

LIBRARY DATABASE DESIGN

INTRODUCTION

I am to design a database for a library. The database is required to store information of the members of the library, items in the library, loan history and overdue fines repayment related to the library.

Having gone through the client's requirements specification carefully, I will be using the Top_down approach for designing a database. This process involves identifying entities, their attributes and relationships between entities. I will first identify the different entities; I will achieve this by identifying the nouns and noun phrases in the requirement specification excluding those nouns that describe the qualities of other objects. I will then establish relationships between the entities and normalize the database to third normal form (3NF). I will build an entity relationship diagram that will help me visualize my database system, I will then proceed to creating a database, creating tables, database objects related to the library and backup the database for security purpose.

Assumptions Made While Designing the Database

- In the Members Information table, I added other attributes namely: Gender, Salt Member_JoinDate, Member_EndDate.
- For a member to be inactive or seize being a member of the library, I assume that the member_enddate column in the membersinfo table should not be null
- In the Loan history table, I added a derived attribute which is Days_Overdue, the days overdue will be computed by finding the difference between the current date and the Due_date.
- In the Library Catalogue, I decomposed the ItemType attribute into a new table to reduce redundancy, this is shown on the ER diagram.
- For this Database design, I assume that members have only one address.
- I assume that the Itemtypes in the LibraryCatalogue are single copies, no Itemtype with more than one copy.
- For this design, I assume that the Author and Year Published columns in the Library Catalogue table represent all the data types.

PART 1

1.1 Normalization

The entities identified in the requirement specification are:

- a) Members Information
- b) Library Catalogue (Items table)
- c) Library Loans
- d) Overdue Fines and Repayment
- e) Former Members

On identifying the different entities, the next step I took was to identify and associate attributes with appropriate entities, I did this by identifying nouns and noun phrases which have characteristics that relate to the entities.

Attributes for Members Information identified in requirement specification:

Full name, address, date of birth, username, password, email address, telephone number.

Attributes for Library Catalogue identified in requirement specification:

Item title, item type, author, year of publication, date item was added, item status, date item identified as lost/removed, ISBN.

Attributes for Library loans identified in requirement specification:

Member, item, date item was taken out, date item is due back, date item was returned.

Attributes for overdue fines and repayment identified in requirement specification:

Repayment Id, date/time of repayment, amount paid, repayment method (cash or card).

After associating the attributes to the appropriate entities, I proceeded to normalize the database to a Third normal form (3NF). The core objective of normalization is to transform structure from lower normal forms to higher normal forms such that there is minimal data

redundancy in the system and of course to curb Data manipulation anomalies (DML) such as update, insert and delete.

1.1.1 Unnormalized Form UNF

The Members Information table is unnormalized because it contains Address attribute, which is a composite attribute, I decomposed the address attribute into a table with different columns. This is shown in the ER Diagram.

1.1.2 First Normal Form 1NF

After decomposing the address attribute and creating a new table for Address, all the entities were in the first normal form because they satisfy its conditions which states that:

A relation R is said to be in first normal form (1NF) if and only if:

- All the attributes are atomic, meaning there are no composite attributes.
- Attributes are not multi_valued.

1.1.3 Second Normal Form 2NF

For the entities to be in second normal form, the attributes should be fully functionally dependent on the primary key and should be in the First normal form. I introduced a primary key (PK) attribute for all the entities.

This is shown on the Entity Relationship Diagram.

1.1.4 Third Normal Form 3NF

All the entities are in the third normal form because they satisfy its conditions which states that:

A relation R is said to be in third normal form (3NF) if and only if:

- R is in second normal form (2NF)
- No non-prime attribute A in R is transitively dependent on the primary key through another non-prime attribute.
- A non-key attribute may not be functionally dependent on another non-key attribute.

1.2 Entity Relationship Diagram (ERD)

I created an Entity-Relationship diagram (ERD) to help me easily visualize my database system. The ER diagram shows all the Entities with their attributes and how they relate to one another. I am using the **Crow's Foot Notation style**.

1.3 Relationship Between Entities

- i. Members information and Addresses – A member has only one address, this is a one-to-one relationship as shown in the ER diagram.
- ii. Members Information and Loan history – A member can exist but have no loan, a member can also have many loans. This is a one to zero or one to many relationships.
- iii. Members Information and OverdueFines_Repayment – A member can exist but make no repayment, a member can also make many repayments. This is a one to zero or one to many relationships.
- iv. Library catalogue and Loan History – An Item in a library can be borrowed once at a time, this is one to one relationship.
- v. Loan History and OverdueFines_Repayment – A loan can be Repaid once or several times; this is one to one or a one-to-many relationships.
- vi. Library Catalogue and ItemTypes – An Itemtype can have one item or multiple items in the Library Catalogue, this type of relationship is a one to one or one to many.

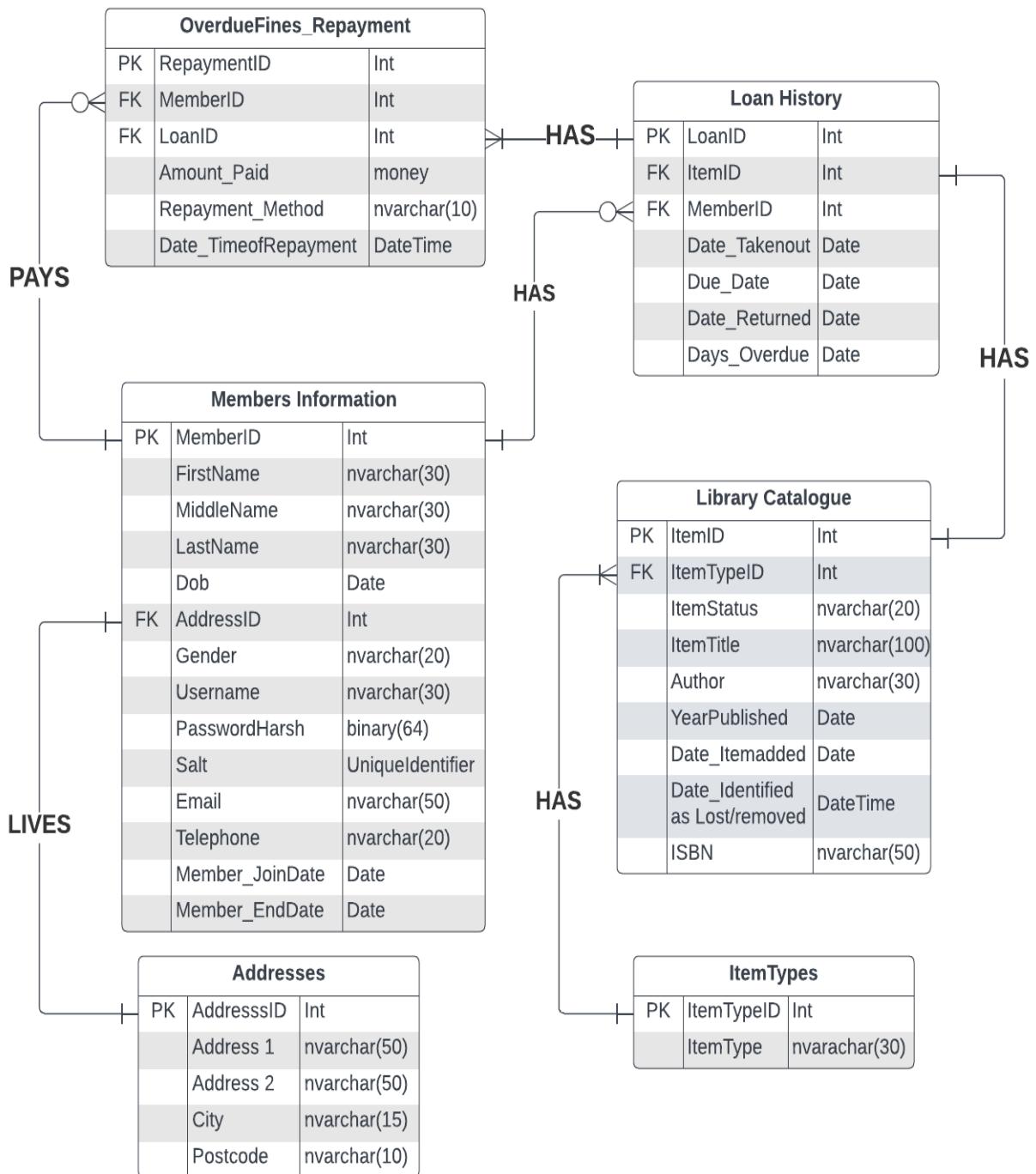
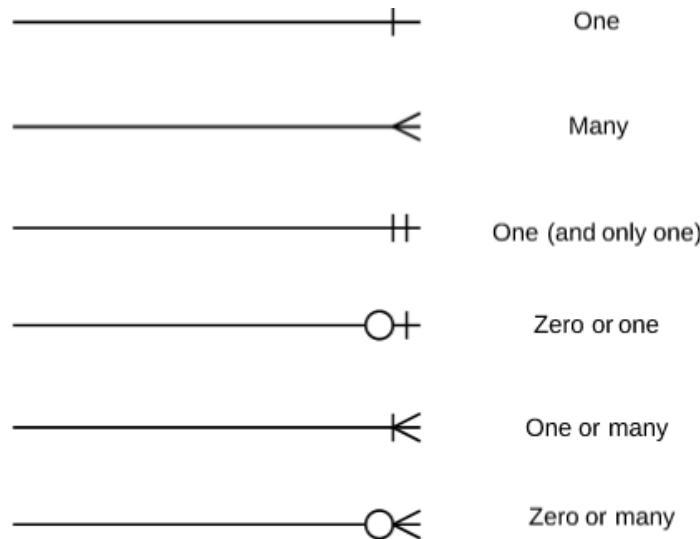


Fig 1.1 Entity Relationship Diagram (Crow's Foot Notation style)

In the ER diagram above, I have clearly highlighted columns with Primary keys (PK) and foreign keys (FK).

Crow's Foot Cardinality

Cardinality defines the maximum number of relationship instances in which an entity can participate.



The table below contains all the columns and their datatypes and reason for choosing a datatype.

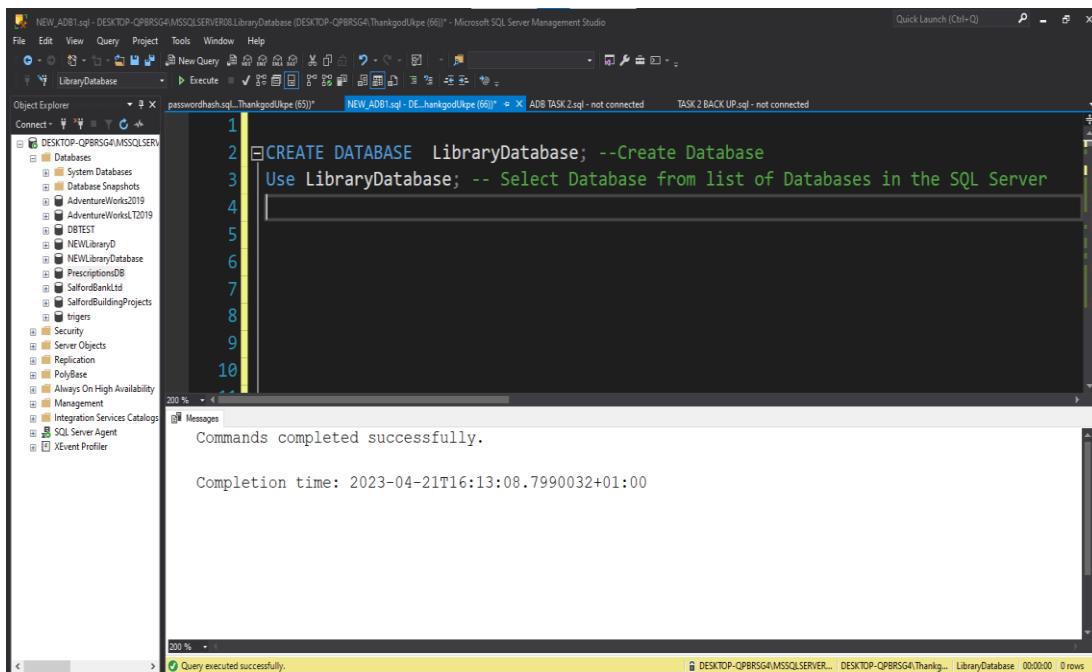
COLUMN NAMES	DATATYPE	REASON
MemberId, AddressId LoanId, ItemId, RepaymentId, ItemtypeId	Int	I chose the integer datatype for all the primary key and foreign key columns. These keys must be unique for each record in a table and the integer datatype provides this uniqueness because they can be incremented automatically by the database system.

Firstname, Middlename, Lastname Gender, Username, Email Telephone, Address1, Address2, City, Postcode, ItemType, Repayment_method, ItemStatus, Itemtitle, Author, ISBN	nvarchar	I chose the nvarchar datatype for these columns because of its flexibility, it can store both Unicode and non-Unicode character data, meaning that it can store characters such as symbols, special characters, and characters from different languages. The nvarchar datatype can store up to 4,000 characters which is sufficient for storing name, address and Itemtitle information.
Member_JoinDate, Member_Enddate, Dob, Date_Takenout, Due_Date, Date_Takenout, Days_Overdue	Date	I used the date datatype for these columns because of the date format to avoid inconsistencies while entering values for these columns.
Date_Identified as Lost/removed	DateTime	I used the datetime datatype because it suits the use of this field which is returning information about both date and time.
PasswordHarsh	binary	This datatype is used to store sensitive data such as passwords as it provides a high level of security. The

		binary datatype is used in hashing algorithm.
Salt	Uniqueidentifier	I used the uniqueidentifier datatype because it generates globally unique values and makes it difficult to guess.
Amount_Paid	Money	This datatype is used to store monetary values, it suits the use case of this field.

1. 4 Creating a Database

I created a Database with the name LibraryDatabase, I then called the USE function to select the created Database from a list of all the Databases in the SQL Server. This is shown in Fig below.



The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer on the left, under the 'DESKTOP-QPBRSG4\MYSQLSERV' node, the 'Databases' folder is expanded, showing various system databases like 'master', 'model', 'msdb', and 'tempdb', along with user databases such as 'AdventureworksLT2019', 'AdventureworksLT2019', 'OBTEST', 'NEWLibrary0', 'NEWLibraryDatabase', 'PrescriptionsDB', 'SaferBankLtd', 'SafeerBuildingProjects', and 'triggers'. In the center pane, there are two tabs: 'passwordhash.sql - ThankgodUkpe (65)' and 'NEW AD81.sql - DE...ThankgodUkpe (66)'. The 'NEW AD81.sql' tab contains the following SQL code:

```

1 CREATE DATABASE LibraryDatabase; --Create Database
2 Use LibraryDatabase; -- Select Database from list of Databases in the SQL Server
3
4
5
6
7
8
9
10

```

Below the code, the status bar indicates 'Commands completed successfully.' and 'Completion time: 2023-04-21T16:13:08.7990032+01:00'. The bottom status bar also shows 'Query executed successfully.', 'DESKTOP-QPBRSG4\MYSQLSERV', 'DESKTOP-QPBRSG4\ThankgodUkpe', 'LibraryDatabase', '00:00:00', and '0 rows'.

1. 5 Creating Tables

- I. I first created the Address table because it has no inferential constraint in it.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows a database named 'DESKTOP-QPBRSG4\MSQLSERV'. The main pane displays the following SQL code:

```
10
11
12
13 -- Creating the Addresses Table
14 CREATE TABLE Addresses(
15     AddressID int IDENTITY(10,1) NOT NULL PRIMARY KEY,
16     Address1 nvarchar(50) NOT NULL,
17     Address2 nvarchar(50) NULL,
18     City nvarchar(50) NULL,
19     Postcode nvarchar(10) NOT NULL);
20
21
22
```

The 'Messages' pane at the bottom shows the output:

```
Commands completed successfully.

Completion time: 2023-04-21T16:19:58.2068215+01:00
```

A status bar at the bottom right indicates 'Query executed successfully.'

- II. I then created the MembersInfo table.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows a database named 'DESKTOP-QPBRSG4\MSQLSERV'. The main pane displays the following SQL code:

```
12
13 -- Creating the Members Information Table
14 CREATE TABLE MembersInfo(
15     MemberId int IDENTITY(1,1) NOT NULL PRIMARY KEY,
16     FirstName nvarchar(30) NOT NULL,
17     MiddleName nvarchar(30) NULL,
18     LastName nvarchar(30) NOT NULL,
19     DateOfBirth Date NOT NULL,
20     Gender nvarchar(20) NOT NULL ,
21     AddressID Int NOT NULL FOREIGN KEY (AddressID)
22     REFERENCES Addresses(AddressID),
23     Username nvarchar(30) UNIQUE NOT NULL,
24     PasswordHash binary(64) NOT NULL,
25     Salt UNIQUEIDENTIFIER,
26     MemberEmail nvarchar(100) NULL, CHECK(MemberEmail Like '%_@%._%'),
27     TelephoneNo nvarchar(20) NULL,
28     Member_JoinDate Date NOT NULL,
29     Member_EndDate Date NULL);
30
31
```

The 'Messages' pane at the bottom shows the output:

```
Commands completed successfully.

Completion time: 2023-04-28T09:30:27.1386136+01:00
```

A status bar at the bottom right indicates 'Query executed successfully.'

III. I created the ItemTypes table.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists several databases, including 'DESKTOP-QPBRSG4\MSSQLSERVER00\LibraryDatabase' and 'DESKTOP-QPBRSG4\ThanggodUkpe (66)'. The 'Query' tab in the center contains the following SQL code:

```
31
32
33
34
35    --Creating the ItemTypes table
36 CREATE TABLE ItemTypes(
37     ItemTypeID int IDENTITY(1,1) NOT NULL PRIMARY KEY,
38     Itemtype nvarchar(20) NOT NULL);
39
40
41
```

The 'Messages' pane at the bottom displays the output:

```
Commands completed successfully.

Completion time: 2023-04-21T17:04:35.8944333+01:00
```

A status bar at the bottom right indicates 'Query executed successfully.'

IV. I created the LibraryCatalogue table

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists several databases, including 'DESKTOP-QPBRSG4\MSSQLSERVER00\LibraryDatabase' and 'DESKTOP-QPBRSG4\ThanggodUkpe (66)'. The 'Query' tab in the center contains the following SQL code:

```
36
37
38
39    --Creating the LibraryCatalogue table
40 CREATE TABLE LibraryCatalogue(
41     ItemId int IDENTITY(100,1) NOT NULL PRIMARY KEY,
42     ItemStatus nvarchar(30) NOT NULL CHECK(ItemStatus IN ( 'Available', 'On loan', 'Overdue', 'Lost_Or_Removed' )),
43     ItemTitle nvarchar(50) NOT NULL,
44     ItemTypeID int NOT NULL FOREIGN KEY (ItemTypeId) ,
45     REFERENCES ItemTypes(ItemTypeId) ,
46     Author nvarchar(30) NULL,
47     YearPublished date NOT NULL,
48     Date_Itemadded date NOT NULL,
49     Date_Identified_asLost_removed date NULL,
50     ISBN nvarchar(30) NULL);
51
52
53
```

The 'Messages' pane at the bottom displays the output:

```
Commands completed successfully.

Completion time: 2023-04-21T17:18:18.3844974+01:00
```

A status bar at the bottom right indicates 'Query executed successfully.'

V. I created the LibraryLoan table

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, a database named 'LibraryDatabase' is selected. In the center pane, a query window displays the following SQL code:

```
--Creating the Libraryloan table
CREATE TABLE LibraryLoan(
    LoanId int IDENTITY(200,1) NOT NULL PRIMARY KEY,
    ItemId int NOT NULL FOREIGN KEY (ItemId)
        REFERENCES LibraryCatalogue(ItemId),
    MemberId int NOT NULL FOREIGN KEY (MemberId)
        REFERENCES MembersInfo(MemberId),
    Date_TakenOut Date NOT NULL,
    DueDate Date NULL,
    DateReturned Date NULL,
    Days_Overdue Int NOT NULL DEFAULT 0);
```

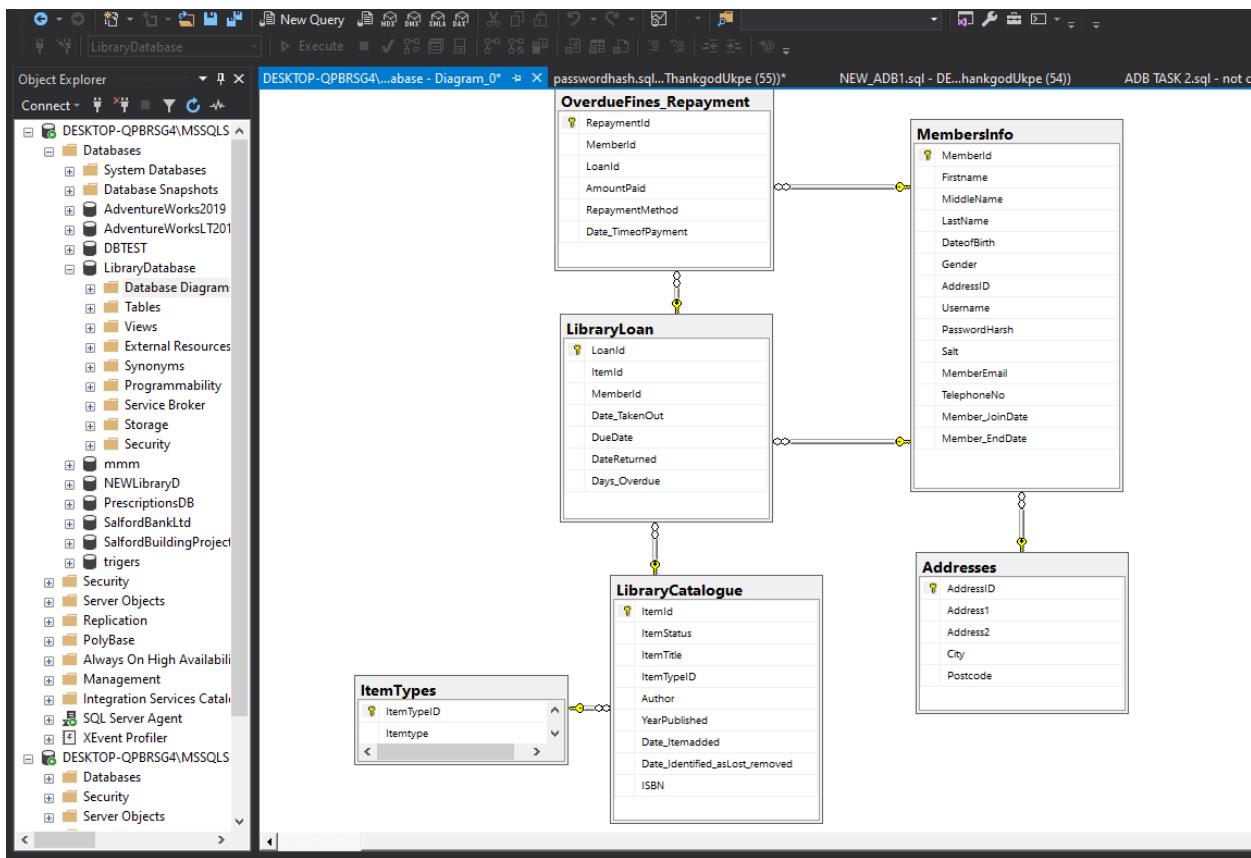
The status bar at the bottom indicates 'Query executed successfully.'

VI. I created the OverduFines_Repayment table

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, a database named 'LibraryDatabase' is selected. In the center pane, a query window displays the following SQL code:

```
CREATE TABLE OverdueFines_Repayment(
    RepaymentId int IDENTITY(500,1) NOT NULL PRIMARY KEY,
    MemberId int NOT NULL FOREIGN KEY (MemberId)
        REFERENCES MembersInfo(MemberId),
    LoanId int NOT NULL FOREIGN KEY (LoanId),
    AmountPaid money NOT NULL,
    RepaymentMethod nvarchar(20) NOT NULL CHECK(RepaymentMethod IN ( 'Cash', 'Card')),
    Date_TimeofPayment DateTime NOT NULL DEFAULT GETDATE());
```

The status bar at the bottom indicates 'Query executed successfully.'



Database Diagram

PART 2

The library requires stored procedures or user defined functions to carry out certain operations.

(a) I created a stored procedure that searches the LibraryCatalogue for matching character string by Title. The code snippet in Fig 2a creates a stored procedure Search_ItemTitle and adds it to the LibraryDatabase, it takes an input parameter @Title and the datatype of the parameter. The select statement selects all the columns in the Librarycatalogue, I used the Where clause together with the Like and wildcard operators to search for the specified procedure parameter in the

Itemtitle column. I used the order by clause to sort the result in by the year published. The exec keyword executes the stored procedure and returns results.

```

82
83 --QUESTION 2(A)
84 --A STORED PROCEDURE THAT SEARCHES THE LIBRARYCATALOGUE FOR MATCHING CHARACTER STRING BY TITLE
85
86 CREATE PROCEDURE Search_ItemTitle
87 @Title NVARCHAR(100)
88 AS
89 BEGIN
90     SELECT *
91     FROM LibraryCatalogue
92     WHERE ItemTitle LIKE '%' + @Title + '%'
93     ORDER BY yearpublished DESC;
94 END
95
96 EXEC Search_ItemTitle 'LORD ';

```

Messages

Commands completed successfully.

Completion time: 2023-04-21T18:26:21.5489307+01:00

Query executed successfully.

Fig 2a

(b) I created a stored procedure items_duedateless_5, this procedure returns a full list of items currently on loan which have a Duedate of less than five days from the current date (system date as at time of execution). The query in the body of the procedure joins two tables Librarycatalogue and Libraryloan based on their common column ItemId. The select statement selects itemtitle column from the Librarycatalogue table, ItemId and Duedate columns from the LibraryLoan table. The where clause is used to specify different conditions, first condition is the Datereturned column should be Null, second condition uses the Datediff function to calculate the difference in days between the Getdate function and the Duedate which must not be less than 0, third condition also uses the Datediff function to calculate the number of days between the Getdate function and Duesdate which must be less than 5. The Getdate function returns the system date as at the time

of execution. The Exec keyword is used to execute the stored procedure and return results, this is shown in Fig 2b.

```

--QUESTION 2(B)
-- A STORED PROCEDURE THAT RETURNS A FULL LIST OF ALL ITEMS CURRENTLY ON LOAN WHICH HAVE
-- WHICH HAVE A DUE DATE OF LESS THAN FIVE DAYS FROM THE CURRENT
-- DATE
CREATE PROCEDURE items_duedateless_5
AS(
SELECT LibraryCatalogue.ItemTitle, LibraryLoan.ItemId, DueDate
FROM LibraryLoan INNER JOIN LibraryCatalogue ON LibraryLoan.ItemId = LibraryCatalogue.ItemId
WHERE DateReturned IS NULL
AND DATEDIFF(dd, GETDATE(), DueDate) < 0
AND DATEDIFF(dd, GETDATE(), DueDate) < 5;
)
Exec items_duedateless_5

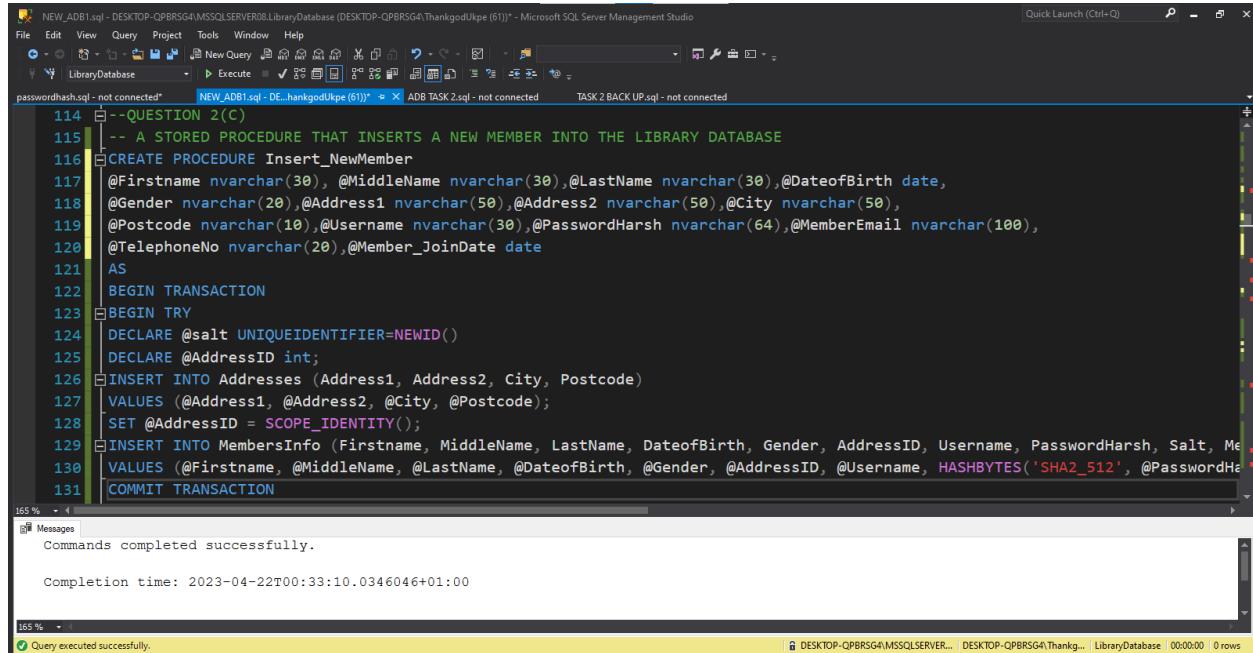
```

Commands completed successfully.
Completion time: 2023-04-21T18:26:21.5489307+01:00

Fig 2b

(c) I created a stored procedure that will insert a new member into the MembersInfo table along with the address information in the Addresses table. The procedure takes in all the attributes of both tables as input except their primary keys because the primary keys increment automatically. To ensure that the stored procedure is a transaction with ACID properties, I used a Try and Catch block to handle any errors that may occur during execution. I declared UniqueIdentifier for the salt column using the NEWID() function which hashes the password. The procedure first inserts data into the Address table before inserting into the Membersinfo table because it has a foreign key referencing the Address table. The Scope_Identity() function was used to retrieve the identity value that was assigned to the newly inserted row in the Address table, this value was assigned to the AddressID

variable and used to insert corresponding row in the MembersInfo table. The Exec keyword is used to execute the procedure, this is shown Fig 2c and Fig 2c contd.



```

114 --QUESTION 2(C)
115 -- A STORED PROCEDURE THAT INSERTS A NEW MEMBER INTO THE LIBRARY DATABASE
116 CREATE PROCEDURE Insert_NewMember
117     @Firstname nvarchar(30), @MiddleName nvarchar(30),@LastName nvarchar(30),@DateofBirth date,
118     @Gender nvarchar(20),@Address1 nvarchar(50),@Address2 nvarchar(50),@City nvarchar(50),
119     @Postcode nvarchar(10),@Username nvarchar(30),@PasswordHarsh nvarchar(64),@MemberEmail nvarchar(100),
120     @TelephoneNo nvarchar(20),@Member_JoinDate date
121 AS
122 BEGIN TRANSACTION
123 BEGIN TRY
124     DECLARE @salt UNIQUEIDENTIFIER=NEWID()
125     DECLARE @AddressID int;
126     INSERT INTO Addresses (Address1, Address2, City, Postcode)
127     VALUES (@Address1, @Address2, @City, @Postcode);
128     SET @AddressID = SCOPE_IDENTITY();
129     INSERT INTO MembersInfo (Firstname, MiddleName, LastName, DateofBirth, Gender, AddressID, Username, PasswordHarsh, Salt, MemberEmail, TelephoneNo, Member_JoinDate)
130     VALUES (@Firstname, @MiddleName, @LastName, @DateofBirth, @Gender, @AddressID, @Username, HASHBYTES('SHA2_512', @PasswordHarsh), @salt, @MemberEmail, @TelephoneNo, @Member_JoinDate)
131     COMMIT TRANSACTION

```

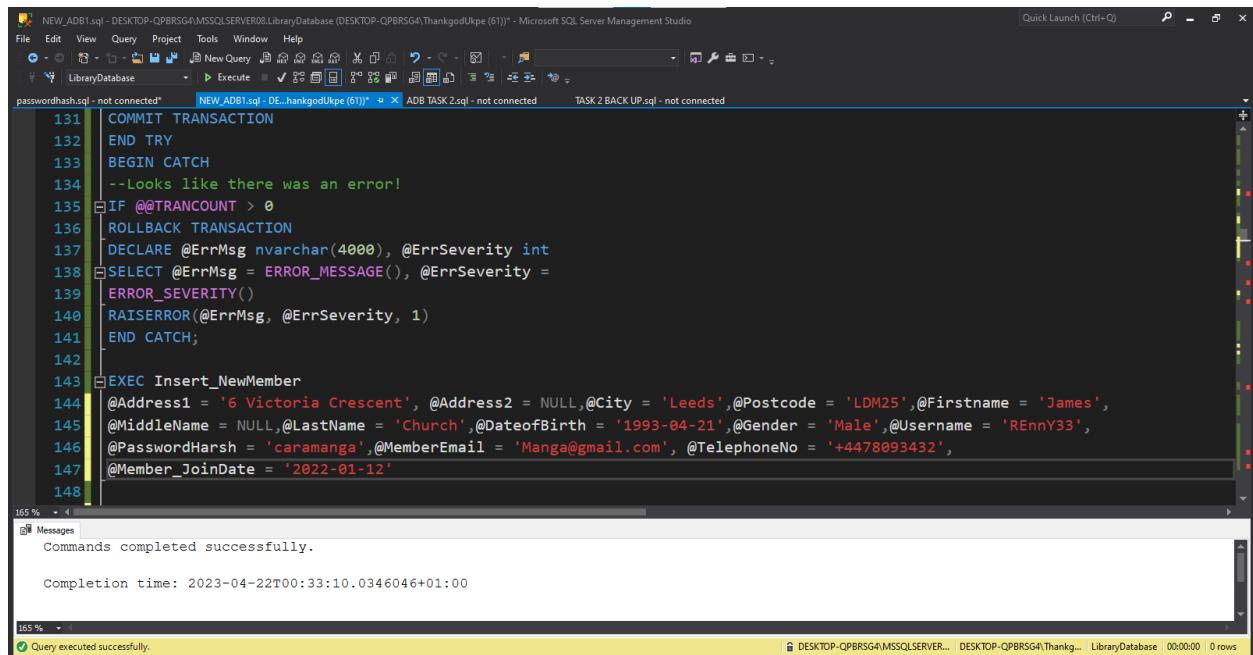
Messages

Commands completed successfully.

Completion time: 2023-04-22T00:33:10.0346046+01:00

Query executed successfully.

Fig 2c



```

131     COMMIT TRANSACTION
132 END TRY
133 BEGIN CATCH
134     --Looks like there was an error!
135     IF @@TRANCOUNT > 0
136         ROLLBACK TRANSACTION
137     DECLARE @ErrMsg nvarchar(4000), @ErrSeverity int
138     SELECT @ErrMsg = ERROR_MESSAGE(), @ErrSeverity =
139     ERROR_SEVERITY()
140     RAISERROR(@ErrMsg, @ErrSeverity, 1)
141     END CATCH;
142
143 EXEC Insert_NewMember
144     @Address1 = '6 Victoria Crescent', @Address2 = NULL,@City = 'Leeds',@Postcode = 'LDM25',@Firstname = 'James',
145     @MiddleName = NULL,@LastName = 'Church',@DateofBirth = '1993-04-21',@Gender = 'Male',@Username = 'REnnY33',
146     @PasswordHarsh = 'caramanga',@MemberEmail = 'Manga@gmail.com', @TelephoneNo = '+4478093432',
147     @Member_JoinDate = '2022-01-12'
148

```

Messages

Commands completed successfully.

Completion time: 2023-04-22T00:33:10.0346046+01:00

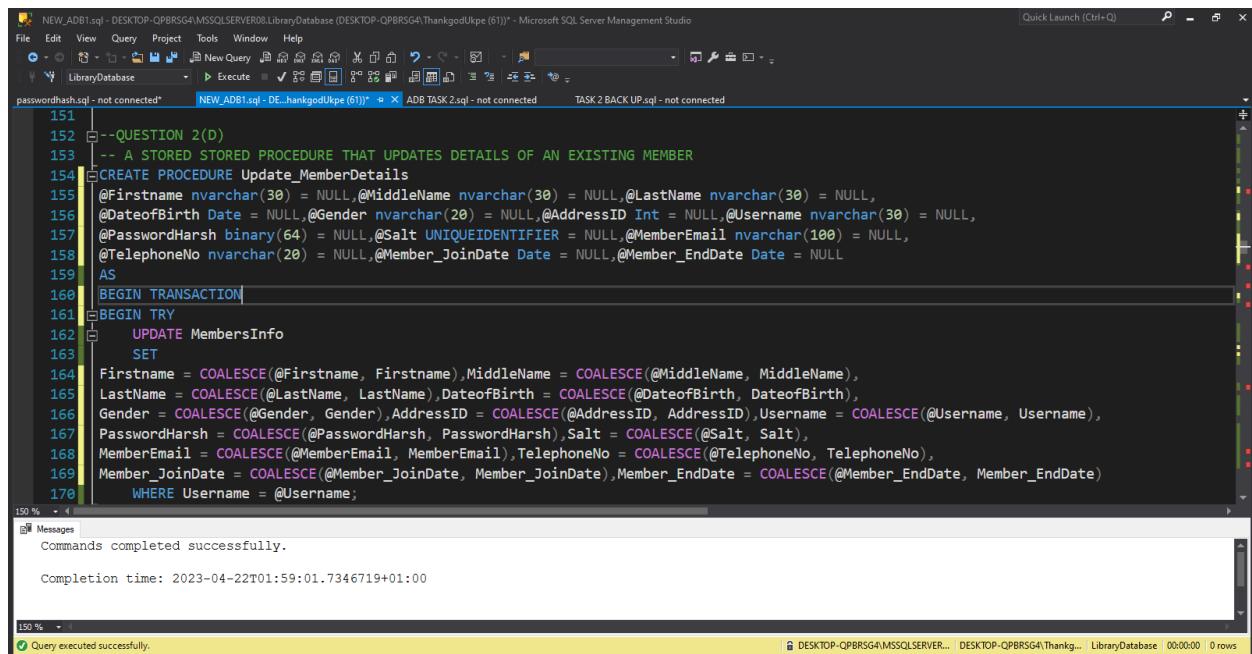
Query executed successfully.

Fig 2c contd

(d) I created a stored procedure to update details of a member in the MembersInfo table. The procedure takes in all the attributes in the MembersInfo table as input parameters. The stored procedure makes use of Try and Catch block which helps to ensure that the procedure can handle any errors that may occur during execution.

The Coalesce function is an SQL advanced function that is used to return the first non-null value in a list of expressions. In this procedure, the Coalesce function is used to allow for optional parameters to be passed into the stored procedure.

The Where clause specifies that the update should only be applied to the member record with specified Username. The Exec keyword executes the stored procedure, this is shown Fig 2d and Fig 2d contd.



The screenshot shows the Microsoft SQL Server Management Studio interface. A new query window is open with the following T-SQL code:

```
151
152 --QUESTION 2(D)
153 -- A STORED PROCEDURE THAT UPDATES DETAILS OF AN EXISTING MEMBER
154 CREATE PROCEDURE Update_MemberDetails
155 @Firstname nvarchar(30) = NULL, @MiddleName nvarchar(30) = NULL, @LastName nvarchar(30) = NULL,
156 @DateOfBirth Date = NULL, @Gender nvarchar(20) = NULL, @AddressID Int = NULL, @Username nvarchar(30) = NULL,
157 @PasswordHash binary(64) = NULL, @Salt UNIQUEIDENTIFIER = NULL, @MemberEmail nvarchar(100) = NULL,
158 @TelephoneNo nvarchar(20) = NULL, @Member_JoinDate Date = NULL, @Member_EndDate Date = NULL
159 AS
160 BEGIN TRANSACTION
161 BEGIN TRY
162     UPDATE MembersInfo
163     SET
164         Firstname = COALESCE(@Firstname, Firstname), MiddleName = COALESCE(@MiddleName, MiddleName),
165         LastName = COALESCE(@LastName, LastName), DateOfBirth = COALESCE(@DateOfBirth, DateOfBirth),
166         Gender = COALESCE(@Gender, Gender), AddressID = COALESCE(@AddressID, AddressID), Username = COALESCE(@Username, Username),
167         PasswordHash = COALESCE(@PasswordHash, PasswordHash), Salt = COALESCE(@Salt, Salt),
168         MemberEmail = COALESCE(@MemberEmail, MemberEmail), TelephoneNo = COALESCE(@TelephoneNo, TelephoneNo),
169         Member_JoinDate = COALESCE(@Member_JoinDate, Member_JoinDate), Member_EndDate = COALESCE(@Member_EndDate, Member_EndDate)
170     WHERE Username = @Username;
```

The code is highlighted in green and blue, indicating syntax. The status bar at the bottom shows "Query executed successfully." and the completion time: 2023-04-22T01:59:01.7346719+01:00.

Fig 2d

```

169 | Member_JoinDate = COALESCE(@Member_JoinDate, Member_JoinDate), Member_EndDate = COALESCE(@Member_EndDate, Member_EndDate)
170 | WHERE Username = @Username;
171 | COMMIT TRANSACTION
172 | END TRY
173 | BEGIN CATCH
174 | --Looks like there was an error!
175 | IF @@TRANCOUNT > 0
176 | ROLLBACK TRANSACTION
177 | DECLARE @ErrMsg nvarchar(4000), @ErrSeverity int
178 | SELECT @ErrMsg = ERROR_MESSAGE(), @ErrSeverity =
179 | ERROR_SEVERITY()
180 | RAISERROR(@ErrMsg, @ErrSeverity, 1)
181 | END CATCH;
182 |
183 | EXEC Update_MemberDetails @Username = 'sjdoe', @LastName = 'MFIOh'
184 | SELECT * FROM MembersInfo

```

Messages
Commands completed successfully.

Completion time: 2023-04-22T01:59:01.7346719+01:00

Query executed successfully.

Fig 2d contd

PART 3

I created a view with the name View_LoanHistory, the view contains information about all previous and current loans, and includes details of the item borrowed, borrowed date, DueDate and associated fines if any for each loan. The select statement selects the LoanId, ItemId, ItemTitle, MemberId, Date_TakenOut, DueDate, DateReturned, Days_Overdue and concatenates the FirstName, Middlename, Lastname as Fullname.

The view computes the Overdue_Fine using a case statement, the Datediff function calculates the difference in days between the due date and Getdate() function which is the current date during execution, if the result is greater than 0, it is multiplied by 10. Otherwise, the Overdue fine is set to 0.

The View joins three tables: LibraryCatalogue, LibraryLoan, MembersInfo, the LibraryCatalogue table is joined with the Libraryloan based on their common column ItemID and the Libraryloan table is joined with the MembersInfo table based on their common column MemberID. A select statement is used to display the view after it's created. This is shown in Fig 3.

The screenshot shows the Microsoft SQL Server Management Studio interface. A query window is open with the following SQL code:

```

189 -- QUESTION 3
190 -- A VIEW ON LIBRARY LOAN SHOWING ALL PREVIOUS AND CURRENT LOANS, AND INCLUDING DETAILS
191 -- OF THE ITEM BORROWED, BORROWED DATE, DUE DATE AND ANY ASSOCIATED FINES FOR EACH LOAN
192 CREATE VIEW View_LoanHistory
193 AS
194 SELECT
195     LoanId, LibraryCatalogue.ItemId, LibraryCatalogue.ItemTitle, MembersInfo.MemberId,
196     MembersInfo.FirstName + ' ' + ISNULL(MembersInfo.MiddleName, '') + ' ' + MembersInfo.LastName AS FullName,
197     Date_TakenOut, DueDate, DateReturned, Days_Overdue,
198     CASE
199         WHEN DATEDIFF(dd, DueDate, GETDATE()) > 0 THEN (DATEDIFF(dd, DueDate, GETDATE()) * 10)
200         WHEN DATEDIFF(dd, DueDate, GETDATE()) < 0 THEN 0
201         ELSE 0
202     END AS Overdue_Fine
203 FROM
204     LibraryLoan
205     INNER JOIN LibraryCatalogue ON LibraryLoan.ItemId = LibraryCatalogue.ItemId
206     INNER JOIN MembersInfo ON LibraryLoan.MemberId = MembersInfo.MemberId;
207
208 SELECT * FROM View_LoanHistory
209

```

The results pane shows a table structure with columns: LoanId, ItemId, ItemTitle, MemberId, FullName, DueDate, DateReturned, Days_Overdue, and Overdue_Fine. The status bar at the bottom indicates "Query executed successfully."

Fig 3

PART 4

I created an after-Update trigger on the Libraryloan table with the name update_ItemStatus, this trigger will update the Itemstatus to in LibraryCatalogue table to available when an Item is returned. The trigger uses the If update statement to check if there is an update on the Datereturned column, the trigger will fire automatically once there is an update on the Datereturned column in the Libraryloan table. The trigger joins the Librarycatalogue table with the Libraryloan table on the ItemId column and where clause is used to filter records that have non-Null Datereturned value. This is shown in Fig 4.

The screenshot shows the Microsoft SQL Server Management Studio interface. In the center pane, there is a code editor window displaying the following SQL script:

```
211
212 -- QUESTION 4
213 -- A TRIGGER THAT UPDATES THE ITEM STATUS IN THE LIBRACATALOGUE TO AVAILABLE WHEN AND ITEM IS RETURNED
214 CREATE TRIGGER update_ItemStatus ON LibraryLOAN
215 AFTER UPDATE
216 AS
217 BEGIN
218 IF UPDATE(DateReturned)
219 BEGIN
220 Update LibraryCatalogue
221 SET ItemStatus = 'Available'
222 FROM LibraryCatalogue Lc
223 Join LibraryLoan Ll ON Lc.ItemId = Ll.ItemId
224 Where Ll.ItemId In (Select ItemId from inserted)
225 and Ll.DateReturned IS NOT NULL
226 END
227 END
```

Below the code editor, the 'Messages' pane shows the output:

```
Commands completed successfully.
```

At the bottom of the screen, the status bar indicates:

```
Completion time: 2023-04-22T03:46:02.8617317+01:00
Query executed successfully.
```

Fig 4

PART 5

I created an Sql function with the name total_loans, it takes an input parameter @date_out with a date datatype, this function returns and integer value that specifies the total number of loans taken out on specific date. The aggregate function count is used to count the total number of records that match the specified date in the Libraryloan table. Where clause is used to filter the Date_takentout column for records that match the input parameter @date_out. After creating the function, a select statement is used to call the function, this is shown in Fig 5.

```

238
239 --Question 5
240 -- A FUNCTION WHICH ALLOWS THE LIBRARY TO IDENTIFY THE TOTAL NUMBER OF LOANS MADE ON A SPECIFIC DATE
241 CREATE FUNCTION total_loans (@date_out AS DATE)
242 RETURNS INT
243 AS
244 BEGIN
245     RETURN (
246         SELECT COUNT(*)
247         FROM LibraryLoan
248         WHERE Date_TakenOut = @date_out);
249 END
250
251 SELECT dbo.total_loans ('2022-12-03') AS Total_loans
252
253
254

```

Messages

Commands completed successfully.

Completion time: 2023-04-22T04:44:48.0970655+01:00

Query executed successfully.

Fig 5

PART 6

6.1 Inserting Records into Tables

```

255 --QUESTION 6
256 --INSERTING RECORDS INTO ADDRESS TABLE
257 INSERT INTO Addresses (Address1, Address2, City, Postcode)
258 VALUES
259     ('12 High Street', 'Flat 1', 'London', 'E1 7AB'),
260     ('25 Oxford Road', 'Apartment 2', 'Manchester', 'M1 5AN'),
261     ('6 Main Street', NULL, 'Birmingham', 'B3 3HJ'),
262     ('42 Park Lane', 'Apartment 7', 'Bristol', 'BS1 5JH'),
263     ('8 Station Road', NULL, 'Leeds', 'LS1 4DY'),
264     ('17 Queen Street', NULL, 'Glasgow', 'G1 3ED'),
265     ('14 Market Square', 'Apartment 11', 'Belfast', 'BT1 2FF'),
266     ('2 The Grove', NULL, 'Cardiff', 'CF10 3BA'),
267     ('33 Highfield Avenue', NULL, 'Edinburgh', 'EH16 5PJ'),
268     ('1 Victoria Road', NULL, 'Sheffield', 'S2 2SS'),
269     ('18 King Street', 'Flat 12', 'Liverpool', 'L1 8HT'),
270     ('10 Bridge Street', 'Flat 4', 'Newcastle upon Tyne', 'NE1 8AD'),

```

Messages

(15 rows affected)

Completion time: 2023-04-22T05:35:24.6405186+01:00

Query executed successfully.

6.1.1 Fig 6.1.1 Inserting records into Address table

The screenshot shows the Microsoft SQL Server Management Studio interface. A query window is open with the following SQL code:

```
275 --INSERTING VALUES INTO MEMBERSINFO TABLE
276 DECLARE @Salt UNIQUEIDENTIFIER = NEWID()
277 INSERT INTO MembersInfo (Firstname, MiddleName, LastName, DateOfBirth, Gender, AddressID, Username, Password)
278 VALUES
279 ('John', 'William', 'Smith', '1985-03-15', 'Male', 10, 'jwsmith', HASHBYTES('SHA2_512', 'ytfw0r&d123' + CAST(@S
280 ('Sarah', 'Jane', 'Doe', '1990-07-12', 'Female', 11, 'sjdoe', HASHBYTES('SHA2_512', 'Sarah#d223' + CAST(@Salt AS
281 ('Adam', 'Joseph', 'Brown', '1995-11-23', 'Female', 12, 'ajbrown', HASHBYTES('SHA2_512', 'browny#Van' + CAST(@S
282 ('Emily', NULL, 'Davis', '1988-04-30', 'Female', 13, 'edavis', HASHBYTES('SHA2_512', 'Err@davis' + CAST(@Salt AS
283 ('Robert', 'Michael', 'Johnson', '1979-12-08', 'Male', 14, 'Mjohnson', HASHBYTES('SHA2_512', 'Jooord12#3' + CAST(@S
284 ('Laura', 'Grace', 'Lee', '1983-06-10', 'Female', 15, 'lglee', HASHBYTES('SHA2_512', 'Word&d@245' + CAST(@Salt AS
285 ('Daniel', NULL, 'Wilson', '1992-01-17', 'Male', 16, 'dpwilson', HASHBYTES('SHA2_512', 'bOnDJames@123' + CAST(@S
286 ('Megan', 'Elizabeth', 'Taylor', '1997-08-27', 'Female', 17, 'Metaylor', HASHBYTES('SHA2_512', 'metaYorr#d123'
287 ('William', 'Henry', 'Clark', '1980-05-20', 'Male', 18, 'whclark', HASHBYTES('SHA2_512', 'ytfclarkFF#1' + CAST(@S
288 ('Amanda', 'Rose', 'Baker', '1989-02-05', 'Female', 19, 'arobaker', HASHBYTES('SHA2_512', 'JBaker$543' + CAST(@S
289 ('Steven', NULL, 'Anderson', '1993-09-14', 'Male', 20, 'standerson', HASHBYTES('SHA2_512', 'Stander@son21' + CAST(@S
290 ('Rachel', 'Nicole', 'Evans', '1998-06-18', 'Female', 21, 'rnevans', HASHBYTES('SHA2_512', 'Rnevan#66tt' + CAST(@S
```

The execution completed successfully, inserting 15 rows. The completion time was 2023-04-22T05:45:21.8169799+01:00.

Fig 6.1.2 Inserting Records into MembersInfo table

The screenshot shows the Microsoft SQL Server Management Studio interface. A query window is open with the following SQL code:

```
295
296
297
298
299
300
301
302 --INSERTING RECORDS INTO THE ITEMTYPES TABLE
303 INSERT INTO ItemTypes (ItemType)
304 VALUES ('Books'), ('Journals'), ('DVDs'), ('Other Media');
```

The execution completed successfully, inserting 4 rows. The completion time was 2023-04-22T05:53:49.3376025+01:00.

Fig 6.1.3 Inserting Records into ItemTypes table

The screenshot shows the Microsoft SQL Server Management Studio interface. A query window is open with the following SQL code:

```

313 | --INSERTING RECORDS INTO THE LIBRARYCATALOGUE TABLE
314 | INSERT INTO LibraryCatalogue (ItemStatus, ItemTitle, ItemTypeID, Author, YearPublished, Date_Itmadded, Date_Id)
315 | VALUES
316 | ('Available', 'The Great Gatsby', 1, 'F. Scott Fitzgerald', '1925-04-10', '2022-01-01', NULL, '978-3-16-148410-')
317 | ('On loan', 'To Kill a Mockingbird', 2, 'Harper Lee', '1960-07-11', '2022-01-02', NULL, NULL),
318 | ('Available', 'The Catcher in the Rye', 1, 'J.D. Salinger', '1951-07-16', '2022-01-03', NULL, '978-3-16-148410-')
319 | ('Overdue', 'The Da Vinci Code', 1, 'Dan Brown', '2003-03-18', '2022-01-04', '2023-03-31', '978-3-16-148410-0')
320 | ('On loan', 'Jurassic Park', 3, 'Michael Crichton', '1990-11-20', '2022-01-05', NULL, NULL),
321 | ('On loan', 'The Matrix', 3, NULL, '1999-03-31', '2022-01-06', NULL, NULL),
322 | ('Available', 'Star Wars: Episode IV - A New Hope', 3, NULL, '1977-05-25', '2022-01-07', NULL, NULL),
323 | ('Lost Or Removed', 'The Shawshank Redemption', 1, 'Stephen King', '1982-09-14', '2022-01-08', '2023-02-28', '978-0-380-8152-9')
324 | ('Available', 'The Lord of the Rings: The Fellowship of the Ring', 1, 'J.R.R. Tolkien', '1954-07-29', '2022-01-09', NULL, NULL)
325 | ('On loan', 'Harry Potter and the Philosopher\'s Stone', 1, 'J.K. Rowling', '1997-06-26', '2022-01-10', NULL, NULL)
326 | ('Available', 'Breaking Bad The Complete Series', 3, NULL, '2008-01-20', '2022-01-11', NULL, NULL),
327 | ('On loan', 'Game of Thrones: The Complete First Season', 3, NULL, '2011-04-17', '2022-01-12', NULL, NULL),
328 | ('Available', 'Stranger Things: Season 1', 3, NULL, '2016-07-15', '2022-01-13', NULL, NULL),

```

The output window shows the results of the execution:

```

(15 rows affected)

Completion time: 2023-04-22T05:56:26.7265659+01:00

```

A status bar at the bottom indicates "Query executed successfully."

Fig 6.1.4 Inserting Records into LibraryCatalogue table

The screenshot shows the Microsoft SQL Server Management Studio interface. A query window is open with the following SQL code:

```

334 | --INSERTING RECORDS INTO THE LIBRARYLOAN TABLE
335 | INSERT INTO LibraryLoan (ItemId, MemberId, Date_TakenOut, DueDate)
336 | VALUES
337 | (100, 1, '2022-10-01', '2023-01-15'),
338 | (101, 2, '2022-11-02', '2023-01-16'),
339 | (102, 8, '2022-12-03', '2023-04-25'),
340 | (103, 4, '2023-04-04', '2023-06-10'),
341 | (104, 5, '2023-01-02', '2023-04-19'),
342 | (105, 6, '2023-04-02', '2023-05-20'),
343 | (106, 7, '2023-02-07', '2023-04-03'),
344 | (107, 8, '2023-01-02', '2023-04-22'),
345 | (108, 9, '2022-12-03', '2023-04-24'),
346 | (109, 10, '2022-11-10', '2023-12-12'),
347 | (110, 11, '2023-02-11', '2023-04-30'),
348 | (111, 1, '2023-01-12', '2023-02-26'),
349 | (112, 13, '2022-12-13', '2023-03-27'),

```

The output window shows the results of the execution:

```

(15 rows affected)

Completion time: 2023-04-22T06:02:22.5893335+01:00

```

A status bar at the bottom indicates "Query executed successfully."

Fig 6.1.5 Inserting Records into LibraryLoan table

```

352
353 --INSERTING RECORDS INTO THE OVERDUEFINES_REPAYMENT TABLE
354 INSERT INTO OverdueFines_Repayment (MemberId, LoanId, AmountPaid, RepaymentMethod)
355 VALUES
356 (1, 200, 10.50, 'Cash'),
357 (2, 201, 15.75, 'Card'),
358 (4, 203, 18.25, 'Card'),
359 (7, 206, 14.75, 'Cash'),
360 (9, 208, 11.50, 'Cash'),
361 (1, 211, 16.50, 'Cash'),
362 (13, 212, 9.25, 'Cash'),
363 (4, 213, 19.75, 'Card'),
364 (15, 214, 22.00, 'Cash');
365
366

```

(9 rows affected)

Completion time: 2023-04-22T06:05:50.0790938+01:00

Query executed successfully.

Fig 6.1.5 Inserting Records into OverdueFines_Repayment table

6.2 Displaying Inserted Records in the Tables Using Select Statement

```

367
368 --VIEW RECORDS IN ADDRESS TABLE
369
370
371
372
373
374
375

```

--VIEW RECORDS IN ADDRESS TABLE

SELECT * FROM ADDRESSES

AddressID	Address1	Address2	City	Postcode
1	10 High Street	Flat 1	London	E1 7AB
2	11	25 Oxford Road	Manchester	M1 5AN
3	12	6 Main Street	Birmingham	B3 3HU
4	13	42 Park Lane	Bristol	BS1 5JH
5	14	8 Station Road	Leeds	LS1 4DY
6	15	17 Queen Street	Glasgow	G1 2ED
7	16	14 Market Square	Belfast	B11 2FF
8	17	2 The Grove	Cardiff	CF10 3BA
9	18	33 Highfield Av...	Edinburgh	EH16 5PJ
10	19	1 Victoria Road	Sheffield	S2 2SS
11	20	18 King Street	Liverpool	L1 8HT
12	21	10 Bridge Street	Newcastle...	NE1 8AD
13	22	5 Church Road	Southamp...	S016 7...
14	23	29 Market Street	Nottingham	NG1 6HK
15	24	3 Abbey Gardens	Oxford	OX1 1AK

Query executed successfully.

Fig 6.2.1 View records in the Addresses table

```

--VIEW RECORDS IN MembersInfo TABLE
SELECT * FROM MembersInfo

```

The screenshot shows the results of the query execution. The results table has columns: MemberId, FirstName, MiddleName, LastName, DateOfBirth, Gender, AddressID, Username, PasswordHash, Salt, MemberEmail, and Tele. The data consists of 15 rows, each representing a member with their details.

MemberId	FirstName	MiddleName	LastName	DateOfBirth	Gender	AddressID	Username	PasswordHash	Salt	MemberEmail	Tele
1	John	William	Smith	1985-03-15	Male	10	jwsmith	CAF0DFA8B8EF98E0B6A597DE9A9252F9640B7B142D8746...	CAB9C82B96954774938E2904ED27491E	Jwsmith@gmail.com	+441
2	Sarah	Jane	Doe	1990-07-12	Female	11	sjdoe	0x3D2E64D205C9278BCB010B058920A617D0A37243C0B3...	CAB9C82B96954774938E2904ED27491E	sjdoe@yahoo.com	NUl
3	Adam	Joseph	Brown	1995-11-23	Female	12	abrown	0x65AA0DD7556603242B3D6B68C7ECBBC12891C04CB26A5...	CAB9C82B96954774938E2904ED27491E	mee@gmail.com	NUl
4	Emily	N	Davis	1988-04-30	Female	13	edavis	0x4B529760CE2E03471E1A68E847120B758600B82F90881...	CAB9C82B96954774938E2904ED27491E	edavis@gmail.com	NUl
5	Robert	Michael	Johnson	1979-12-08	Male	14	rjohnson	0xBCCB91D00E9C4B4097026D0E8C7049E815042C9FB6C1...	CAB9C82B96954774938E2904ED27491E	rjohnson@yahoo.com	+441
6	Laura	Grace	Lee	1983-06-10	Female	15	iglee	0x3B7C55839577A56248921B47961806E88709633024F487...	CAB9C82B96954774938E2904ED27491E	iglee@gmail.com	NUl
7	Daniel	N	Wilson	1992-01-17	Male	16	dwilson	0xFB048909991F9C50515E8DC03A50FBD7409347B0D7F6C...	CAB9C82B96954774938E2904ED27491E	dwilson@gmail.com	+441
8	Megan	Elizabeth	Taylor	1987-08-27	Female	17	mtaylor	0x8A0151A9E262B8F7C08E7CE03297B6C0C3885D0F5114902...	CAB9C82B96954774938E2904ED27491E	mtaylor@shoo.com	+441
9	William	Henry	Clark	1980-05-20	Male	18	wclark	0x0233DAA1C72AE350C87702356093488B46F6C051CA3...	CAB9C82B96954774938E2904ED27491E	wclark@hotmail.com	+441
10	Amanda	Rose	Baker	1989-02-25	Female	19	arbakker	0x62E65A29AF34103869586A2448921307E949373A10E4C4B7...	CAB9C82B96954774938E2904ED27491E	ardakke@gmail.com	NUl
11	Steven	N	Anderson	1993-09-14	Male	20	standerson	0x287F53A6D04A721EAD2E2ABC96BF0F161A31E72A5E59F...	CAB9C82B96954774938E2904ED27491E	standerson@yahoo.com	+441
12	Rachel	Nicole	Evanis	1998-06-18	Female	21	mevanis	0xA4594F2913045674586887A7959864537B8E172699453...	CAB9C82B96954774938E2904ED27491E	mevanis@hotmail.com	NUl
13	Matthew	Christopher	Oark	1986-06-22	Male	22	matthewoark	0x1D1D8944F5F8128F4C93A7D24F3290539FCAC91A522887...	CAB9C82B96954774938E2904ED27491E	matthewoark@gmail.com	NUl
14	Sophie	N	Allen	1994-06-18	Female	23	sophiaallen	0x0E2994D84E76A754C817FC2932050E441947AA10C611...	CAB9C82B96954774938E2904ED27491E	sophiaallen@gmail.com	NUl
15	Christopher	N	Green	1993-09-12	Male	24	christophergreen	0x0DD0D503C93C0796F4668505208FB8175671DF2928C23...	CAB9C82B96954774938E2904ED27491E	christophergreen@hotmail.com	+441

Fig 6.2.2 View records in the MembersInfo table

```

--VIEW RECORDS IN ItemTypes TABLE
SELECT * FROM ItemTypes

```

The screenshot shows the results of the query execution. The results table has columns: ItemTypeID and ItemType. The data consists of 4 rows, each representing an item type.

ItemTypeID	ItemType
1	Books
2	Journals
3	DVDs
4	Other Media

Fig 6.2.3 View records in the ItemTypes table

```
--VIEW RECORDS IN Librarycatalogue TABLE
SELECT * FROM Librarycatalogue
```

ItemID	ItemStatus	ItemTitle	ItemTypeID	Author	YearPublished	Date_Received	Date_Identified_asLost_removed	ISBN
100	Available	The Great Gatsby	1	F. Scott Fitzgerald	1925-04-10	2022-01-01	NULL	978-3-16-148410-0
101	On loan	The Lord Chandos Letter	2	Haner Lee	1960-07-11	2022-01-02	NULL	NULL
102	Available	The Catcher in the Rye	1	J.D. Salinger	1951-07-16	2022-01-03	NULL	978-3-16-148410-0
103	Overdue	The Da Vinci Code	1	Dan Brown	2003-03-18	2022-01-04	2023-03-31	978-3-16-148410-0
104	On loan	Jurassic Park	3	Michael Crichton	1990-11-20	2022-01-05	NULL	NULL
105	On loan	The Matrix	3	NULL	1999-03-31	2022-01-06	NULL	NULL
106	Available	Star Wars Episode IV - A New Hope	3	NULL	1977-05-25	2022-01-07	NULL	NULL
107	Lost_Or_Removed	The Shawshank Redemption	1	Stephen King	1986-09-14	2022-01-08	2023-02-28	978-1-43-035595-1
108	Available	The Lord of the Rings: The Fellowship of the Ring	1	J.R.R. Tolkien	1954-07-29	2022-01-09	NULL	978-0-451-52493-5
109	On loan	Harry Potter and the Philosopher's Stone	1	J.K. Rowling	1997-06-26	2022-01-10	NULL	978-3-16-148410-0
110	Available	Breaking Bad The Complete Series	3	NULL	2009-01-20	2022-01-11	NULL	NULL
111	On loan	Game of Thrones: The Complete Final Season	3	NULL	2011-04-17	2022-01-12	NULL	NULL
112	Available	Stranger Things: Season 1	3	NULL	2016-07-15	2022-01-13	NULL	NULL
113	Lost_Or_Removed	1984	1	George Orwell	1949-06-08	2022-01-14	2023-01-31	978-0-451-52493-5
114	Available	The Lord of Castle Black	2	Mario Puzo	1969-03-10	2022-01-15	NULL	NULL

Query executed successfully.

Fig 6.2.4 View records in the LibraryCatalogue table

```
--VIEW RECORDS IN Libraryloan TABLE
SELECT * FROM Libraryloan
```

LoanID	ItemID	MemberID	Date_TakenOut	DueDate	DateReturned	Days_Overage
200	100	1	2023-10-01	2023-01-15	NULL	0
201	101	2	2023-11-02	2023-01-16	NULL	0
202	102	8	2022-12-03	2023-04-25	NULL	0
203	103	4	2023-04-04	2023-04-10	NULL	0
204	104	5	2023-01-02	2023-04-19	NULL	0
205	105	6	2023-04-02	2023-05-20	NULL	0
206	106	7	2023-02-07	2023-04-03	NULL	0
207	107	8	2023-01-02	2023-04-22	NULL	0
208	108	9	2022-12-03	2023-04-24	NULL	0
209	109	10	2022-11-10	2023-12-12	NULL	0
210	110	11	2023-02-11	2023-04-30	NULL	0
211	111	1	2023-01-12	2023-02-26	NULL	0
212	112	13	2022-12-13	2023-03-27	NULL	0
213	113	4	2022-12-03	2023-02-28	NULL	0
214	114	15	2022-12-15	2023-01-29	NULL	0

Query executed successfully.

Fig 6.2.5 View records in the LibraryLoan table

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the 'LibraryDatabase' database is selected. In the center pane, a query window titled 'passwordhashsql - not connected' contains the following SQL code:

```
--VIEW RECORDS IN Libraryloan TABLE
SELECT * FROM OverdueFines_Repayment
```

The results pane displays a table with 9 rows of data:

RepaymentId	MemberId	LoanId	AmountPaid	RepaymentMethod	Date_TimeofPayment
500	1	200	10.50	Cash	2023-04-22 06:05:50.073
501	2	201	15.75	Card	2023-04-22 06:05:50.073
502	4	203	18.25	Card	2023-04-22 06:05:50.073
503	7	206	14.75	Cash	2023-04-22 06:05:50.073
504	9	208	11.50	Cash	2023-04-22 06:05:50.073
505	1	211	16.50	Cash	2023-04-22 06:05:50.073
506	13	212	9.25	Cash	2023-04-22 06:05:50.073
507	4	213	19.75	Card	2023-04-22 06:05:50.073
508	15	214	22.00	Cash	2023-04-22 06:05:50.073

At the bottom of the results pane, a message indicates: 'Query executed successfully.'

Fig 6.2.6 View records in the OverdueFines_Repayment table

6.3 Demonstrate the Database Features

To demonstrate the database, I inserted records into the six tables I created and used this to test and ensure that all SELECT queries, user-defined functions, stored procedures, and triggers are working.

The functionalities of these queries and features had been explained extensively in Parts 2 to 5, here I am just executing the commands.

```

-- A STORED PROCEDURE THAT SEARCHES THE LIBRARYCATALOGUE FOR MATCHING CHARACTERS
CREATE PROCEDURE Search_ItemTitle
    @Title NVARCHAR(100)
AS
BEGIN
    SELECT *
    FROM LibraryCatalogue
    WHERE ItemTitle LIKE '%' + @Title + '%'
    ORDER BY yearpublished DESC;
END
-- The EXEC keyword executes the stored procedure
EXEC Search_ItemTitle 'LORD';

```

ItemID	ItemStatus	ItemTitle	ItemTypeID	Author	YearPublished	Date_Received	Date_Identified_asLost_Removed	ISBN
114	Available	The Lord of the Rings: The Fellowship of the Ring	2	Mario Puzo	1969-03-10	2022-01-15	NULL	NULL
101	On loan	The Lord of the Rings: The Two Towers	2	Harper Lee	1960-07-11	2022-01-02	NULL	NULL
108	Available	The Lord of the Rings: Return of the King	1	J.R.R. Tolkien	1954-07-29	2022-01-09	NULL	978-0-451-52493-5

Fig 6.3.1 Executing the Stored procedure in Part 2a above.

```

-- QUESTION 2(B)
-- A STORED PROCEDURE THAT RETURNS A FULL LIST OF ALL ITEMS CURRENTLY ON LOAN
-- WHICH HAVE A DUE DATE OF LESS THAN FIVE DAYS FROM THE CURRENT DATE
CREATE PROCEDURE items_duedateless_5
AS
SELECT LibraryCatalogue.ItemTitle, LibraryLoan.ItemID, DueDate
FROM LibraryLoan INNER JOIN LibraryCatalogue ON LibraryLoan.ItemID = LibraryCatalogue.ItemID
WHERE DateReturned IS NULL
    AND DATEDIFF(dd, GETDATE(), DueDate) < 0
    AND DATEDIFF(dd, GETDATE(), DueDate) < 5;

```

ItemTitle	ItemID	DueDate
The Catcher in the Rye	102	2023-04-25
The Shawshank Redemption	107	2023-04-22
The Lord of the Rings: The Fellowship of the Ring	108	2023-04-24

Fig 6.3.2 Executing the Stored procedure in Part 2b above.

```

136    END CATCH;
137
138 EXEC Insert_NewMember
139     @Address1 = '6 Victoria Crescent', @Address2 = NULL, @City = 'Leeds',
140     @Postcode = 'LDM25', @Firstname = 'James', @MiddleName = NULL,
141     @LastName = 'Church', @DateOfBirth = '1993-04-21', @Gender = 'Male',
142     @Username = 'REnnY33', @PasswordHarsh = 'Scaramanga',
143     @MemberEmail = 'Manga@gmail.com', @TelephoneNo = '+4478093432',
144     @Member_JoinDate = '2022-01-12'
145
146
147
(1 row affected)

(1 row affected)

```

Query executed successfully.

Fig 6.3.2 Executing the Stored procedure in Part 2c above.

MemberID	Firstname	MiddleName	LastName	DateOfBirth	Gender	AddressID	Username	PasswordHash	Salt	MemberEmail	TelephoneNo
1	John		Doe	1990-07-12	Female	11	sjdoe	0x3D2E64D05C9927B83B...0B058B92A61D7D43ACB3	CAB99CB29639-4774-938E-290AED27491E	sjdoe@yahoo...	NULL
2	Sarah	Jane	Doe	1990-07-12	Female	11	sjdoe...	0x3D2E64D05C9927B83B...0B058B92A61D7D43ACB3	CAB99CB29639-4774-938E-290AED27491E	sjdoe@yahoo...	NULL
3	Adam	Joseph	Brown	1995-11-23	Female	12	abrown...	0x65AAD0DD75566032A2B30B68B7CEBCBC12B91C4DB26A5	CAB99CB29639-4774-938E-290AED27491E	mee@gmail.com	NULL
4	Emily	NULL	Davis	1988-04-30	Female	13	edavis...	0x4B529760CE2EA3471EAE68E8471298756900828290881	CAB99CB29639-4774-938E-290AED27491E	edavis@gmail...	NULL
5	Robert	Michael	Johnson	1979-12-08	Male	14	rjohnson...	0xBCB891D0DE9C4B493702B60ECC7D49E1B1504C28FB69...	CAB99CB29639-4774-938E-290AED27491E	rjohnson@yahoo...	+44757890
6	Laura	Grace	Lee	1983-06-10	Female	15	lglee...	0x3B7C58936F7A52469318A7961E06E8907D9633004F875	CAB99CB29639-4774-938E-290AED27491E	lglee@yahoo...	+4476322378
7	Daniel	NULL	Wilton	1992-01-17	Male	16	dwilson...	0xFb0AB8098991F9C55D95EB0C03483F9BD7049347BDTR82...	CAB99CB29639-4774-938E-290AED27491E	dpwilson@gmail.c...	+447552345
8	Megan	Elizabeth	Taylor	1997-09-27	Female	17	metaylor...	0x940151A8F3C2B0F7C08E7C83297E6C0C8E80EDE114903C...	CAB99CB29639-4774-938E-290AED27491E	metaylor@yahoo...	NULL
9	William	Henry	Cark	1988-05-20	Male	18	whcark...	0x23280AA1C72A4E90C57778023659348B45FC0C12A3...	CAB99CB29639-4774-938E-290AED27491E	whcark@hotmail...	+44753456
10	Amberly	Rose	Baker	1989-02-05	Female	19	arobaker...	0x82E5429F341038E956A4449321307E49473A10EC4B77...	CAB99CB29639-4774-938E-290AED27491E	arobaker@gmail.c...	NULL
11	Steven	NULL	Anderson	1993-09-14	Male	20	standers...	0x287F56340DF4A721EAD2E2A8C6B6BFDF61A31E72AE5E98F...	CAB99CB29639-4774-938E-290AED27491E	standerson@yahoo...	+44754567
12	Rachel	Nicole	Evans	1998-06-18	Female	21	mevans...	0x84B594F291130456F7368897A7958645372B8E1726934374...	CAB99CB29639-4774-938E-290AED27491E	mevans@hotmail...	NULL
13	Matthew	Christopher	Cark	1988-02-22	Male	22	matthew...	0x1D10B944F5F51812F4C59A37D24329059CAFAC91A522870...	CAB99CB29639-4774-938E-290AED27491E	matthewclark@q...	NULL
14	Sophie	NULL	Allen	1994-06-18	Female	23	sophia...	0xD5294D84E76A75ACB17C2393205DE41947AA10C611...	CAB99CB29639-4774-938E-290AED27491E	sophiaallen@gmail...	NULL
15	Christo...	NULL	Green	1993-09-12	Male	24	christop...	0x0DD05038C9C978956F4665052D0F8B1E75671D7F2328C2...	CAB99CB29639-4774-938E-290AED27491E	christophergreen...	+447123409
16	James	NULL	Church	1993-04-21	Male	25	REmY...	0x34CA2002E7B33AE4AAC10D14A9A02E2544E496E79581...	CAB99CB29639-4774-938E-290AED27491E	Manga@gmail.com	+4478093432

Query executed successfully.

Fig 6.3.3 Select query on the two tables displays the new records inserted.

After executing the stored procedure for 2c, two rows were affected as seen in Fig 6.3.2. I then wrote a Select query to view the newly inserted rows in the Addresses and MembersInfo tables, this is shown in Fig 6.3.3.

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the 'LibraryDatabase' database is selected. In the center pane, a query window titled 'passwordhash.sql - not connected' contains the following T-SQL code:

```

166 BEGIN CATCH
167     --Looks like there was an error!
168     IF @@TRANCOUNT > 0
169         ROLLBACK TRANSACTION
170     DECLARE @ErrMsg nvarchar(4000), @ErrSeverity int
171     SELECT @ErrMsg = ERROR_MESSAGE(), @ErrSeverity =
172         ERROR_SEVERITY()
173     RAISERROR(@ErrMsg, @ErrSeverity, 1)
174 END CATCH;
175
176 --The EXEC keyword executes the stored procedure created in 2c
177 EXEC Update_MemberDetails @Username = 'sjdoe', @LastName = 'Majid'

```

The 'Messages' pane at the bottom shows the output:

```

(1 row affected)

Completion time: 2023-04-22T15:07:10.4323381+01:00

```

The status bar at the bottom right indicates 'Query executed successfully.'

Fig 6.3.4 Executing the stored procedure in Part 2d.

After executing the stored procedure, one row was affected as seen in Fig 6.3.4. I then wrote a select query to view the affected row in the MembersInfo table, this is shown in Fig 6.3.5.

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the 'LibraryDatabase' database is selected. In the center pane, a query window titled 'passwordhash.sql - not connected' contains the following T-SQL code:

```

175
176 --The EXEC keyword executes the stored procedure created in 2c
177 EXEC Update_MemberDetails @Username = 'sjdoe', @LastName = 'Majid'
178
179
180

```

Below the code, a 'Results' tab is open, showing the output of the 'SELECT * FROM MembersInfo' query:

MemberId	Firstname	MiddleName	Lastname	DateOfBirth	Gender	AddressID	Username	PasswordHash	Salt	MemberEmail	TelephoneNo	Me...
1	John	Willam	Smith	1985-03-15	Male	10	jwsmith	0xAFD0FAB8B8EF98E0B6A57E0E959A252F9640887B5142D08748E...	CAB98C82-9689-4774-938E-2904ED27491E	Jwsmith@gmail.com	+4475551234	20
2	Sarah	Jane	Majid	1990-07-12	Female	11	sjdoe	0x302E64D205C9927838BC10B058B92D61D7D7A32742C937...	CAB98C82-9689-4774-938E-2904ED27491E	sjdoe@yahoo.com	NULL	20
3	Adam	Joseph	Brown	1995-11-23	Female	12	abrown	0x65AAD0D755666032A83D868BC7ECBBC12B91C04C826AE5...	CAB98C82-9689-4774-938E-2904ED27491E	mee@gmail.com	NULL	20
4	Emily	NULL	Davis	1988-04-30	Female	13	edavis	0x4B529700CE2E6A3471E16E8E847129B758600B8290B81E...	CAB98C82-9689-4774-938E-2904ED27491E	edavis@gmail.com	NULL	20
5	Robert	Michael	Johnson	1979-12-08	Male	14	rjohnson	0xBCCB91DD0E9C4B4970926060C87D749E8190428FB8C1...	CAB98C82-9689-4774-938E-2904ED27491E	mjohnson@yahoo.com	+447557590	20
6	Laura	Grace	Lee	1983-06-10	Female	15	lglee	0x3B7C58936FT56248521BA7961E306E8907D9633084F4075...	CAB98C82-9689-4774-938E-2904ED27491E	dpwilson@gmail.com	+4475552345	20
7	Daniel	NULL	Wilson	1992-01-17	Male	16	dowlson	0xFB0AB090999F15C55095EB0DC03AE09D70409347BD07FCZ...	CAB98C82-9689-4774-938E-2904ED27491E	dpwilson@gmail.com	+44755523278	20
8	Megan	Elizabeth	Taylor	1997-09-27	Female	17	metaylor	0x9A0151A973C7B7C708CE833...	CAB98C82-9689-4774-938E-2904ED27491E	metaylor@yahoo.com	NULL	20
9	William	Henry	Clark	1980-05-20	Male	18	vhark...	0x23300AA1C721E380C07778D23E60934B8646FC5C15CA3...	CAB98C82-9689-4774-938E-2904ED27491E	vhark@hotmail.com	+447553456	20
10	Amanda	Rose	Baker	1989-02-05	Female	19	arbak...	0x2E85A29AF341D3B6596A044932107EA9437A3D0E4877...	CAB98C82-9689-4774-938E-2904ED27491E	arbakera@gmail.com	NULL	20
11	Steven	NULL	Anderson	1993-09-14	Male	20	standers...	0x287553A6DF4721ADE2A3BCB66FD61A317AE5B93FE...	CAB98C82-9689-4774-938E-2904ED27491E	standerson@yahoo...	+447554567	20
12	Rachel	Nicole	Evans	1998-06-18	Female	21	mevans...	0x4B499A29113D456F7C96697A7958645372B8E1726943934...	CAB98C82-9689-4774-938E-2904ED27491E	mevans@hotmail.com	NULL	20
13	Matthew	Christopher	Clark	1986-02-22	Male	22	matthew...	0x1D1DB944F58128F4C93A37D24F329D539CFAC91A228B70...	CAB98C82-9689-4774-938E-2904ED27491E	matthewdclark@gmail...	NULL	20
14	Sophia	NULL	Allen	1994-06-18	Female	23	sophia...	0xD62994D4E76A75ACB17C23932050E41947AA10C6112...	CAB98C82-9689-4774-938E-2904ED27491E	sophiaallen@gmail.com	NULL	20
15	Christo...	NULL	Green	1993-09-12	Male	24	christop...	0xDDDD5038C90CD7896F46685052D8FB1E75671D2928SC2...	CAB98C82-9689-4774-938E-2904ED27491E	christophergreen@ho...	+447123409	20
16	James	NULL	Church	1993-04-21	Male	25	RErnY33	0x3442020E7E8334FE40C1D149A02E5C444E936E79591E...	CAB98C82-9689-4774-938E-2904ED27491E	Menga@gmail.com	+447093432	20

The status bar at the bottom right indicates 'Query executed successfully.'

Fig 6.3.5 Select query that displays the affected row on the MembersInfo table.

The screenshot shows the Microsoft SQL Server Management Studio interface. In the center, there is a large code editor window displaying a T-SQL SELECT statement:

```

196    INNER JOIN LibraryCatalogue ON LibraryLoan.ItemId = LibraryCatalogue.ItemId
197    INNER JOIN MembersInfo ON LibraryLoan.MemberId = MembersInfo.MemberId;
198
199
200    SELECT * FROM View_LoanHistory
201
202

```

Below the code editor is a results grid titled "Results". The grid contains 15 rows of data from the View_LoanHistory:

LoanId	ItemId	ItemTitle	MemberId	FullName	DueDate	DateReturned	Days_Overdue	Overdue_Fine
1	200	The Great Gatsby	1	John William Smith	2022-10-01	NULL	0	970
2	201	The Lord Chandos Letter	2	Sarah Jane Majid	2022-11-02	NULL	0	960
3	202	The Catcher in the Rye	8	Megan Elizabeth Taylor	2022-12-03	NULL	0	0
4	203	The Da Vinci Code	4	Emily Davis	2023-04-04	NULL	0	0
5	204	Jurassic Park	5	Robert Michael Johnson	2023-01-02	NULL	0	30
6	205	The Matrix	6	Laura Grace Lee	2023-04-02	NULL	0	0
7	206	Star Wars: Episode IV - A New Hope	7	Daniel Wilson	2023-02-07	NULL	0	150
8	207	The Shawshank Redemption	8	Megan Elizabeth Taylor	2023-01-02	NULL	0	0
9	208	The Lord of the Rings: The Fellowship of the Ring	9	William Henry Clark	2022-12-03	NULL	0	0
10	209	Harry Potter and the Philosopher's Stone	10	Amanda Rose Baker	2022-11-10	NULL	0	0
11	210	Breaking Bad: The Complete Series	11	Steven Anderson	2023-02-11	NULL	0	0
12	211	Game of Thrones: The Complete First Season	1	John William Smith	2023-01-12	NULL	0	550
13	212	Stranger Things: Season 1	13	Matthew Christopher C...	2022-12-13	NULL	0	260
14	213	1984	4	Emily Davis	2022-12-03	NULL	0	530
15	214	The Lord of the Rings: The Two Towers	15	Christopher Green	2022-12-15	NULL	0	830

At the bottom of the screen, a message bar indicates: "Query executed successfully." and shows the connection details: DESKTOP-QPBRSG4\MSQLSERV... | DESKTOP-QPBRSG4\Thang... | LibraryDatabase | 00:00:00 | 15 rows.

Fig 6.3.6 Select query that displays the created view in Part 3

The screenshot shows the Microsoft SQL Server Management Studio interface. In the center, there is a large code editor window displaying a T-SQL UPDATE statement:

```

214    Where L1.ItemId In (Select ItemId from inserted)
215    and L1.DateReturned IS NOT NULL
216    END
217
218    UPDATE LibraryLoan
219    SET DateReturned = GETDATE()
220    WHERE LOANId = 205
221
222
223
224
225

```

Below the code editor is a results grid titled "Messages". The grid shows two rows affected:

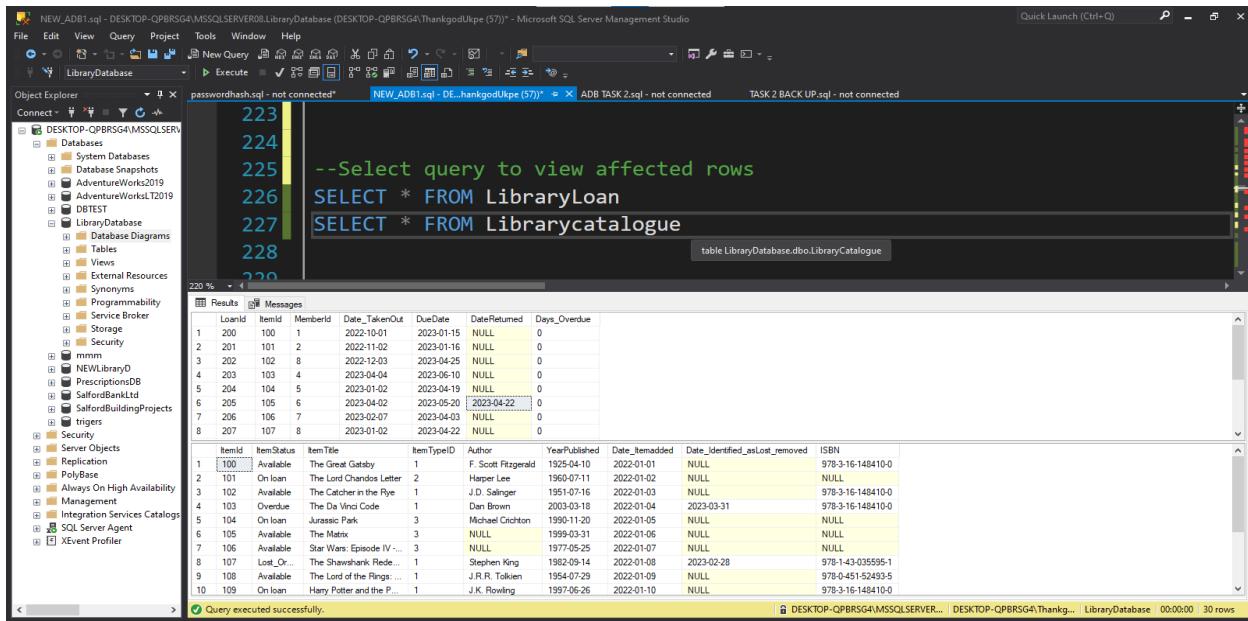
(1 row affected)
(1 row affected)

At the bottom of the screen, a message bar indicates: "Query executed successfully." and shows the connection details: DESKTOP-QPBRSG4\MSQLSERV... | DESKTOP-QPBRSG4\Thang... | LibraryDatabase | 00:00:01 | 0 rows.

Fig 6.3.7 Update on the Libraryloan table to fire the Trigger created in Part 4

After updating the Libraryloan table, two rows were affected as shown in Fig 6.3.7. The first row is the update on the Datereturned column in the Libraryloan table and the second

row affected is the Itemstatus column in the LibraryCatalogue table. I then wrote a Select query on the two tables to view the affected rows, this is shown in Fig 6.3.8.



--Select query to view affected rows

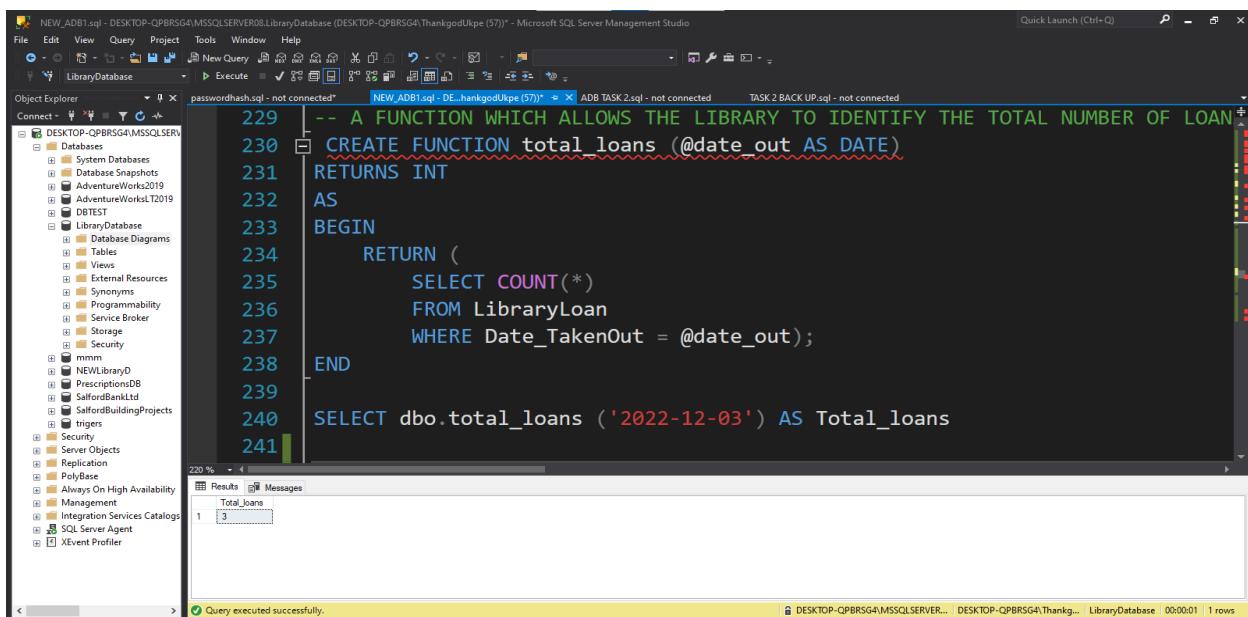
```
SELECT * FROM LibraryLoan
SELECT * FROM Librarycatalogue
```

LoanId	ItemId	MemberId	Date_TakenOut	DueDate	DateReturned	Days_Overdue	
1	200	100	2022-10-01	2023-01-15	NULL	0	
2	201	101	2022-11-02	2023-01-16	NULL	0	
3	202	102	2022-12-03	2023-04-25	NULL	0	
4	203	103	4	2023-04-04	2023-06-10	NULL	0
5	204	104	5	2023-01-02	2023-04-19	NULL	0
6	205	105	6	2023-04-02	2023-05-20	2023-04-22	0
7	206	106	7	2023-02-07	2023-04-03	NULL	0
8	207	107	8	2023-01-02	2023-04-22	NULL	0

ItemId	ItemStatus	ItemTitle	ItemTypeID	Author	YearPublished	Date_Remanded	Date_Identified_asLost	ISBN
1	Available	The Great Gatsby	1	F. Scott Fitzgerald	1925-04-10	2022-01-01	NULL	978-0-14-10273-4
2	On loan	The Lord Chandos Letter	2	Homer Lee	1960-07-11	2022-01-02	NULL	978-0-14-10273-4
3	Available	The Catcher in the Rye	1	J.D. Salinger	1951-07-16	2022-01-03	NULL	978-0-14-10273-4
4	On loan	The Da Vinci Code	1	Dan Brown	2003-03-18	2022-01-04	2023-03-31	978-0-14-10273-4
5	On loan	Jurassic Park	3	Michael Crichton	1990-11-20	2022-01-05	NULL	978-0-14-10273-4
6	Available	The Matrix	3	NULL	1999-03-31	2022-01-06	NULL	978-0-14-10273-4
7	Available	Star Wars: Episode IV - A New Hope	3	George Lucas	1977-05-25	2022-01-07	NULL	978-0-14-10273-4
8	Lost Or	The Shawshank Redemption	1	Stephen King	1982-09-14	2022-01-08	2023-02-28	978-0-43-035595-1
9	Available	The Lord of the Rings: The Fellowship of the Ring	1	J.R.R. Tolkien	1954-07-29	2022-01-09	NULL	978-0-45-152493-5
10	On loan	Harry Potter and the Philosopher's Stone	1	J.K. Rowling	1997-06-26	2022-01-10	NULL	978-0-14-10273-4

Query executed successfully.

Fig 6.3.8 Select query to view the affected rows.



```
-- A FUNCTION WHICH ALLOWS THE LIBRARY TO IDENTIFY THE TOTAL NUMBER OF LOANS
CREATE FUNCTION total_loans (@date_out AS DATE)
RETURNS INT
AS
BEGIN
    RETURN (
        SELECT COUNT(*)
        FROM LibraryLoan
        WHERE Date_TakenOut = @date_out);
END
```

```
SELECT dbo.total_loans ('2022-12-03') AS Total_loans
```

Total_loans
1
3

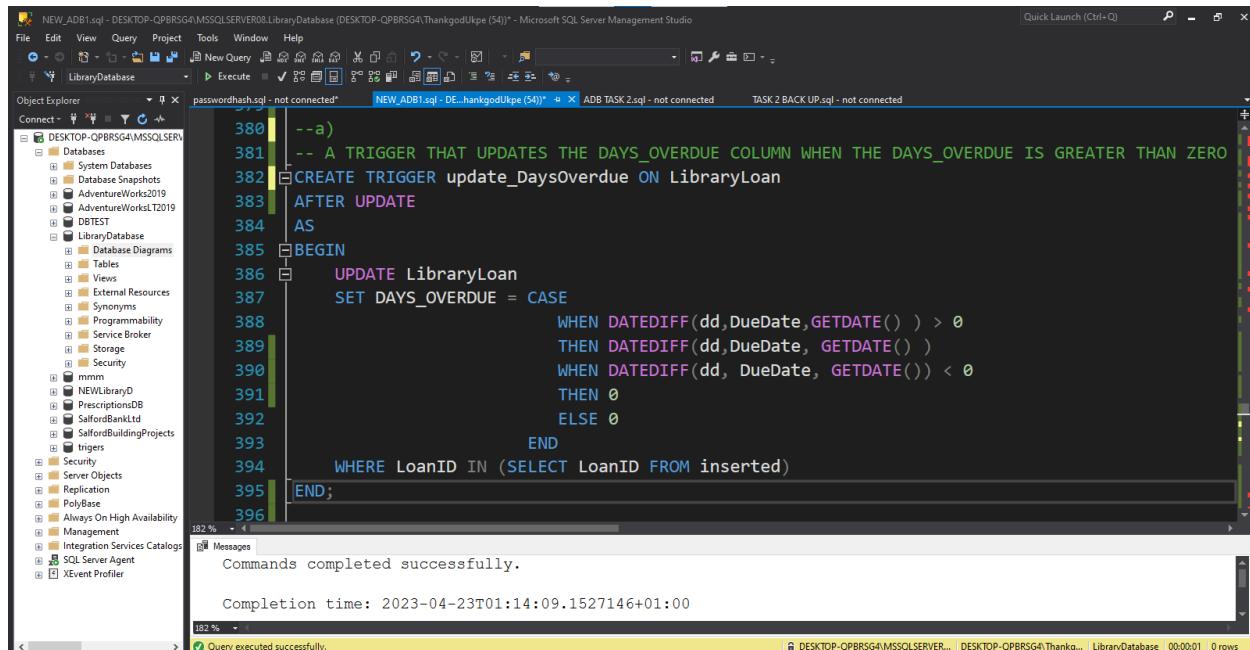
Query executed successfully.

Fig 6.3.9 Select query calls the function created in Part 5 and returns the result.

PART 7

Extra Database Objects that Will be Useful to the Library

- a) I created a trigger update_Daysoverdue on the LibraryLoan table. This trigger updates the Days_Overdue column when there is an update on the DueDate column in the LibraryLoan table. The trigger uses a Case statement to specify some conditions for days_overdue for each loan. The first condition uses the Datediff function to calculate the difference in days between the due date and the current date, if result is greater than zero then the value is returned as days_overdue. If the result is equal to or less than zero, the number of days overdue is set to zero. This is shown in fig 7a.



The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the 'LibraryDatabase' database is selected. In the center pane, a script window displays the creation of a trigger:

```
380 --a)
381 -- A TRIGGER THAT UPDATES THE DAYS_OVERDUE COLUMN WHEN THE DAYS_OVERDUE IS GREATER THAN ZERO
382 CREATE TRIGGER update_DaysOverdue ON LibraryLoan
383 AFTER UPDATE
384 AS
385 BEGIN
386     UPDATE LibraryLoan
387     SET DAYS_OVERDUE = CASE
388         WHEN DATEDIFF(dd,DueDate,GETDATE()) > 0
389             THEN DATEDIFF(dd,DueDate, GETDATE())
390         WHEN DATEDIFF(dd, DueDate, GETDATE()) < 0
391             THEN 0
392         ELSE 0
393     END
394     WHERE LoanID IN (SELECT LoanID FROM inserted)
395 END;
```

The status bar at the bottom indicates "Commands completed successfully." and "Completion time: 2023-04-23T01:14:09.1527146+01:00".

Fig 7a

To demonstrate this trigger, I updated the DueDate column in the Libraryloan table as seen in Fig 7a1. After updating, I wrote a select statement to view the affected row, this is shown in Fig 7a2.

```

389     THEN DATEDIFF(dd,DueDate, GETDATE() )
390     WHEN DATEDIFF(dd, DueDate, GETDATE()) < 0
391     THEN 0
392     ELSE 0
393
394     WHERE LoanID IN (SELECT LoanID FROM inserted)
395 END;
396
397 UPDATE LIBRARYLOAN
398 SET DUEDATE = '2023-4-15'
399 WHERE LOANID = 211
400

```

(1 row affected)

(1 row affected)

Completion time: 2023-04-23T01:38:38.6480676+01:00

Query executed successfully.

Fig 7a1

```

391     THEN 0
392     ELSE 0
393
394     WHERE LoanID IN (SELECT LoanID FROM inserted)
395 END;
396
397 UPDATE LIBRARYLOAN
398 SET DUEDATE = '2023-4-15'
399 WHERE LOANID = 211
400
401 select * from libraryloan
402

```

LoanId	Itemid	MemberId	Date_TakenOut	DueDate	DateReturned	Days_Overdue
5	204	104	5	2023-01-02	2023-04-19	NULL
6	205	105	6	2023-04-02	2023-05-20	2023-04-22
7	206	106	7	2023-02-07	2023-04-03	NULL
8	207	107	8	2023-01-02	2023-04-22	NULL
9	208	108	9	2022-12-03	2023-04-24	NULL
10	209	109	10	2022-11-10	2023-12-12	NULL
11	210	110	11	2023-02-11	2023-04-30	NULL
12	211	111	1	2023-01-12	2023-04-15	NULL
13	212	112	13	2022-12-13	2023-03-27	NULL
14	213	113	4	2022-12-03	2023-02-28	NULL
15	214	114	15	2022-12-15	2023-01-29	NULL

Query executed successfully.

Fig 7a2

b) I created a stored procedure Outstanding_balance to calculate and return the total overdue fine, total amount paid and any outstanding balance of a member. This stored procedure takes an input parameter @memberId of datatype ‘int’. The procedure uses left join to connect the LibraryLoan table with the OverdueFines_Repayment table on the LoanId column. The select statement uses the SUM function to calculate the total overdue, which is the product of Days_overdue of each loan and the @Overduefee_rate variable that was declared. The SUM function also calculates the total amount paid which is the sum of the AmountPaid column from the OverdueFines_Repayment table. The outstanding balance is calculated as the difference between the total overdue fine and total amount paid. This is shown in Fig 7b.

```

403 --b)
404 -- A STORED PROCEDURE THAT RETURNS THE THE TOTAL_OVERDUEFINE, TOTA_AMOUNT PAID,
405 --AND OUTSTANDING BALANCE OF A MEMBER
406
407 CREATE PROCEDURE Outstanding_balance @memberID int
408 AS
409 BEGIN
410     DECLARE @Overduefee_rate MONEY;
411     SET @Overduefee_rate = 10;
412     SELECT sum(L.Days_Overdue * @Overduefee_rate) AS Total_Overduefine,sum(R.AmountPaid) AS Total_Amount_Paid
413     ,(sum(L.Days_Overdue * @Overduefee_rate) - sum(R.AmountPaid)) AS Outstanding_Balance
414     From LibraryLoan L
415     LEFT JOIN OverdueFines_Repayment R ON R.LoanID = L.LoanID
416     WHERE L.MemberId = @memberID
417 END
418

```

Commands completed successfully.
Completion time: 2023-04-23T02:10:44.4069564+01:00

Query executed successfully.

Fig 7b

To demonstrate how this stored procedure works, I used the Exec keyword to execute the stored procedure with an input parameter 1, this is shown in Fig 7b1.

```

412 | SELECT          SUM(L.Days_Overdue * @Overduefee_rate) AS Total_OverdueTine, SUM(R.AmountPaid) AS Tot
413 | (sum(L.Days_Overdue * @Overduefee_rate) - sum(R.AmountPaid)) AS Outstanding_Balance
414 |
415 | From LibraryLoan L
416 | LEFT JOIN OverdueFines_Repayment R ON R.LoanID = L.LoanID
417 | WHERE L.MemberId = @memberID
418 |
419 | END
420 |
421 | Exec Outstanding_balance @memberID =1

```

Total_Overage	Totalamount_paid	Outstanding_Balance
80.00	27.00	53.00

Query executed successfully.

Fig 7b1

- c) I created a trigger Date_Identified_asLost on the Librarycatalogue table, this trigger will fire and update the Date_Identified_asLost_removed column when the Itemstatus column is updated. The trigger uses the case statement to check whether the Itemstatus column in the updated row is set to Lost_or_Removed. If true, the Date_Identified_asLost_removed column in LibraryCatalogue table is updated to the current date and time using the Getdate() function. This is shown in Fig 7c.

```

422 -- C
423 -- A TRIGGER TO UPDATE THE DATE_IDENTIFIED_ASLOST COLUMN IN THE LIBRARY CATALOGUE TABLE
424 -- WHEN ITEMSTATUS IS LOST_OR_REMOVED
425
426 CREATE TRIGGER Date_Identified_asLost ON Librarycatalogue
427 AFTER UPDATE
428 AS
429 BEGIN
430     UPDATE lc
431     SET lc.Date_Identified_asLost_removed =
432         CASE
433             WHEN lc.Itemstatus = 'Lost_Or_Removed' THEN GETDATE()
434         END
435     FROM inserted i
436     INNER JOIN LibraryCatalogue lc ON lc.ItemId = i.ItemId
437 END

```

Commands completed successfully.

Completion time: 2023-04-23T02:59:38.9140109+01:00

Query executed successfully.

Fig 7c

To demonstrate how the trigger works, I updated the Librarycatalogue table as shown in Fig 7c1. I then wrote a select query to view the updated row in the Librarycatalogue table, this is shown in Fig 7c2.

```

435 FROM inserted i
436     INNER JOIN LibraryCatalogue lc ON lc.ItemId = i.ItemId
437
438
439
440 update LibraryCATALOGUE
441 set ItemStatus = 'Lost_Or_Removed'
442 where itemid = 100
443
444

```

(1 row affected)

(1 row affected)

Completion time: 2023-04-23T03:25:20.4770724+01:00

Query executed successfully.

Fig 7c1

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the database 'LibraryDatabase' is selected. In the center pane, a query window titled 'passwordhashsql - not connected' contains the following T-SQL code:

```

435 |     FROM inserted i
436 |     INNER JOIN LibraryCatalogue lc ON lc.ItemId = i.ItemId
437 |   END
438 |
439 | update LibraryCATALOGUE
440 | set ItemStatus = 'Lost_Or_Removed'
441 | where itemid = 100
442 |
443 |
444 |
445 |
446 | select * from LibraryCatalogue
447 |
448 |

```

Below the code, the results grid displays the following data:

ItemID	ItemStatus	ItemTitle	ItemTypeID	Author	YearPublished	Date_Released	Date_Identified_asLost_removed	ISBN
100	Lost_Or_Removed	The Great Gatsby	1	F. Scott Fitzgerald	1925-04-10	2022-01-01	2023-04-23	978-3-16-148410-0
101	On loan	The Lord Chando's Letter	2	Harper Lee	1960-07-11	2022-01-02	NULL	NULL
102	Available	The Catcher in the Rye	1	J.D. Salinger	1951-07-16	2022-01-03	NULL	978-3-16-148410-0
103	Overdue	The Da Vinci Code	1	Dan Brown	2003-03-18	2022-01-04	2023-03-31	978-3-16-148410-0
104	On loan	Jurassic Park	3	Michael Crichton	1990-11-20	2022-01-05	NULL	NULL
105	Available	The Matrix	3	NULL	1999-03-31	2022-01-05	NULL	NULL
106	Available	Star Wars Episode IV - A New Hope	3	NULL	1977-05-25	2022-01-07	NULL	NULL
107	Lost_Or_Removed	The Shawshank Redemption	1	Stephen King	1982-09-14	2022-01-08	2023-02-28	978-1-43035595-1
108	Available	The Lord of the Rings: The Fellowship of the Ring	1	J.R.R. Tolkien	1954-07-29	2022-01-09	NULL	978-0-451-52493-5

At the bottom of the results grid, it says 'Query executed successfully.'

Fig 7c2

d) I created a table named Former_Members, shown in Fig 7d.

I then created a trigger MoveToFormerMembers on the MembersInfo table, before creating the trigger, this trigger checks if there is an update on the MemberEnd_Date column in the MembersInfo table. If true, it moves the member information into the Former_Members table. The trigger also deletes the member's records in OverdueFines_Repayment and LibraryLoan tables and finally deletes the member's information from the MembersInfo table. This is shown in Fig 7d1.

To demonstrate how this trigger works, I updated the Member_Enddate column in the MembersInfo table, Fig 7d2 shows the number of rows that were affected. I then wrote a select query on the MembersInfo and Former_Members tables to view the affected rows. As seen in Fig 7d3, the information of MemberId 1 has been deleted from the MembersInfo table and moved to Former_Members table.

File Edit View Query Project Tools Window Help

New Query Execute

Object Explorer

LibraryDatabase

```

425 --d
426 -- A TRIGGER THAT WILL MOVE THE MEMBERS THAT LEAVE THE LIBRARY INTO A FORMER MEMBERS TABLE
427 --Create Former_Members table
428 CREATE TABLE FORMER_MEMBERS(
429     MemberId int IDENTITY(1,1) NOT NULL PRIMARY KEY,
430     Fisrtname nvarchar(30) NOT NULL,
431     MiddleName nvarchar(30) NULL,
432     LastName nvarchar(30) NOT NULL,
433     DateofBirth Date NOT NULL,
434     Gender nvarchar(20) NOT NULL ,
435     AddressID Int NOT NULL FOREIGN KEY (AddressID)
436     REFERENCES Addresses(AddressID),
437     Username nvarchar(30) UNIQUE NOT NULL,
438     PasswordHarsh binary(64) NOT NULL,
439     Salt UNIQUEIDENTIFIER,
440     MemberEmail nvarchar(100) NULL, CHECK(MemberEmail Like '%@%.-%'),
441     TelephoneNo nvarchar(20) NULL,
442     Member_JoinDate Date NOT NULL,
443     Member_EndDate Date NULL);
444

```

Messages

Commands completed successfully.

Completion time: 2023-04-28T09:30:27.1386136+01:00

Query executed successfully.

Fig 7d

File Edit View Query Project Tools Window Help

New Query Execute

Object Explorer

LibraryDatabase

```

465 CREATE TRIGGER MoveToFormerMembers
466 ON MembersInfo
467 AFTER UPDATE
468 AS
469 BEGIN
470 IF UPDATE (MEMBER_ENDDATE)
471 BEGIN
472     INSERT INTO Former_Members(Fisrtname, MiddleName, LastName, DateofBirth, Gender, AddressID, Username, PasswordHarsh, Salt, MemberEmail)
473     SELECT Fisrtname, MiddleName, LastName, DateofBirth, Gender, AddressID, Username, PasswordHarsh, Salt, MemberEmail
474     FROM inserted
475     DELETE FROM OverdueFines_Repayment
476     WHERE MemberId IN
477         (SELECT MemberId FROM inserted)
478     DELETE FROM LibraryLoan
479     WHERE MemberId IN
480         (SELECT MemberId FROM inserted)
481     DELETE FROM MembersInfo
482     WHERE MemberId IN
483         (SELECT MemberId FROM inserted);
484 END
485 END

```

Messages

Commands completed successfully.

Completion time: 2023-04-23T04:56:44.0031232+01:00

Query executed successfully.

Fig 7d1

NEW_ADB1.sql - DESKTOP-QPBRSG4\MSQLSERVER08\LibraryDatabase (DESKTOP-QPBRSG4\ThankgodUkpe (54)) - Microsoft SQL Server Management Studio

```

474    FROM inserted
475    DELETE FROM OverdueFines_Repayment
476      WHERE MemberId IN
477        (SELECT MemberId FROM inserted)
478    DELETE FROM LibraryLoan
479      WHERE MemberId IN
480        (SELECT MemberId FROM inserted)
481    DELETE FROM MembersInfo
482      WHERE MemberId IN
483        (SELECT MemberId FROM inserted);
484  END
485 END
486
487 UPDATE MembersInfo
488 SET Member_EndDate = GETDATE()
489 WHERE MemberId = 1
490
491

```

(1 row affected)

(2 rows affected)

(2 rows affected)

Query executed successfully.

Fig 7d2

NEW_ADB1.sql - DESKTOP-QPBRSG4\MSQLSERVER08\LibraryDatabase (DESKTOP-QPBRSG4\ThankgodUkpe (54)) - Microsoft SQL Server Management Studio

```

492
493
494
495 SELECT * FROM MembersInfo
496 SELECT * FROM FORMER_MEMBERS
497
498
499
500
501
502
503

```

Results

MemberId	Firstname	MiddleName	LastName	DatedOfBirth	Gender	AddressID	Username	PasswordHash	Salt	MemberEmail
1	Sarah	Jane	Majid	1990-07-12	Female	11	sjode	0x3D2E6402D95978B8BC10B058B920A410D70A37243C837...	CAB9BCB2-9689-4774-438E-2904ED27491E	sjode@yahoo.com
2	Adam	Joseph	Brown	1995-01-23	Female	12	ajbrown	0x65AA0DD7556603242B30668BC7ECBBC12B91C04C826A5...	CAB9BCB2-9689-4774-438E-2904ED27491E	mee@gmail.com
3	Emily	NULL	Davis	1988-04-30	Female	13	edavis	0x4B529760CEAE3471E1A16E8E347128785800828790881...	CAB9BCB2-9689-4774-438E-2904ED27491E	edavis@gmail.com
4	Robert	Michael	Johnson	1975-12-08	Male	14	rJohnson	0xBCCB9100E9C4B4D970286D7ECBD749EB1504C28FBFBC1...	CAB9BCB2-9689-4774-438E-2904ED27491E	mjohnson@yahoo.com
5	Laura	Grace	Lee	1983-06-10	Female	15	lglee	0x387C559367456248921B7961E00E89709633084F4875...	CAB9BCB2-9689-4774-438E-2904ED27491E	dpwilson@gmail.com
6	Daniel	NULL	Wilson	1992-01-17	Male	16	dpwilson	0xF804B909991F155059EBCDC03A0E9FD7409347BD07FC2...	CAB9BCB2-9689-4774-438E-2904ED27491E	dpwilson@gmail.com
7	Megan	Elizabeth	Taylor	1997-06-27	Female	17	metaylor	0x9A0151AB3C289F7C007C832978EC0C38E0D8E14903C...	CAB9BCB2-9689-4774-438E-2904ED27491E	metaylor@yahoo.com
8	William	Henry	Clark	1980-05-20	Male	18	whclark	0x023300AAC172AE50CB778023E60934886468C1051CA3...	CAB9BCB2-9689-4774-438E-2904ED27491E	whclark@hotmail.com
9	Ananda	Rose	Baker	1989-02-05	Female	19	anrbaker	0x82E95A29AF34103B69586A244B921307E94373A10EC4B77...	CAB9BCB2-9689-4774-438E-2904ED27491E	anrbaker@gmail.com
10	Steven	NULL	Anderson	1993-06-14	Male	20	stAnderson	0x27F5K3467F4A771FAF7A9RCR6RF1F61A1F72A5F598F...	CAR9RCR2-9689-4774-438E-2904ED27491F	stAnderson@yahoo.com
11										

Results

MemberId	Firstname	MiddleName	LastName	DatedOfBirth	Gender	AddressID	Username	PasswordHash	Salt	MemberEmail	TelephoneNo	Member...
1	John	William	Smith	1985-03-15	Male	10	jwsmith	0xAFD0FABBBEEF98E0B6A97E0E99A252F9640887651420874...	CAB9BCB2-9689-4774-438E-2904ED27491E	Jwsmith@gmail.com	+447551234	2023-01

Query executed successfully.

Fig 7d3

PART 8 Advice and Guidance

8.1 Data Integrity and Concurrency

Data integrity is a concept and process that ensures the accuracy, completeness, consistency, validity data. Different measures were taken to enforce data integrity in the Library Database. I used Primary key constraints for each entity to ensure entity constraints, meaning that each table in the database will have a unique identifier. I used foreign key constraints to enforce referential integrity, this ensures that data in related tables is consistent and prevents any inconsistency. I used unique constraint for the username column to ensure that duplicate values are not entered for this field.

I used check constraints on the fields like member email address and item status to ensure that characters or variables other from the one specified are not entered for these fields.

Concurrency is the process of managing multiple operations in the database simultaneously without having them conflict and leading to inconsistencies. I used transaction management in stored procedure (Insert_Newmember) to ensure that the stored procedure does not violate the ACID properties of the transaction and not lead to inconsistencies in data.

8.2 Database Security

Database security is the mechanism that protects the database against intentional or accidental loss, destruction or misuse.

Several measures were put in place to enforce proper security for the database system.

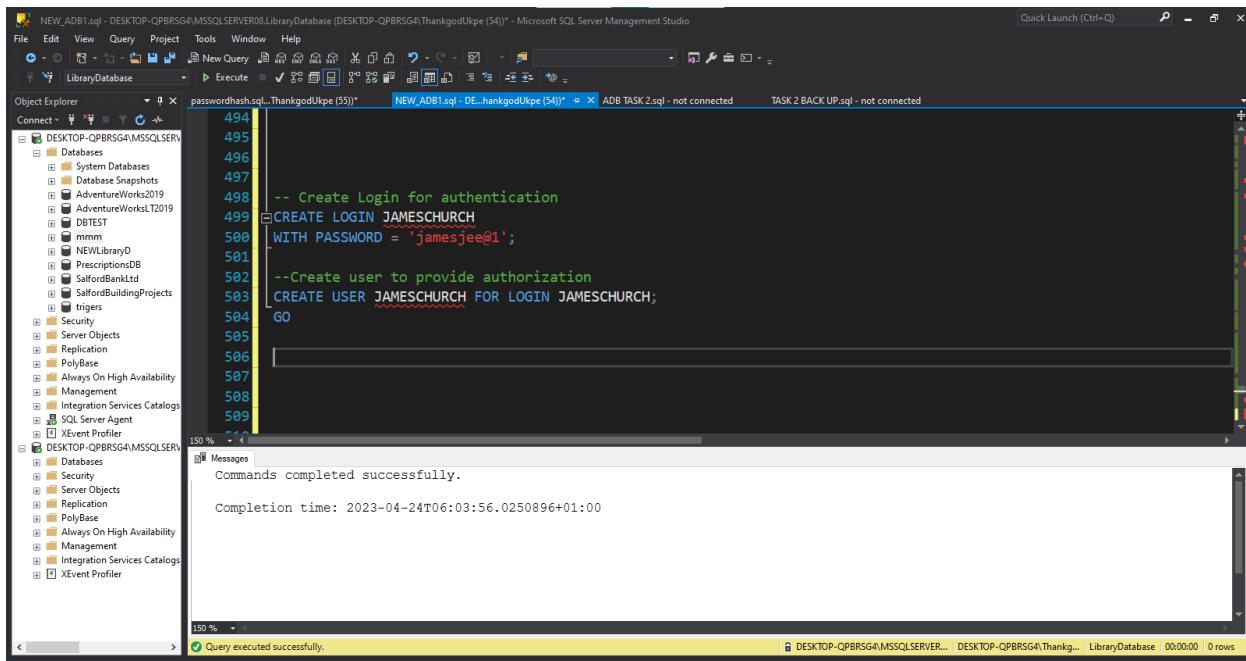
8.2.1 Password Hashing

I created a column named ‘passwordhash’ in the Membersinfo table which takes different characters but does not store the password as they are inputted. The one-way harsh function takes the password and transforms it into encrypted string of characters that is used to verify the authenticity of a user during login process. In case of an attack on the database, the hashing mechanism is an adequate method of storing passwords as it is

computationally infeasible to reverse the hash and determine the original passwords. This helps in enforcing database security.

8.2.2 Creating Login and Database User

I created a login for users to help verify the authenticity of users. Users have unique usernames associated with passwords which must be supplied before they gain access to the SQL server instance. I created a database user which gives a user authorization to access the database. This is shown in Fig 8.2.2



The screenshot shows the Microsoft SQL Server Management Studio (SSMS) interface. In the Object Explorer, two servers are listed: 'DESKTOP-QPBRSG4\SQLSERV08' and 'DESKTOP-QPBRSG4\SQLSERV09'. Under 'DESKTOP-QPBRSG4\SQLSERV08', several databases are visible, including 'AdventureworksLT2019', 'DBTEST', 'master', 'model', 'msdb', 'NEWLibrary0', 'PrescriptionDB', 'SalfordBankLtd', 'SalfordBuildingProjects', and 'triggers'. Under 'DESKTOP-QPBRSG4\SQLSERV09', databases like 'Adventureworks2019', 'AdventureworksLT2019', 'DBTEST', 'master', 'model', 'msdb', 'NEWLibrary0', 'PrescriptionDB', 'SalfordBankLtd', 'SalfordBuildingProjects', and 'triggers' are listed. The 'Messages' pane at the bottom displays the following output:

```
494  
495  
496  
497  
498 -- Create Login for authentication  
499 CREATE LOGIN JAMESCHURCH  
500 WITH PASSWORD = 'jamesjee@1';  
501  
502 --Create user to provide authorization  
503 CREATE USER JAMESCHURCH FOR LOGIN JAMESCHURCH;  
504 GO  
505  
506  
507  
508  
509  
  
Commands completed successfully.  
Completion time: 2023-04-24T06:03:56.0250896+01:00  
  
Query executed successfully.
```

Fig 8.2.2

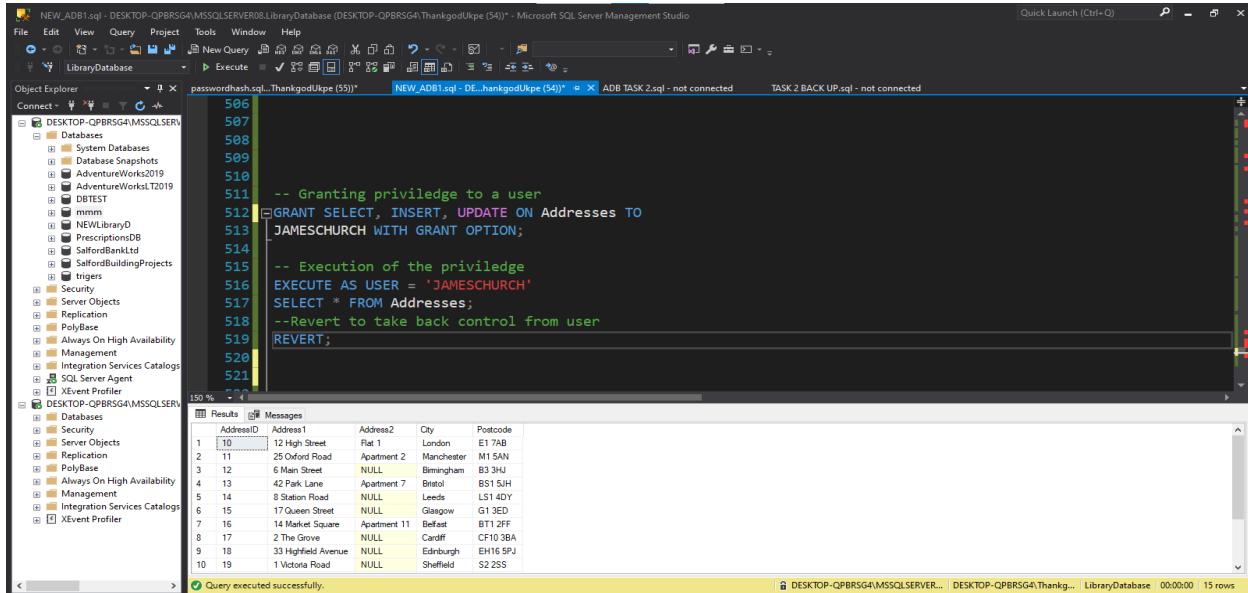
8.2.3 Creating Roles and Granting Privileges

After creating the database user, I used a grant statement to grant user permissions on the Address table. As shown in Fig 8.2.3, the user can only select, insert and update data on the Address table.

I created a role and granted select, insert and update permissions to the role. The role allows the addition of multi users who may require the same set of privileges. I used the Deny statement to deny the role or users from subsequently being granted the delete

permission. I granted the role permission to execute the stored procedure created in 2b and added a new member to the role. This is shown in Fig 8.2.3.2

These are Authentication and Authorization mechanisms which help in enforcing database security.



```

506
507
508
509
510
511 -- Granting privilege to a user
512 GRANT SELECT, INSERT, UPDATE ON Addresses TO
513 JAMESCHURCH WITH GRANT OPTION;
514
515 -- Execution of the privilege
516 EXECUTE AS USER = 'JAMESCHURCH'
517 SELECT * FROM Addresses;
518 -- Revert to take back control from user
519 REVERT;
520
521

```

The screenshot shows the SQL Server Management Studio interface. The Object Explorer on the left lists several databases, including 'NEWLibraryDB' and 'PrescriptionDB'. The central pane displays the following SQL script:

```

-- Granting privilege to a user
GRANT SELECT, INSERT, UPDATE ON Addresses TO
JAMESCHURCH WITH GRANT OPTION;

-- Execution of the privilege
EXECUTE AS USER = 'JAMESCHURCH'
SELECT * FROM Addresses;
-- Revert to take back control from user
REVERT;

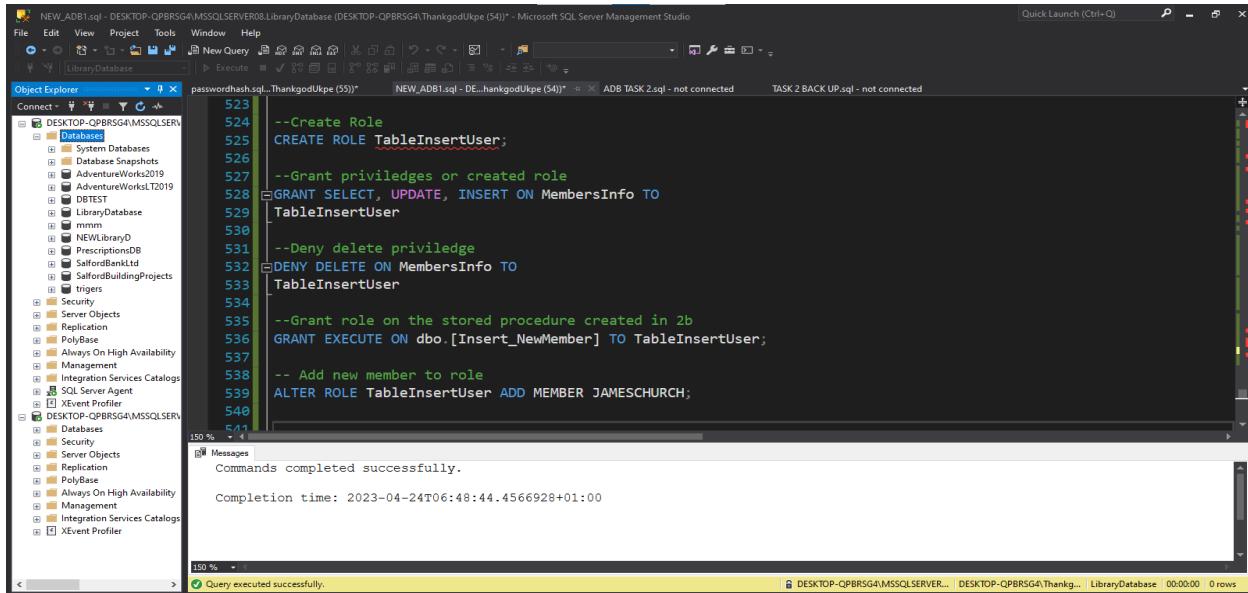
```

Below the script, the 'Results' tab shows the output of the 'SELECT' query:

AddressID	Address1	Address2	City	Postcode
1	10	12 High Street	Riot 1	London E1 7AB
2	11	25 Oxford Road	Apartment 2	Manchester M1 5AN
3	12	6 Main Street	NULL	Birmingham B3 3HJ
4	13	42 Park Lane	Apartment 7	Bristol BS1 5JH
5	14	8 Station Road	NULL	Leeds LS1 4DY
6	15	17 Queen Street	NULL	Glasgow G1 3ED
7	16	14 Market Square	Apartment 11	Belfast BT12 2FF
8	17	2 The Grove	NULL	Cardiff CF10 3BA
9	18	33 Highfield Avenue	NULL	Edinburgh EH16 5PJ
10	19	1 Victoria Road	NULL	Sheffield S2 2SS

A message at the bottom of the results pane says 'Query executed successfully.'

Fig 8.2.3.1



```

523
524 --Create Role
525 CREATE ROLE TableInsertUser;
526
527 --Grant privileges or created role
528 GRANT SELECT, UPDATE, INSERT ON MembersInfo TO
529 TableInsertUser;
530
531 --Deny delete privilege
532 DENY DELETE ON MembersInfo TO
533 TableInsertUser;
534
535 --Grant role on the stored procedure created in 2b
536 GRANT EXECUTE ON dbo.[Insert_NewMember] TO TableInsertUser;
537
538
539
540 -- Add new member to role
ALTER ROLE TableInsertUser ADD MEMBER JAMESCHURCH;
541

```

The screenshot shows the SQL Server Management Studio interface. The Object Explorer on the left lists several databases, including 'NEWLibraryDB' and 'PrescriptionDB'. The central pane displays the following SQL script:

```

--Create Role
CREATE ROLE TableInsertUser;

--Grant privileges or created role
GRANT SELECT, UPDATE, INSERT ON MembersInfo TO
TableInsertUser;

--Deny delete privilege
DENY DELETE ON MembersInfo TO
TableInsertUser;

--Grant role on the stored procedure created in 2b
GRANT EXECUTE ON dbo.[Insert_NewMember] TO TableInsertUser;

-- Add new member to role
ALTER ROLE TableInsertUser ADD MEMBER JAMESCHURCH;

```

The 'Messages' tab at the bottom shows the message 'Commands completed successfully.' and the completion time 'Completion time: 2023-04-24T06:48:44.4566928+01:00'.

Fig 8.2.3.2

8.2.3 Database Backup and Recovery

Database backup and recovery are important aspects of database management, as they ensure that data is protected, and that the database can be quickly restored in the event of a problem.

I have provided a backup file for the library database in case of system failure.

CONCLUSION

The database stores information of members and items. Keeps track of loans, fines and fine repayments in the library.

The library has a lot of functionalities such as:

- Keeping track of all items on loan, inserting new records into the MembersInfo table and likewise updating information on it.
- The library has a function that updates the Itemstatus when an item is available making it possible for members to borrow it.
- The library has a function that generates the total number of loans taken out on a specific date.
- It has a function that calculates the number of days overdue and updates the Days_Overdue column when a loan is overdue.
- The library also has a special function that computes the total overdue fine of a member, amount paid and any outstanding balance.
- The library has a function that records the date and time an item is identified to be lost.
- The library has a function that deletes records of non active members from the membersInfo table and moves them to Former_members table.
- The library has mechanisms put in place to authenticate and provide authorization to access the database.
- The library has a backup and recovery mechanism put in place incase a database failure.