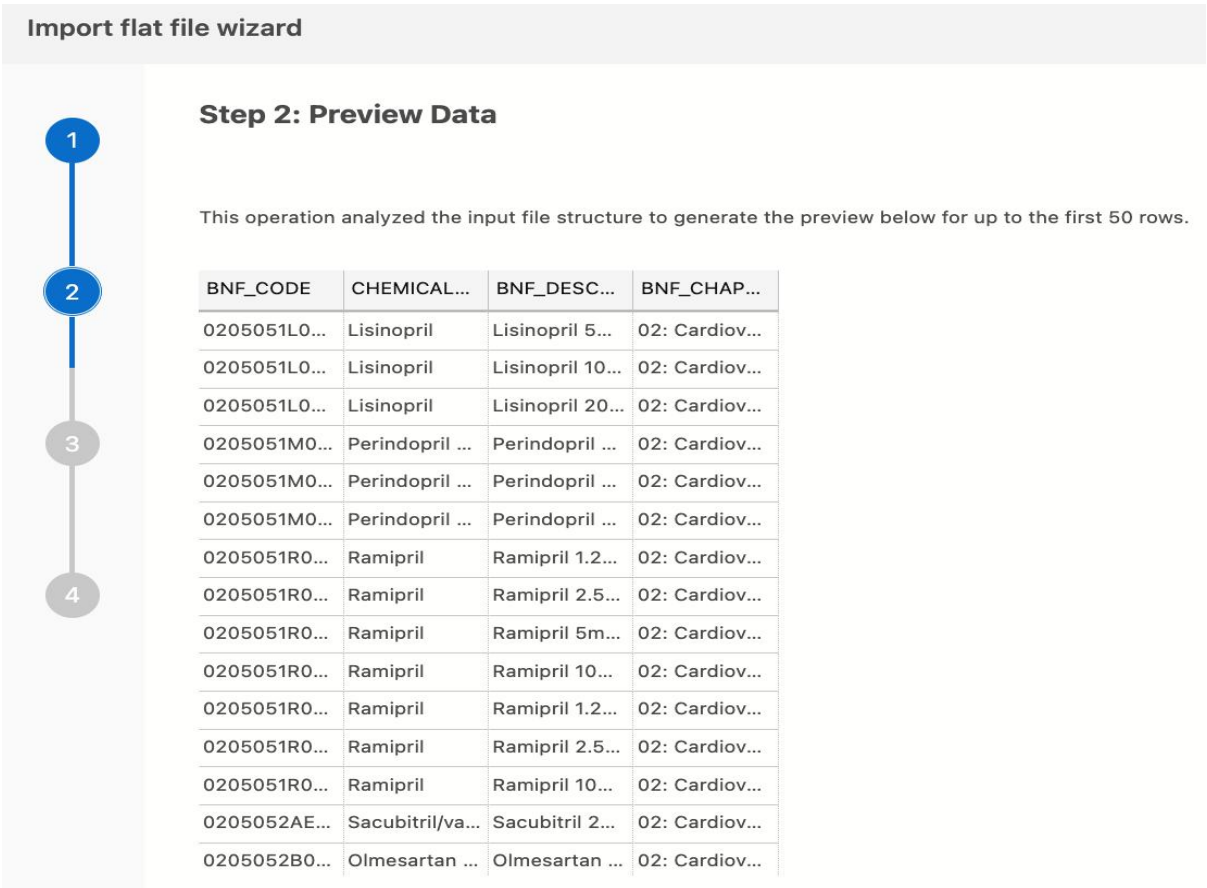
Another file is the **Drug.csv** file which contains details of various drugs that have been prescribed by each practice, including the chemical substance and the product description. **BNF_CHAPTER_PLUS_CODE** is a unique identifier that is a code particular to each drug and can be easily assumed as a primary key based on the table information.

The third file is the **Prescriptions.csv** file which provides information on each prescription. Each row corresponds to a particular prescription. The **PRESCRIPTION_CODE** column provides a unique identifier for each prescription which serves as the primary key.

As a database consultant, I am faced with the challenge of analysing the prescribing data, to under more about the types of medication being prescribed, the organizations doing the prescribing, and the quantities prescribed to fulfil all necessary requirements.

IMPORTING THE CSV FILES AND DATATYPES MODIFICATION.





BNF_CODE	CHEMICAL...	BNF_DESC...	BNF_CHAP...
0205051L0...	Lisinopril	Lisinopril 5...	02: Cardiov...
0205051L0...	Lisinopril	Lisinopril 10...	02: Cardiov...
0205051L0...	Lisinopril	Lisinopril 20...	02: Cardiov...
0205051M0...	Perindopril ...	Perindopril ...	02: Cardiov...
0205051M0...	Perindopril ...	Perindopril ...	02: Cardiov...
0205051M0...	Perindopril ...	Perindopril ...	02: Cardiov...
0205051R0...	Ramipril	Ramipril 1.2...	02: Cardiov...
0205051R0...	Ramipril	Ramipril 2.5...	02: Cardiov...
0205051R0...	Ramipril	Ramipril 5m...	02: Cardiov...
0205051R0...	Ramipril	Ramipril 10...	02: Cardiov...
0205051R0...	Ramipril	Ramipril 1.2...	02: Cardiov...
0205051R0...	Ramipril	Ramipril 2.5...	02: Cardiov...
0205051R0...	Ramipril	Ramipril 10...	02: Cardiov...
0205052AE...	Sacubitril/va...	Sacubitril 2...	02: Cardiov...
0205052B0...	Olmesartan ...	Olmesartan ...	02: Cardiov...

Fig 1: Importing Drugs csv file.

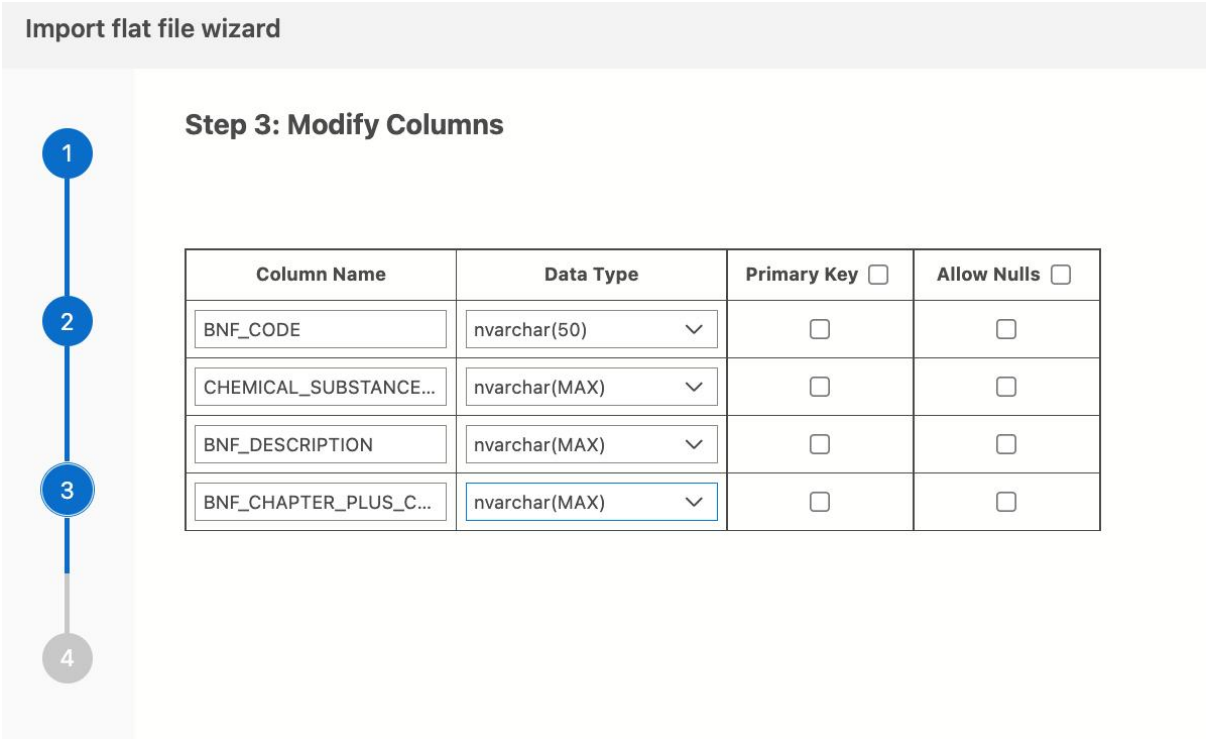


Fig 2: Modifying Drugs csv data types

Fig 1 and 2 shows the importation process for importing drugs csv file, during this process I encountered importation error with data type of column **CHEMICAL_SUBSTANCE_BNF_DESCR**, **BNF_DESCRIPTION, BNF_CHAPTER_PLUS_CODE,** And I was able to solve this data types error by changing the data types from **nvarchar(50)** to **nvarchar(MAX),** This data type is often used for storing large amounts of text data, which the nvarchar(50) data type can't accommodate.

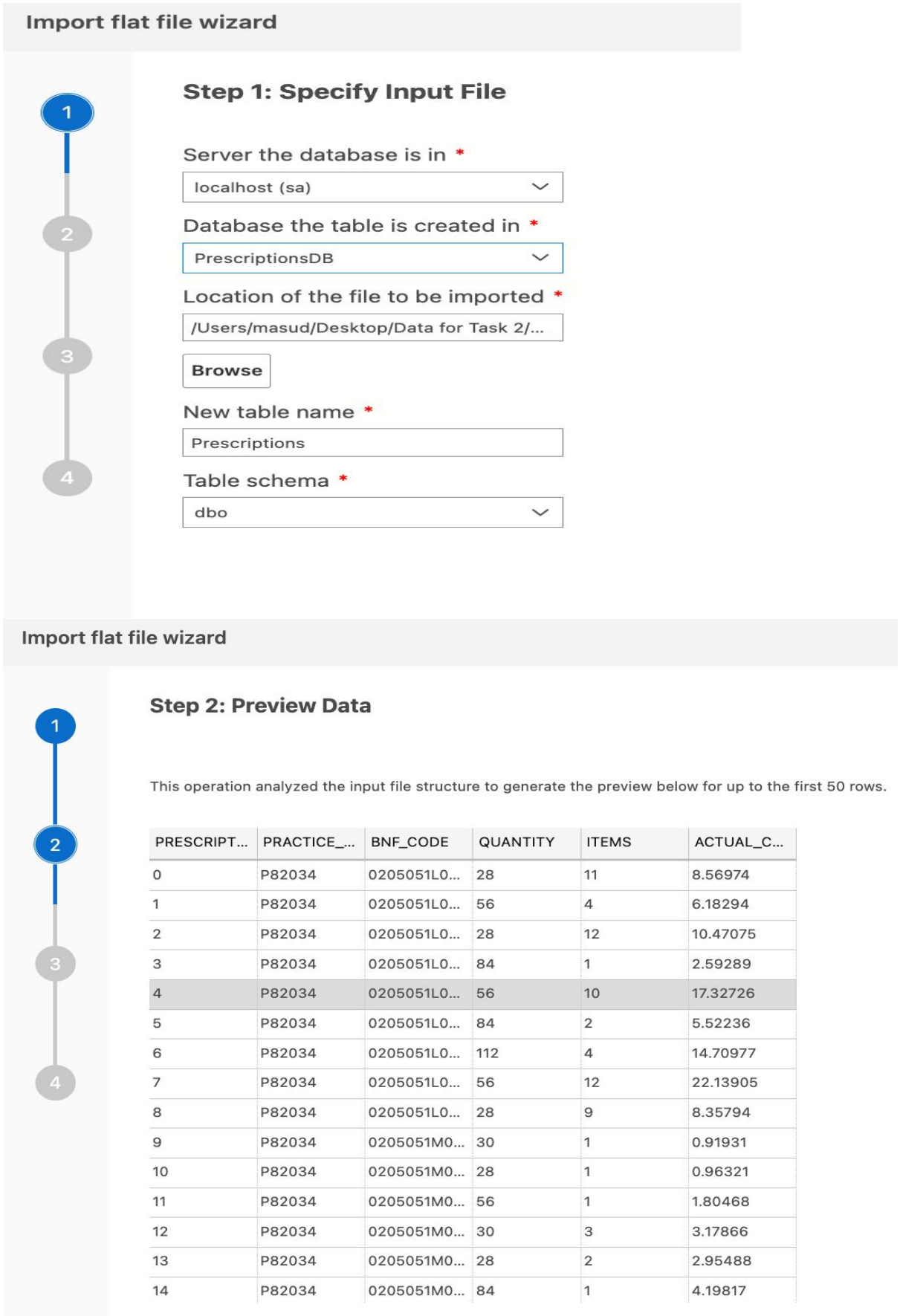


Fig 3: Importing prescriptions csv file

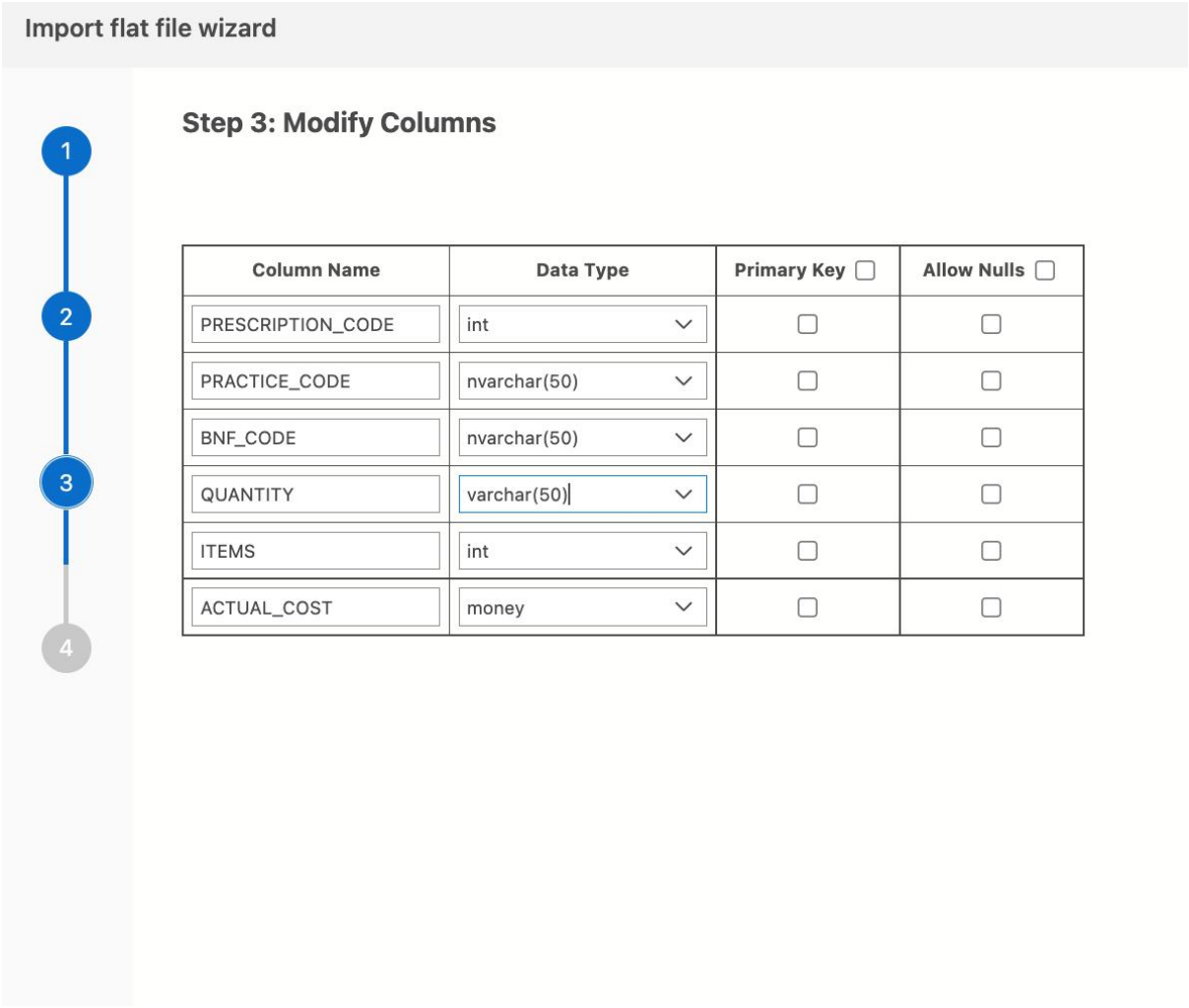


Fig 4: Modifying Prescription file csv data types

Fig 3 and **4** shows the importation process for Prescription file csv. I encountered some data types error which I was able to resolve by changing the database columns into appropriate data types. **PRESCRIPTION_CODE** column was changed from **small-int** to **int**, the **PRACTICE_CODE** and **BNF_CODE** column was also both changed to **nvarchar** data types. The **QUANTITY AND ITEMS** column was both changed to **nvarchar (50)** and **varchar (50)** respectively. And finally, the **ACTUAL_COST** was changed to data type **money**.

ALLOCATING PRIMARY AND FOREIGN KEY VALUES TO TABLE COLUMNS.

KOSEMANI_ABIOLA_ASSESSMENT_2.sql - localh...DB (sa)

Users > masud > Desktop > KOSEMANI_ABIOLA_ASSESSMENT_2.sql

Run | Cancel | Disconnect | Change Connection | PrescriptionsDB | Estimated Plan | Enable Actual Plan | Enable SQLCMD | Export as Notebook

```
1
2   ------------ Create a database with database name PrescriptionsDB ----------
3
4   CREATE DATABASE PrescriptionsDB
5
6
7   ---------- USE the just created PrescriptionsDB ----------
8   USE PrescriptionsDB;
9
10
11  ------------ alter the Prescriptions table and make PRESCRIPTION_CODE column the primary key column ----------
12
13  ALTER TABLE dbo.Prescriptions
14  ADD PRIMARY KEY (PRESCRIPTION_CODE)
15
16
17  ------------- alter the Drugs table and make BNF_CODE column the primary key ---------------
18
19  ALTER TABLE dbo.Drugs
20  ADD PRIMARY KEY (BNF_CODE)
21
22
23  ------------------ alter the Medical_Practice table and make PRACTICE_CODE column the primary key column ------------
24
25  ALTER TABLE dbo.Medical_Practice
26  ADD PRIMARY KEY (PRACTICE_CODE)
27
28
29  ---------------------- alter Prescriptions table and make BNF_CODE the foreign key column which refrences drugs table----------
30
31  ALTER TABLE dbo.Prescriptions
32  ADD CONSTRAINT fk_bnf_code
33  FOREIGN KEY (BNF_CODE) REFERENCES dbo.Drugs (BNF_CODE)
34
35
36  ------------------------ alter Prescriptions table and make PRACTICE_CODE the foreign key column which refrences medical_practice table----------
37
38  ALTER TABLE dbo.Prescriptions
39  ADD CONSTRAINT fk_pratice_code
40  FOREIGN KEY (PRACTICE_CODE) REFERENCES dbo.Medical_Practice(PRACTICE_CODE)
41
42
```

Fig 5: Adding foreign key and primary key constraint.

Fig 5 shows the process of adding the foreign key and primary key as specified in the task requirements, the application of the foreign key and primary key gives us the ability to perform join queries by bringing together tables, after the formed relationship due to keys allocation.

From fig 5: the create database PrescriptionDB create a database with database name PrescriptionDB. The USE PrescriptionDB makes it possible to make use of PrescriptionDB for the SQL queries to follow. This in my case, switch the database environment from masters to PrescriptionDB env.

The **FIRST ALTER STATEMENT** as labelled in **fig 5**, for easy explanation allocate the primary key to **PRESCRIPTION_CODE** column in the **Prescriptions** table by using the **ALTER** statement.

The **SECOND ALTER STATEMENT** as labelled in **fig 5**, allocate the primary key to **BNF_CODE** column in the **Drugs** table by using the **ALTER** statement.

The **THIRD ALTER STATEMENT** as labelled in **fig 5**, allocate the primary key to **PRATICE_CODE** column in the **Medical_Practice** table by using the **ALTER** statement.

The **FOURTH ALTER STATEMENT** as labelled in **fig 5**, allocate the foreign key to column **BNF_CODE** in **Prescriptions** table. The **BNF_CODE** is a primary key column in the **Drug's** table. Hence the foreign key constraint in named **fk_bnf_code.**

The **FIFTH ALTER STATEMENT** as labelled in **fig 5**, allocate the foreign key to column **PRACTICE_CODE** in **Prescriptions** table. The **PRACTICE_CODE** is a primary key column in the **Prescription's** table. Hence the foreign key constraint in named **fk_practice_code.**

Due to this established relationship, I was able to come up with an appropriate Entity relationship diagram (ERD).

ENTITY RELATIONSHIP DIAGRAM

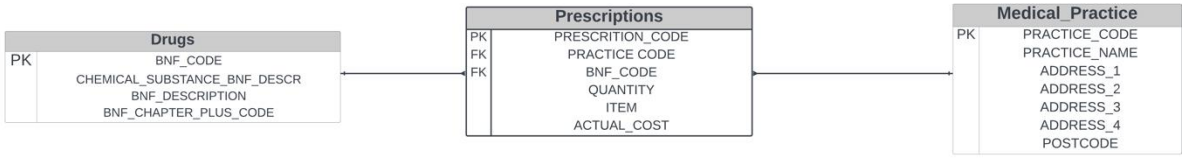


Fig 6: Entity relationship diagram for Prescription database.

Fig 6 shows the relationship between the Drugs table, the prescriptions table and the medical practice table and the shared relationship. The **Drugs** table have a **one-to-many** relationship with the **prescriptions** table, that is a certain drug can be prescribe more than once.

The **medical practice** table on the hand shares a **one-to-many** relationship with the **Prescriptions** table, that is practice can prescribe more than one drug.

The **Drugs** table have the **BNF_CODE** as the primary column and as a foreign key in the **Prescriptions** table with constraint name **fk_bnf_code**. The prescription table have **PRACTICE_CODE** and **BNF_CODE** as the foreign key and the **PRESCRIPTION_CODE** as the primary key.

The **Medical Practice** table have the **PRACTICE CODE** column as the primary key, which is also a foreign key with a **fk_practice_code** constraint name in the Prescriptions table.

T-SQL STATEMENT

QUESTION 2

```
43
44  ---------------------- QUESTION NO 2 ----------------------
45
46  ---------------search the drugs table, where bnf_description column contains capsu or tab characters in them----------------
47
48  select * from dbo.Drugs where bnf_description like '%capsu%' or bnf_description like '%tab%'
```

Results Messages

	BNF_CODE	CHEMICAL_SUBSTANCE_BNF_DESCR	BNF_DESCRIPTION	BNF_CHAPTER_PLUS_CODE
1	0101010I0AAAEAE	Magnesium oxide	Magnesium oxide 100mg tab…	01: Gastro-Intestinal Sys…
2	0101010I0BEAAAC	Magnesium oxide	Oromag 160 capsules	01: Gastro-Intestinal Sys…
3	0101010R0AAAAAA	Simeticone	Simeticone 100mg capsules	01: Gastro-Intestinal Sys…
4	0101010R0AAAHAH	Simeticone	Simeticone 125mg capsules	01: Gastro-Intestinal Sys…
5	0101010R0BHAAAA	Simeticone	WindSetlers 100mg capsules	01: Gastro-Intestinal Sys…
6	010102100BBAQAF	Compound alginates and proprie…	Rennie Peppermint chewabl…	01: Gastro-Intestinal Sys…
7	010102100BBARAF	Compound alginates and proprie…	Rennie Spearmint chewable…	01: Gastro-Intestinal Sys…
8	0101021B0AAAPAP	Alginic acid compound preparat…	Sodium algin 500mg/Potass…	01: Gastro-Intestinal Sys…
9	0101021B0BEAQAP	Alginic acid compound preparat…	Gaviscon Advance Mint che…	01: Gastro-Intestinal Sys…
10	0101021B0BEARA0	Alginic acid compound preparat…	Gaviscon Double Action ch…	01: Gastro-Intestinal Sys…
11	0101021B0BEAUAR	Alginic acid compound preparat…	Gaviscon Peppermint chewa…	01: Gastro-Intestinal Sys…
12	0101021B0BEAWAR	Alginic acid compound preparat…	Gaviscon Strawberry chewa…	01: Gastro-Intestinal Sys…
13	0101021C0AAAFAF	Calcium carbonate	Calcium carbonate 500mg c…	01: Gastro-Intestinal Sys…
14	0101021C0BIABAF	Calcium carbonate	Rennie Orange 500mg chewa…	01: Gastro-Intestinal Sys…
15	0102000A0AAAAAA	Alverine citrate	Alverine 60mg capsules	01: Gastro-Intestinal Sys…
16	0102000A0AAABAB	Alverine citrate	Alverine 120mg capsules	01: Gastro-Intestinal Sys…
17	0102000A0BBAAAA	Alverine citrate	Spasmonal 60mg capsules	01: Gastro-Intestinal Sys…
18	0102000A0BBABAB	Alverine citrate	Spasmonal Forte 120mg cap…	01: Gastro-Intestinal Sys…

Fig 7: This query returns the records which have a matching value using the like operator in **bnf_description** with the word **capsu** or tab.

QUESTION 3.

```
52
53  ---------------------- QUESTION NO 3 ----------------------
54  ---------------------- select prescription_code, and multiply quantity by item then tittle the outcome of the result 'total quantity prescriptions'
55
56  select prescription_code, (floor(quantity) * items) as 'total quantity prescriptions' FROM dbo.Prescriptions
57
58
```

Results Messages

	prescription_code	total quantity prescriptions
1	0	308
2	1	224
3	2	336
4	3	84
5	4	560
6	5	168
7	6	448
8	7	672
9	8	252
10	9	30
11	10	28
12	11	56
13	12	90
14	13	56
15	14	84
16	15	30
17	16	112
18	17	112
19	18	224
20	19	112
21	20	224
22	21	168
23	22	84
24	23	560

Fig 8: This query selects prescription code column and multiply each column quantity by the items as the total quantity prescriptions from the prescriptions table.

QUESTION 4.

```
61  ---------------------------- QUESTION NO 4 --------------------------------
62
63  ------------- select only distinct(different) values form chemical_substance_bnf_descr column in drugs table --------------------
64
65  select distinct(chemical_substance_bnf_descr) from dbo.Drugs
66
67
```

Results Messages

	chemical_substance_bnf_descr
1	Acamprosate calcium
2	Acarbose
3	Aceclofenac
4	Acenocoumarol
5	Acetazolamide
6	Acetic acid
7	Acetylcysteine
8	Aciclovir
9	Acipimox
10	Aclidinium bromide
11	Aclidinium bromide/formoterol
12	Acrivastine
13	Adapalene
14	Adapalene and benzoyl peroxide
15	Adhesive Discs/Rings/Pads/Plas…
16	Adhesive Dressing Remover Ster…
17	Adhesive Removers (Sprays/Liqu…
18	Adrenaline
19	Alclometasone dipropionate
20	Alendronic acid
21	Alfacalcidol
22	Alfuzosin hydrochloride
23	Alginic acid compound preparat…
24	Alimemazine tartrate
25	Aliskiren
26	Alkyl sulphate

Fig 9: This query selects distinct different and unique value result from the drugs table.

QUESTION 5

```
69
70  ------------------- QUESTION NO 5 -------------
71
72  -----------------calculates the minimum, maximum, and average actual cost for each distinct value of bnf_chapter_plus_code --
73  ----- in the drugs table, and also counts the total number of prescriptions for each bnf_chapter_plus_code value ---
74
75  select
76      max(p.actual_cost) as 'minimum cost',
77      avg(p.actual_cost) as 'average cost',
78      min(p.actual_cost) as 'maxiximum cost',
79      d.bnf_chapter_plus_code,
80      count(*) as 'total number of prescriptions'
81  from
82      dbo.drugs AS d
83      join dbo.prescriptions  AS p on d.bnf_code = p.bnf_code
84  group by
85  d.bnf_chapter_plus_code
86
87
88
```

Results Messages

	minimum cost	average cost	maxiximum cost	bnf_chapter_plus_code	total number of prescriptions
1	434.7997	21.4403	1.34	11: Eye	2676
2	1402.4766	22.2507	0.4864	13: Skin	5692
3	2552.9234	38.3528	0.2057	21: Appliances	5856
4	2599.1877	40.2214	0.2181	20: Dressings	1014
5	1193.3481	32.2498	0.4331	19: Other Drugs and Prepa…	338
6	11094.7521	35.9956	0.1311	02: Cardiovascular System	19186
7	1262.2079	21.2474	0.1966	05: Infections	4657
8	882.5271	82.2616	1.5894	23: Stoma Appliances	1697
9	2197.1551	54.9382	0.29	08: Malignant Disease and…	754
10	538.8857	42.8654	2.8703	22: Incontinence Applianc…	535
11	2777.6906	35.1252	0.1405	01: Gastro-Intestinal Sys…	8777
12	1476.6376	16.2459	0.2713	10: Musculoskeletal and J…	3634

Fig 10: This query calculates the **maximum, average and minimum** cost using the sum operator and also returns values of **bnf_chapter_plus_code** column in the drugs table.

QUESTION 6

```
88
89  ---------------- QUESTION NO 6 -----------------
90  ---- this query returns the most expensive prescription prescribed(max(p.actual_cost) as 'maximum cost') by each practice name ----
91  ---- sorted in descending order('maximum cost' desc) by prescription cost('maximum cost') ----------
92  --- this also returns the rows where the most expensive prescriprions is more than 4000(having max(p.actual_cost) > 4000) -------
93
94  select
95      d.bnf_description,
96      max(p.actual_cost) as 'maximum cost',
97      m.practice_name
98  from
99      dbo.medical_practice m
100     join dbo.prescriptions p on m.practice_code = p.practice_code
101     join dbo.drugs d on p.bnf_code = d.bnf_code
102 group by
103  d.bnf_description,
104  m.practice_name
105 having
106    max(p.actual_cost) > 4000
107 order by
108    'maximum cost' desc
109
```

Results Messages

	bnf_description	maximum cost	practice_name
1	Adjuvanted quadrivalent f…	21570.9208	UNSWORTH GROUP PRACTICE
2	Influenza vaccine (surfac…	11651.9342	UNSWORTH GROUP PRACTICE
3	Apixaban 5mg tablets	11094.7521	UNSWORTH GROUP PRACTICE
4	Adjuvanted quadrivalent f…	11031.5885	BROMLEY MEADOWS SURGERY
5	Supemtek Quadrivalent vac…	8659.596	KILDONAN HOUSE
6	Apixaban 5mg tablets	7313.581	KEARSLEY MEDICAL CENTRE
7	Apixaban 2.5mg tablets	6430.9405	UNSWORTH GROUP PRACTICE
8	Apixaban 5mg tablets	6069.7747	BOLTON COMMUNITY PRACTICE
9	Flucelvax Tetra vacc inj …	5641.1537	BROMLEY MEADOWS SURGERY
10	Apixaban 5mg tablets	5223.9864	KILDONAN HOUSE
11	Apixaban 5mg tablets	4626.9594	HARWOOD MEDICAL CENTRE
12	Apixaban 5mg tablets	4377.6525	MANDALAY MEDICAL CENTRE
13	Elemental 028 Extra powde…	4250.4384	CROMPTON VIEW SURGERY
14	Trelegy Ellipta 92microg/…	4161.8103	UNSWORTH GROUP PRACTICE
15	Fluad Tetra vaccine inj 0…	4031.9573	DALEFIELD SURGERY

Fig 11: This query returns the most expensive prescriptions prescribed by each practice which is sorted in descending order by the prescription cost and return most expensive rows which is greater than 4000.

QUESTION 7A

```
114
115   --------------------QUESTION NO 7A----------------------
116   --Find the drugs that were prescribed by more than one medical practice
117
118   SELECT Drugs.BNF_DESCRIPTION
119   FROM Drugs
120   JOIN Prescriptions ON Drugs.BNF_CODE = Prescriptions.BNF_CODE
121   GROUP BY Drugs.BNF_DESCRIPTION
122   HAVING COUNT(DISTINCT Prescriptions.PRACTICE_CODE) > 1
123
```

Results Messages

	BNF_DESCRIPTION
1	4SURE hypodermic insulin …
2	A2A Spacer with small/med…
3	AaCarb Carbomer 0.2% eye …
4	AacuLose Hypromellose 0.3…
5	AacuLose Hypromellose 0.5…
6	AaproMel 0.3% eye drops
7	AaproMel 0.5% eye drops
8	AaqEye Carmellose 0.5% ey…
9	Abasaglar 100units/ml sol…
10	Abasaglar KwikPen 100unit…
11	Abidec Multivitamin drops
12	Abilify 10mg tablets
13	Able Spacer
14	Able Spacer with mask sma…
15	AbsorbaGel sachets
16	Acamprosate 333mg gastro-…
17	Acarbose 100mg tablets
18	Acarbose 50mg tablets
19	Accrete D3 One a Day 1000…
20	Accrete D3 tablets
21	Acenocoumarol 1mg tablets
22	Acetazolamide 250mg modif…
23	Acetazolamide 250mg table…
24	Acetic acid 2% ear spray
25	Acetylcysteine 5% eye dro…
26	Aciclovir 200mg dispersib…
27	Aciclovir 200mg tablets

Fig 10: This query returns BNF_DESCRIPTION column form drugs prescribed by more than one medical practice.

QUESTION 7B

```
123
124
125  ------------------QUESTION NO 7B-----------------
126  ---returns the sum of total cost of each prescription prescribe by kildonan house----
127
128  SELECT d.BNF_DESCRIPTION, SUM(p.ACTUAL_COST) AS 'sum total cost of prescriptions'
129  FROM Prescriptions AS p
130  JOIN Medical_Practice  AS mp ON p.PRACTICE_CODE = mp.PRACTICE_CODE
131  JOIN Drugs AS d ON p.BNF_CODE = d.BNF_CODE
132  WHERE mp.PRACTICE_NAME = 'KILDONAN HOUSE'
133  GROUP BY d.BNF_DESCRIPTION;
134
```

Results Messages

	BNF_DESCRIPTION	sum total cost of prescriptions
1	4SURE hypodermic insulin …	3.7055
2	4SURE hypodermic insulin …	3.7055
3	AacuLose Hypromellose 0.3…	4.7336
4	AacuLose Hypromellose 0.5…	0.938
5	Abasaglar KwikPen 100unit…	291.1444
6	Abilify 10mg tablets	89.8059
7	Able Spacer	8.2338
8	Able Spacer with mask sma…	6.7067
9	Acamprosate 333mg gastro-…	183.2178
10	Acarbose 50mg tablets	27.3773

Fig 11: This query returns the sum of total cost of each prescription, where practice name is **KILDONAN HOUSE**, grouped by drug **BNF_DESCRIPTION** column.

QUESTION 7C

```
137
138  ----------QUESTION NO 7C-----------------------
139  --- this returns practice name with total numbers of prescriptions greater than 500 with the sum of total numbers of items ---
140  --- and average prescriptions cost orderby total numbers of items, the average cost, the number of prescriptions in desc order---
141  SELECT
142      Medical_Practice.PRACTICE_NAME,
143      SUM(Prescriptions.ITEMS) AS 'total numbers of items',
144      AVG(Prescriptions.ACTUAL_COST) AS 'the average cost',
145      COUNT(*) AS 'the number of prescriptions'
146  FROM Prescriptions
147  JOIN Medical_Practice ON Prescriptions.PRACTICE_CODE = Medical_Practice.PRACTICE_CODE
148  GROUP BY Medical_Practice.PRACTICE_NAME
149  HAVING COUNT(*) > 500
150  ORDER BY 'total numbers of items', 'the average cost', 'the number of prescriptions' DESC;
151
152
```

Results Messages

	PRACTICE_NAME	total numbers of items	the average cost	the number of prescriptions
1	3D MEDICAL CENTRE	2415	20.2257	860
2	CHARLOTTE STREET SURGERY	2555	32.4355	880
3	BRADFORD STREET SURGERY	3164	24.6666	1195
4	BARDOC GP OOH	3239	19.7205	721
5	DR M DAKSHINA-MURTHI	3452	23.1765	1213
6	BEEHIVE SURGERY	4629	33.8173	1043
7	GREAT LEVER ONE	4774	25.7716	1353
8	DEANE CLINIC 1	5059	22.7894	1198
9	EDGWORTH MEDICAL CENTRE	5181	30.0431	1726
10	ORIENT HOUSE MEDICAL CENT…	5573	28.3464	1581

Fig 12: This query returns practice name with their total numbers of prescriptions greater than 500 with the sum of total numbers of items and average prescriptions cost order-by total numbers of items, the average cost and the number of prescriptions in descending order

QUESTION 7D.

```
154
155   -------------------------QUESTION NO 7D-------------------------
156   --select practice code, and count the total number of prescriptions from prescriptions table where quantity is greater than 500 ---
157   ---- and the practice have prescribe more than once, group by practice code ---
158
159   SELECT PRACTICE_CODE, COUNT(*) AS 'Total number of prescriptions'
160   FROM Prescriptions
161   WHERE FLOOR(QUANTITY) > 500
162   GROUP BY PRACTICE_CODE
163   HAVING COUNT(*) > 1
164   ORDER BY 'Total number of prescriptions' DESC;
165
```

Results Messages

	PRACTICE_CODE	Total number of prescriptions
1	P82015	205
2	Y03079	147
3	P82008	133
4	P82006	131
5	P82004	125
6	P82007	115
7	P82003	110
8	P82001	110
9	P82031	95
10	P82010	95
11	P82009	94
12	P82016	88
13	P82002	85
14	P82021	84
15	P82023	78
16	P82005	76
17	P82609	74
18	P82643	72

Fig 13: This query counts the total numbers of practice code in the prescriptions table where the quantity is greater than 500, group by practice code, with practice which must have made more than one prescription ordered by total number of prescriptions.

QUESTION 7E

```
173
174    select count(*) as 'total number of practice centers in address_3', address_3
175    from dbo.medical_practice
176    group by address_3;
177
```

Results Messages

	total number of practice centers in address_3	address_3
1	1	*NULL*
2	45	BOLTON
3	2	BRIGHTMET, BOLTON
4	2	BROMLEY CROSS
5	1	CAPTAIN LEES ROAD
6	1	CENTRE FOR HEALTH
7	1	FARNWORTH
8	1	HALLIWELL
9	1	HEATON, BOLTON
10	1	HORWICH
11	1	LITTLE LEVER
12	1	NAVIGATION PARK
13	1	WATERS MEETING ROAD
14	1	WESTHOUGHTON

```
177
178    select count(*) as 'total number of practice centers in address_4', address_4
179    from dbo.medical_practice
180    group by address_4;
181
```

Results Messages

	total number of practice centers in address_4	address_4
1	19	*NULL*
2	1	ASHBURNER STREET,BOLTON
3	9	BOLTON
4	1	GREATER MANCHESTER
5	28	LANCASHIRE
6	1	WATERS MEETING RD, BOLTON
7	1	WESTHOUGHTON,BOLTON

```
167
168    --------------------------QUESTION NO 7E--------------------------------
169
170    select count(*) as 'total number of practice centers in address_2', address_2
171    from dbo.medical_practice
172    group by address_2;
173
```

Results Messages

	total number of practice centers in address_2	address_2
1	8	*NULL*
2	1	119 CHURCH STREET
3	1	160 ST.HELENS ROAD
4	1	2 LUCY STREET
5	1	2–4 MOOR LANE
6	1	374–376 ST HELENS ROAD
7	1	469 CHORLEY OLD ROAD
8	2	501 CROMPTON WAY
9	1	933 BLACKBURN RD,SHARPLES

Fig 14: This shows the total number of practice centres in each address.

CONCLUSION

I have been able to satisfy the requirement of this task by writing an SQL statements that analyses all three provided CSV files.

I was able to make use of various methods to analyse the data, which includes the use of nested queries including the use of exists or in, the use of joins with system functions, use of group by, having, and order by clauses.

I was also able to apply the foreign key and primary keys to the right column in various tables by establishing the right relationship among the tables. This has also in generating an entity relationship diagram which was also one of the task requirements.