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_enddata 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.

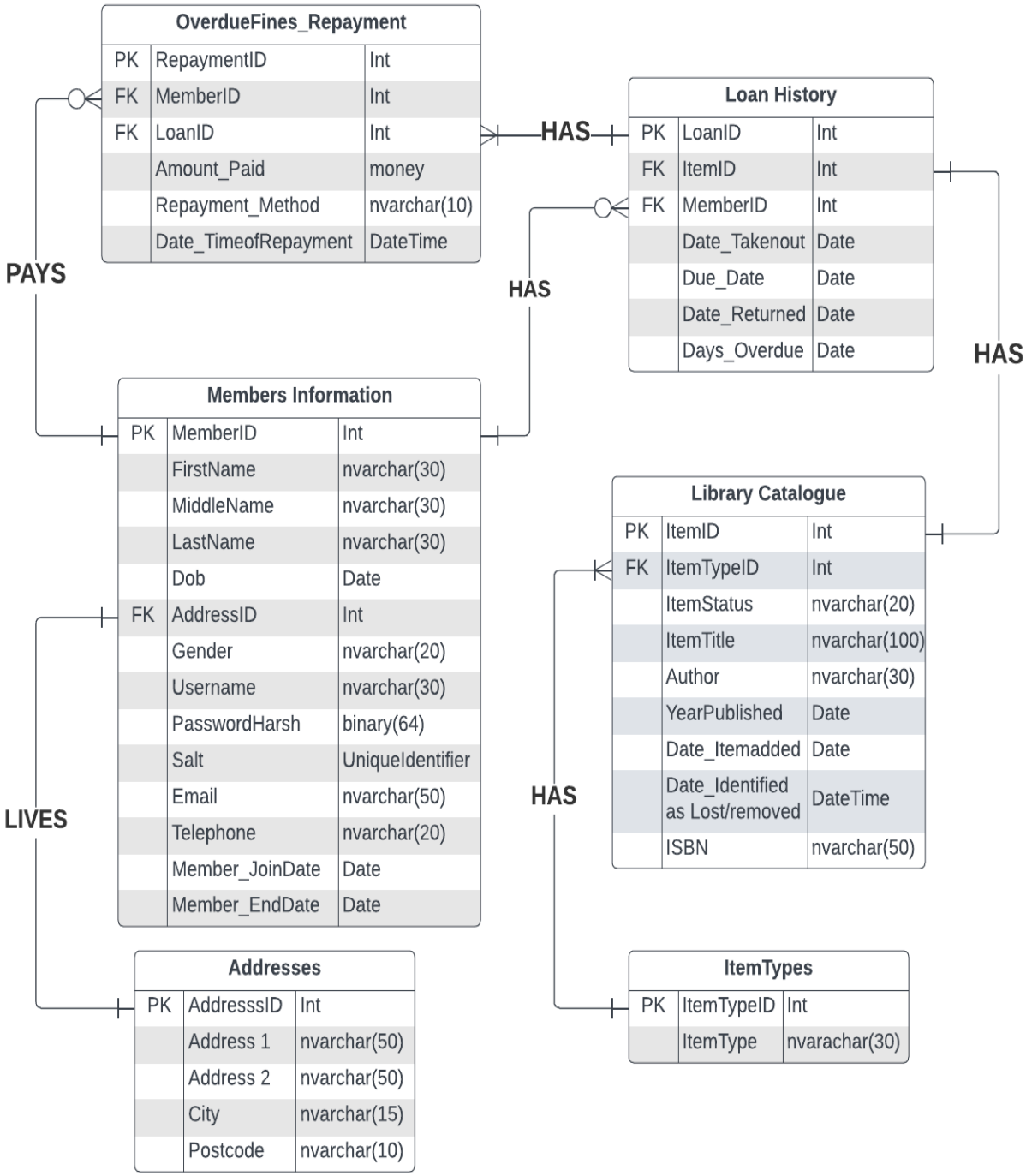
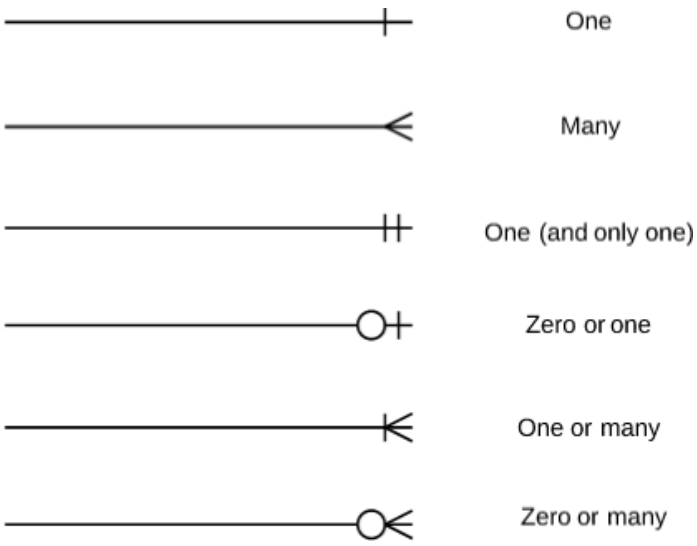


Fig 1.1 Entity Relationship Diagram (Crow's Foot Notation style)

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.

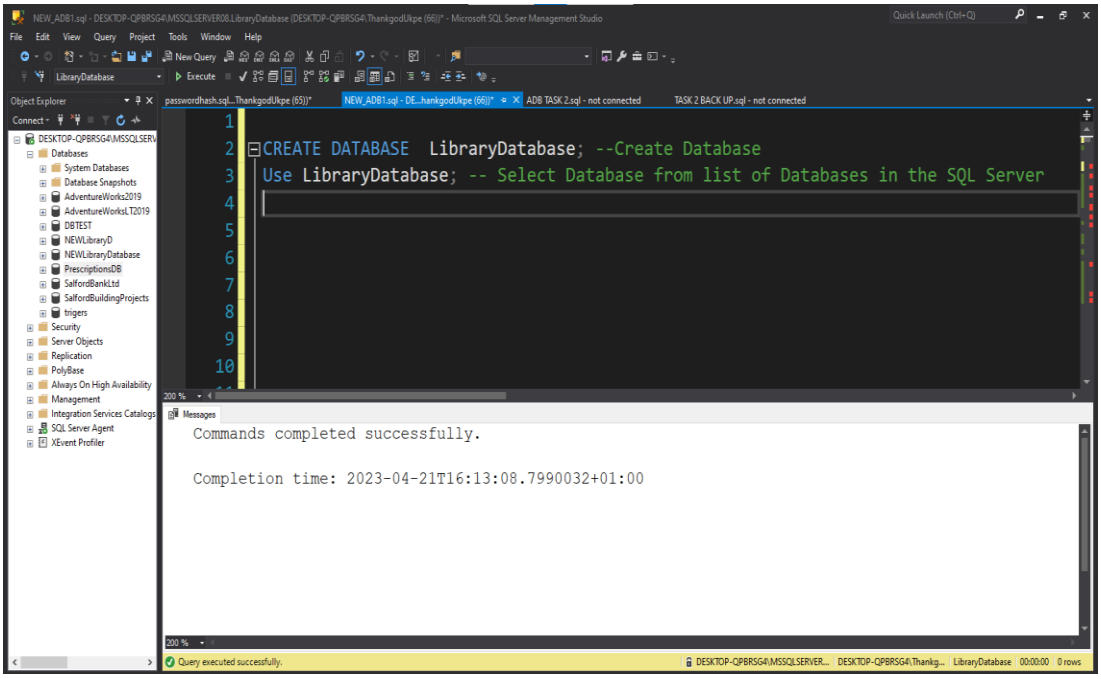
COLUMN NAMES	DATATYPE	REASON
MemberId, AddressId LoanId, ItemId, RepaymentId, ItemtypeId	Int	I chose the integer datatype for all the primary key and foreign key columns. These keys must be unique for each record in a table and the integer datatype provides this uniqueness because they can be incremented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

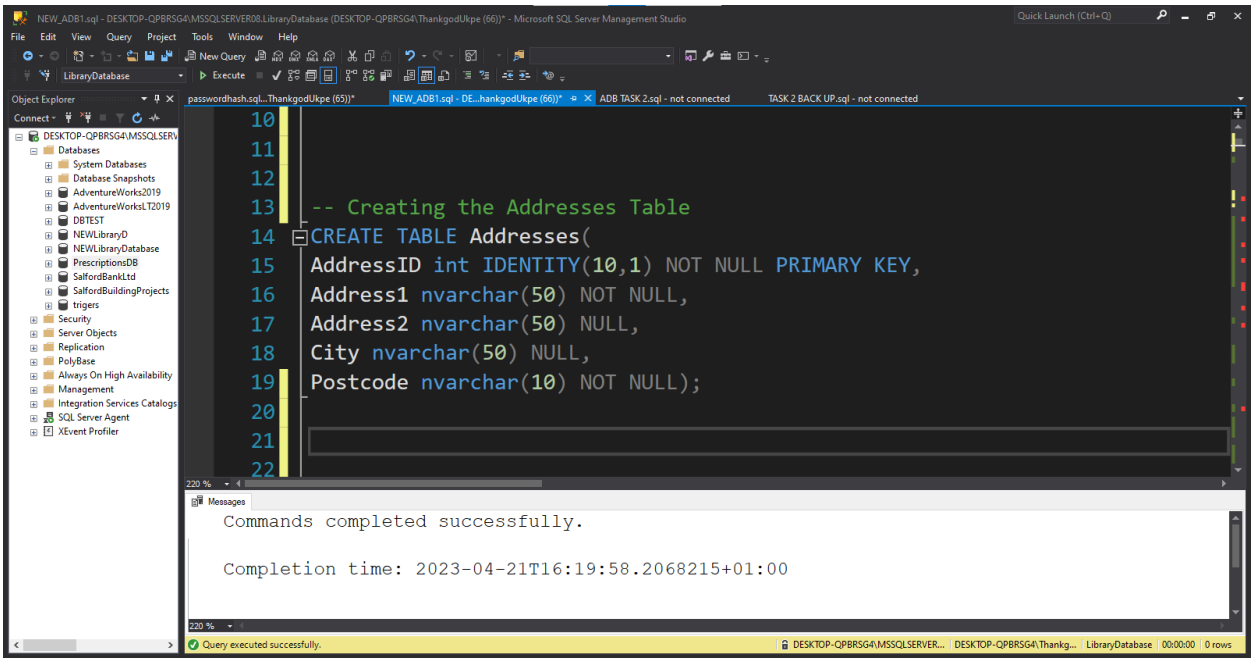
1. 4 Creating a Database

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.

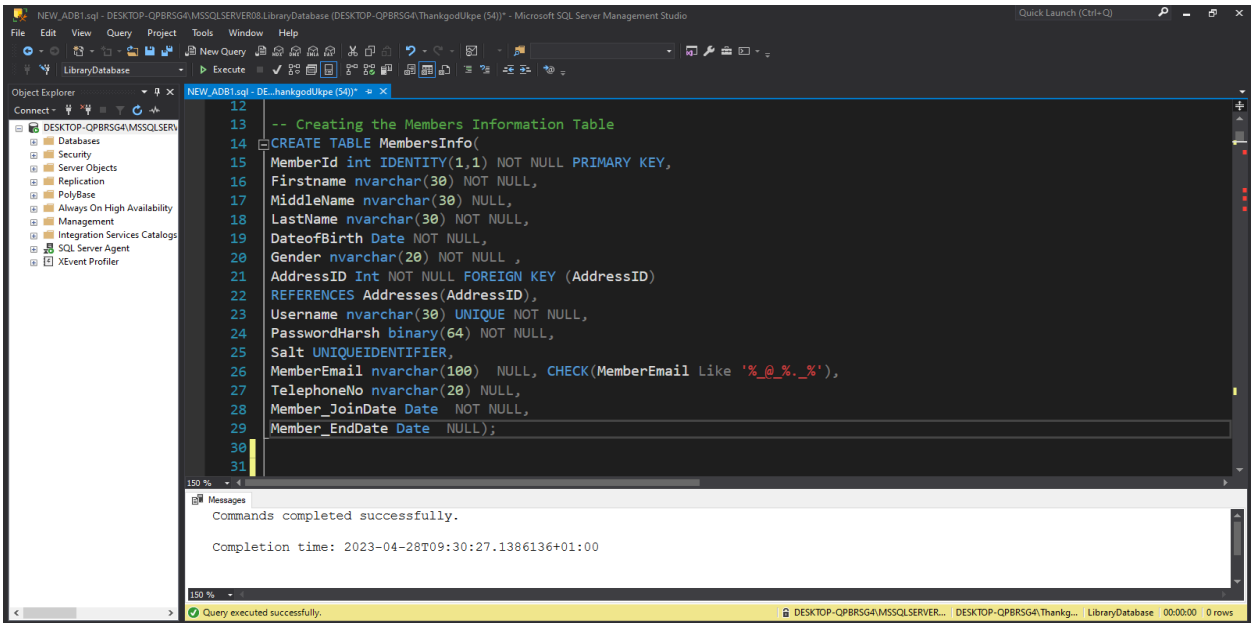


1. 5 Creating Tables

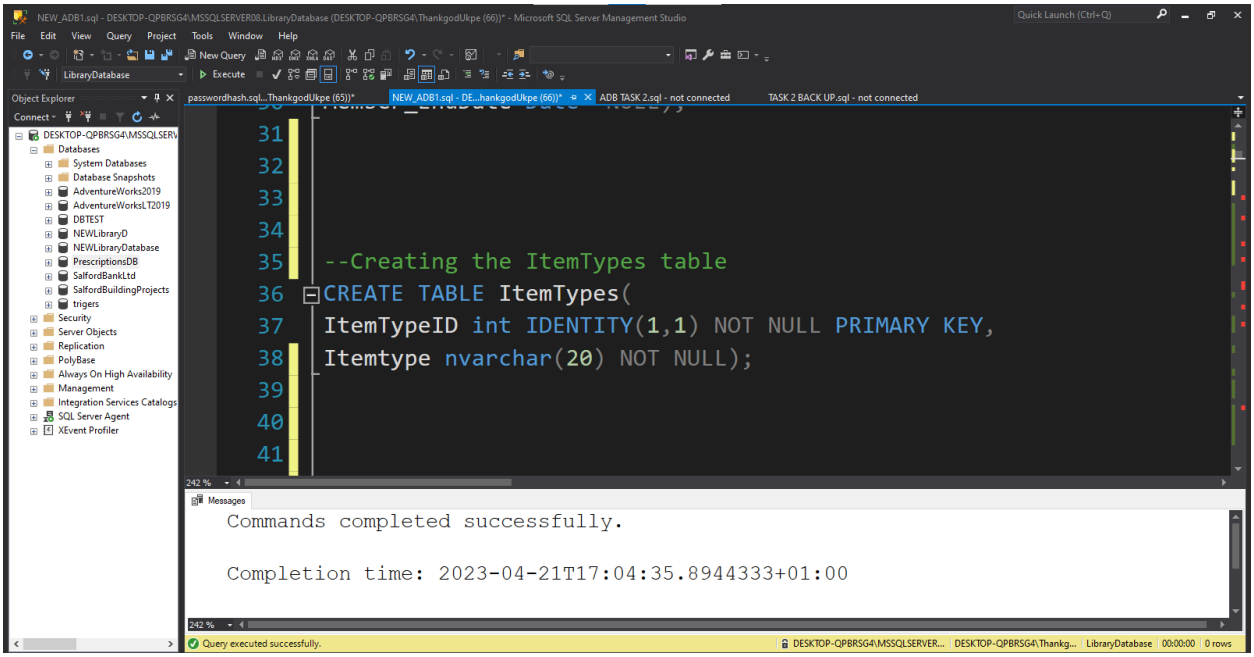
I. I first created the Address table because it has no inferential constraint in it.



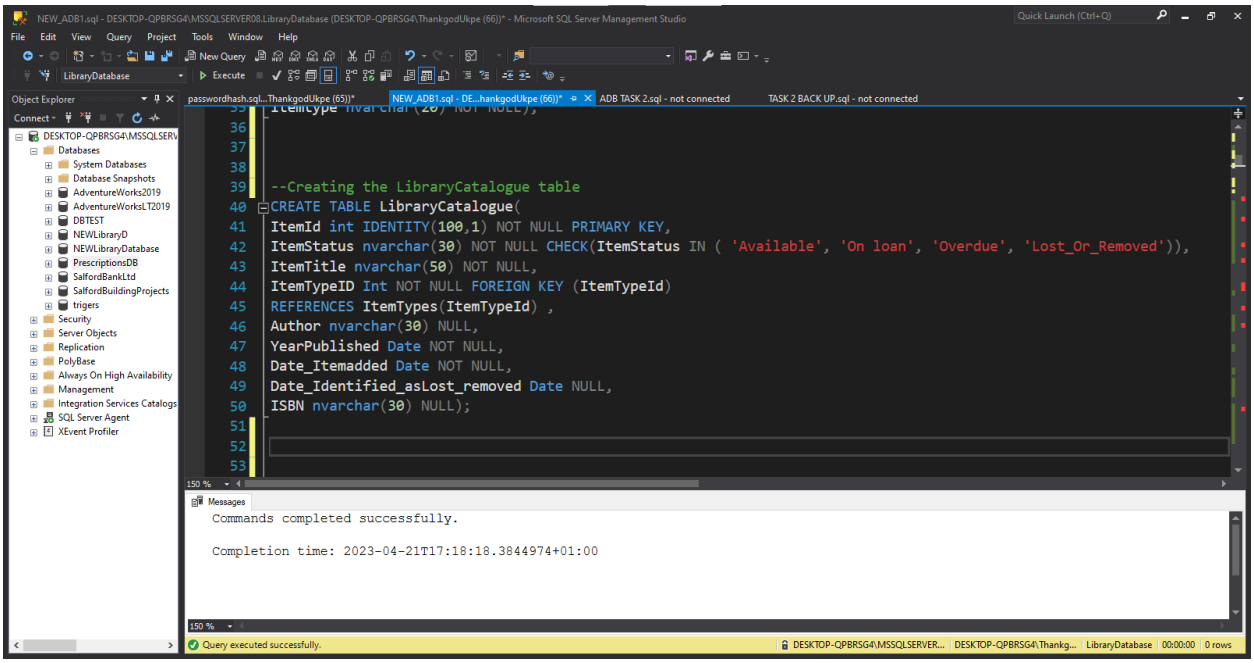
II. I then created the MembersInfo table.



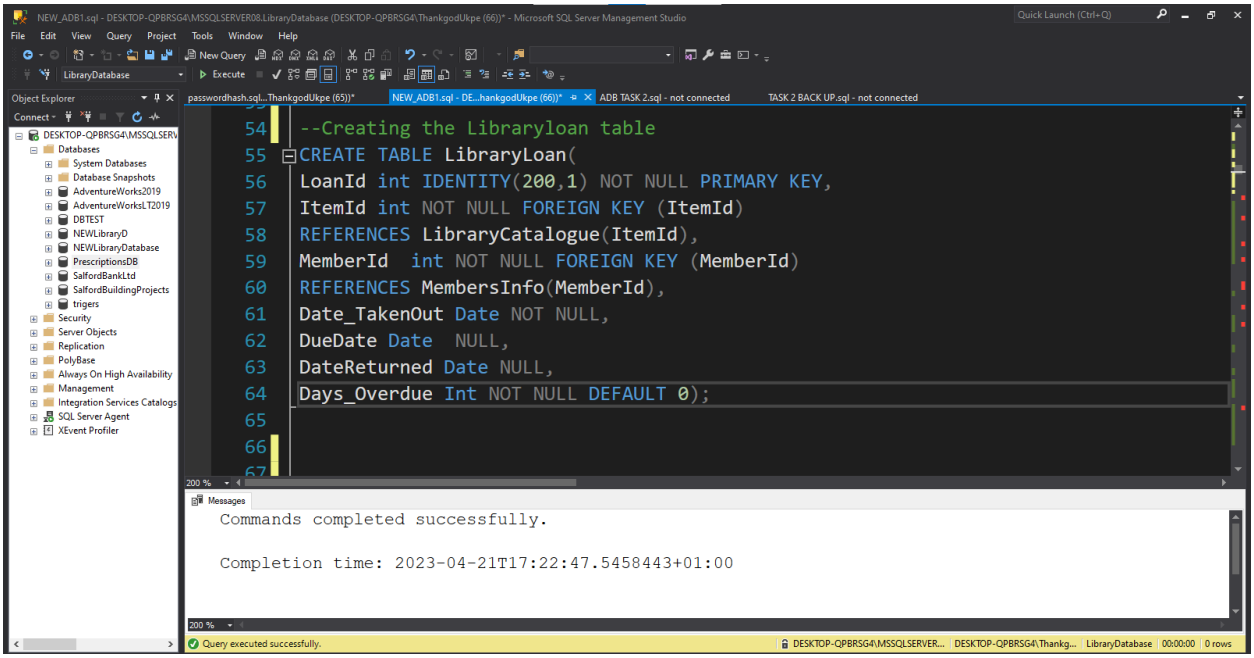
III. I created the ItemTypes table.



IV. I created the LibraryCatalogue table



V. I created the LibraryLoan table

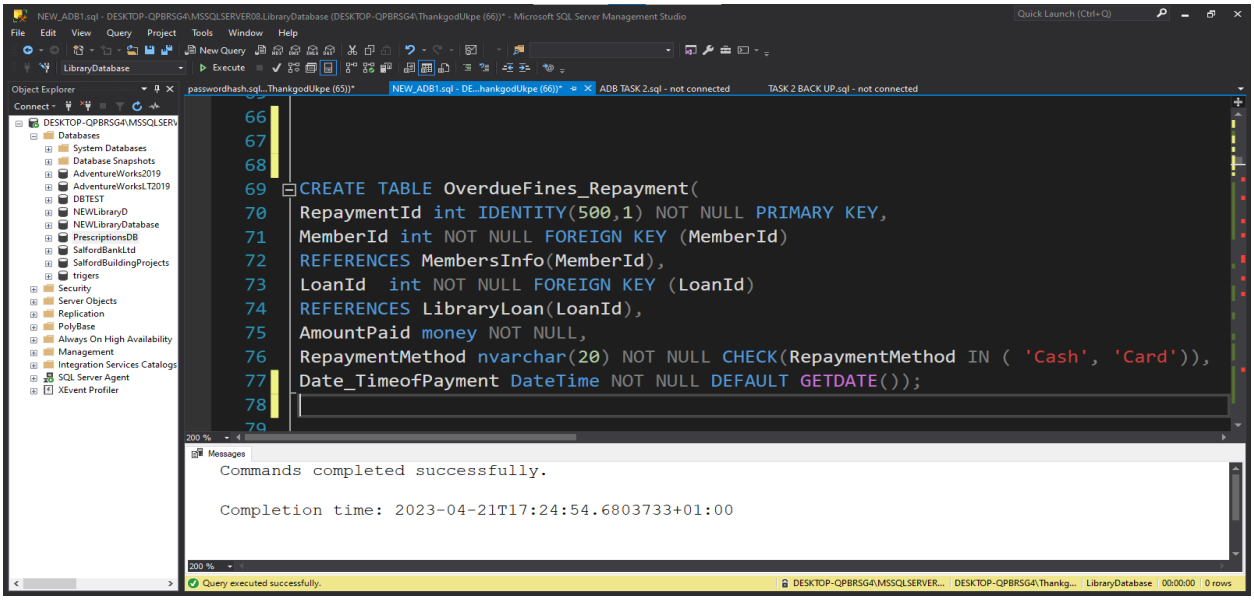


```
--Creating the Libraryloan table
CREATE TABLE LibraryLoan(
LoanId int IDENTITY(200,1) NOT NULL PRIMARY KEY,
ItemId int NOT NULL FOREIGN KEY (ItemId)
REFERENCES LibraryCatalogue(ItemId),
MemberId  int NOT NULL FOREIGN KEY (MemberId)
REFERENCES MembersInfo(MemberId),
Date_TakenOut Date NOT NULL,
DueDate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

```
Commands completed successfully.

Completion time: 2023-04-21T17:22:47.5458443+01:00
```

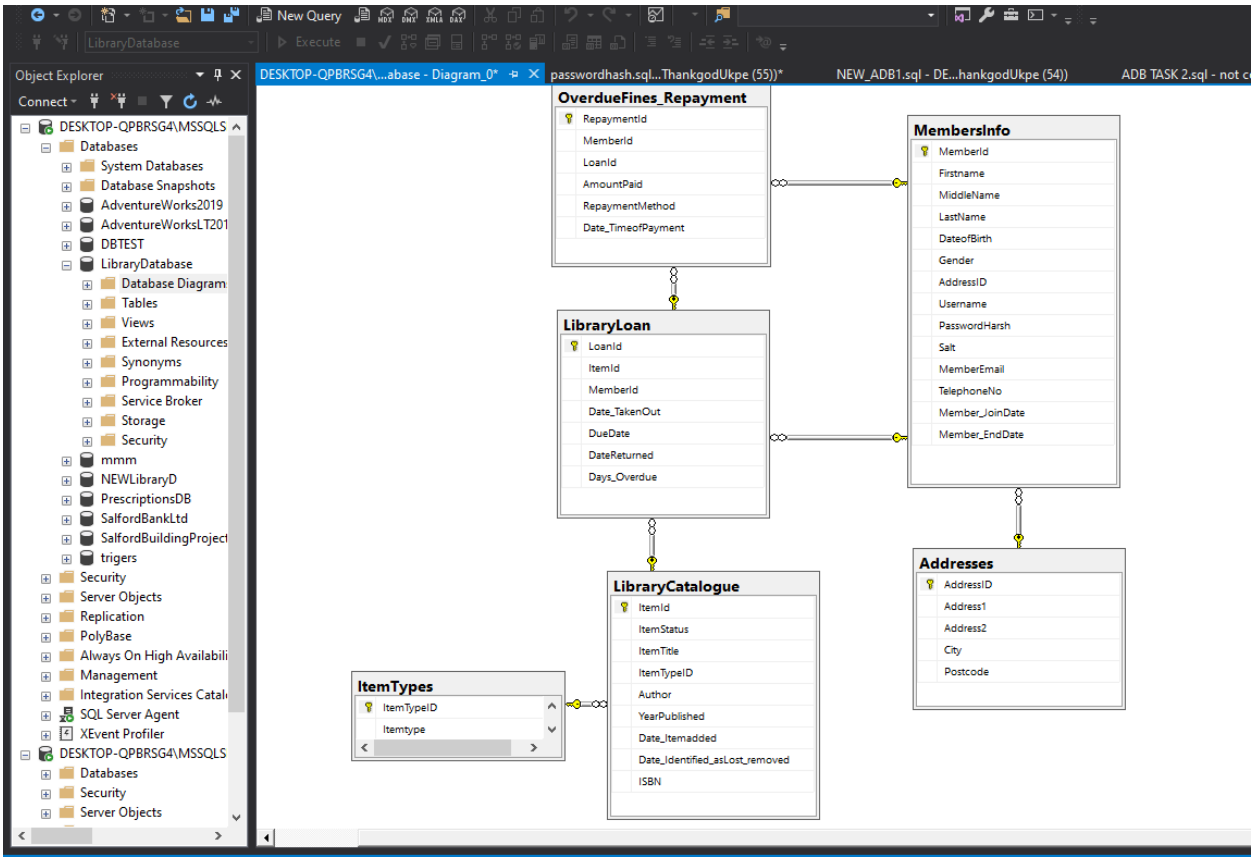
VI. I created the OverduFines_Repayment table



```
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)
REFERENCES LibraryLoan(LoanId),
AmountPaid money NOT NULL,
RepaymentMethod nvarchar(20) NOT NULL CHECK(RepaymentMethod IN ( 'Cash', 'Card')),
Date_TimeofPayment DateTime NOT NULL DEFAULT GETDATE());
```

```
Commands completed successfully.

Completion time: 2023-04-21T17:24:54.6803733+01:00
```



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.

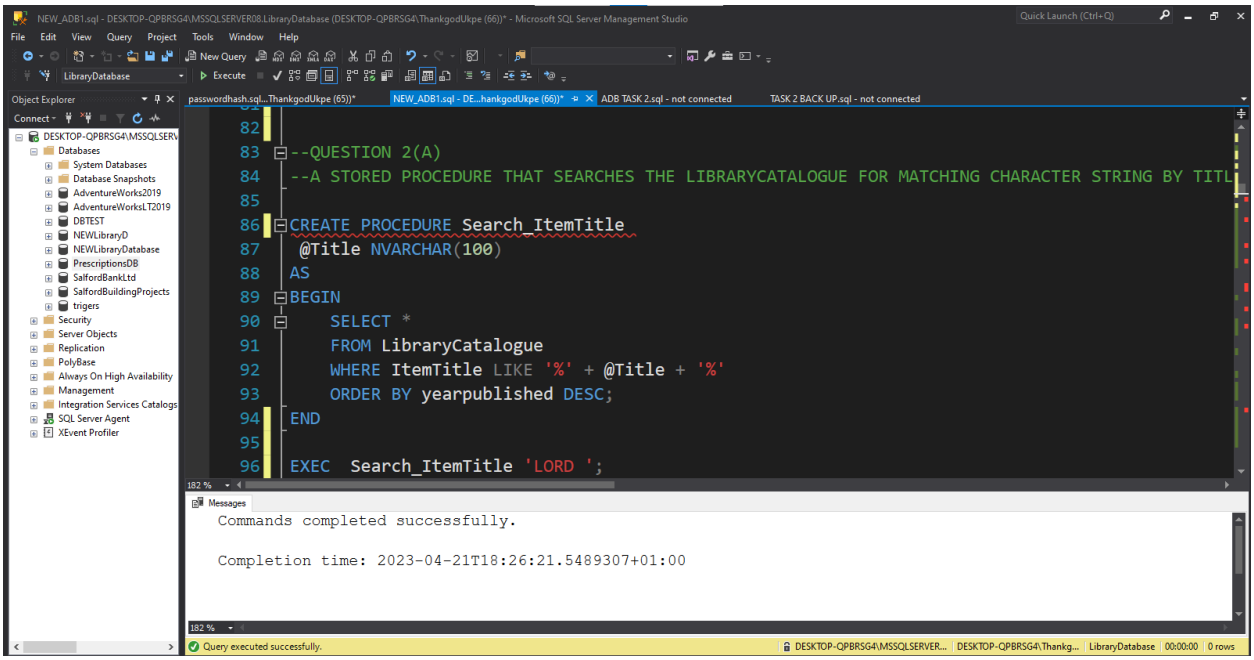


Fig 2a

(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.

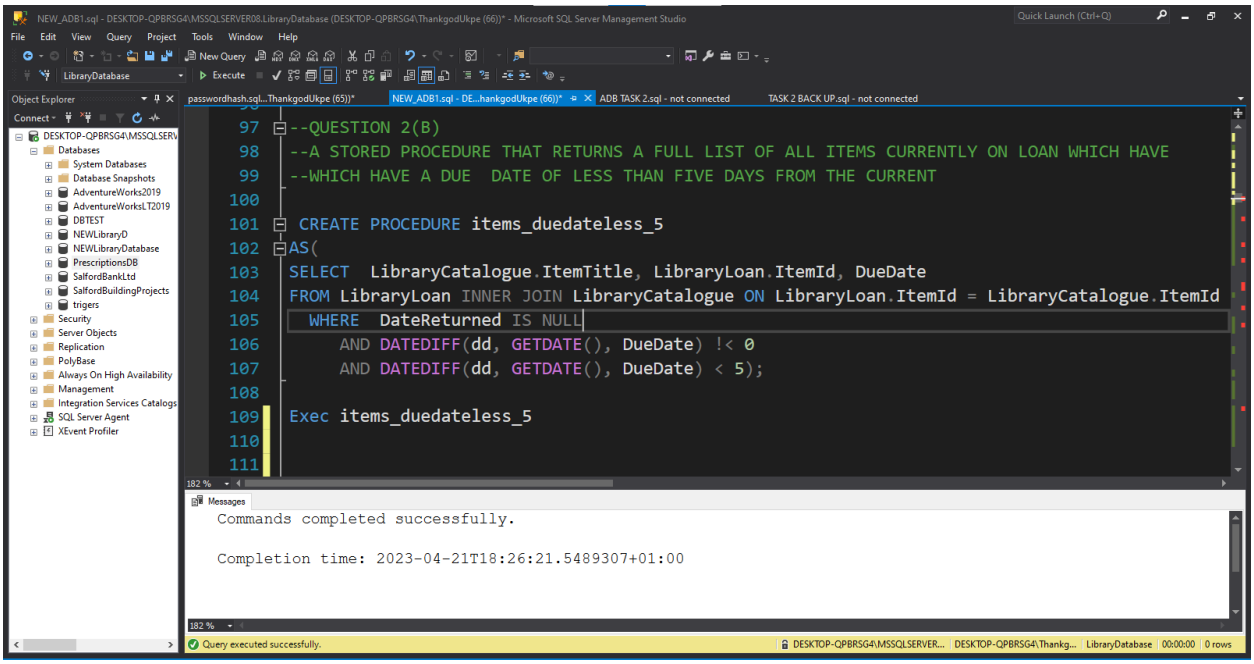


Fig 2b

(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.

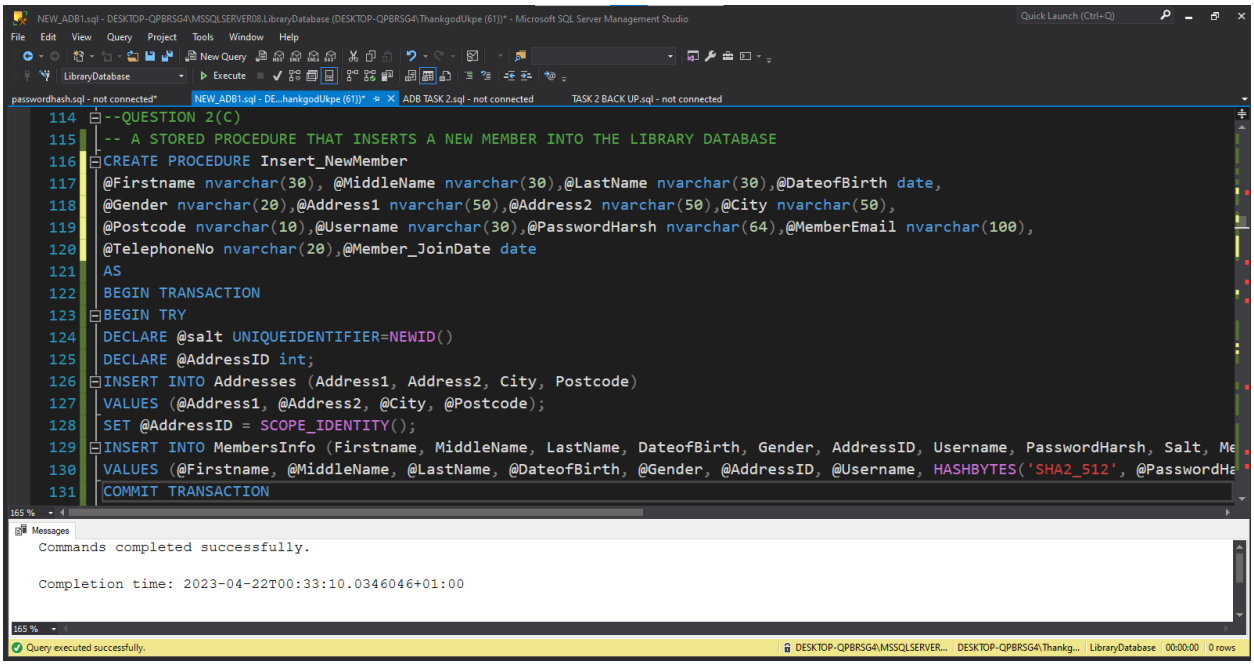


Fig 2c

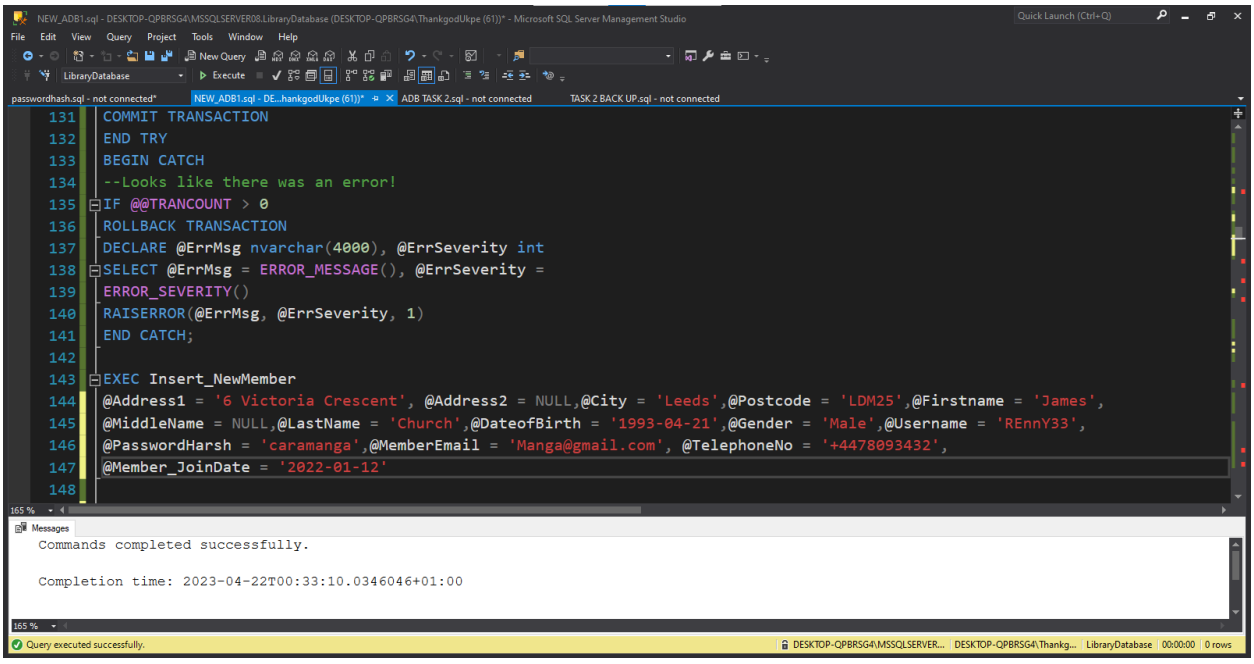


Fig 2c contd

(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.

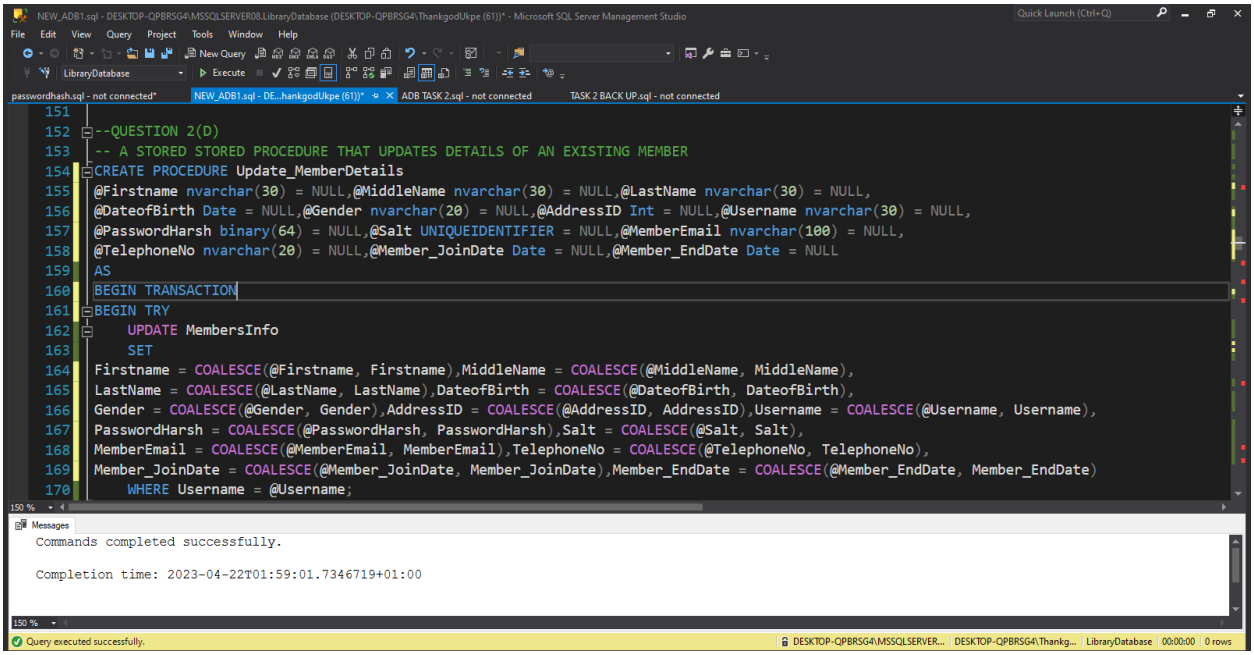


Fig 2d

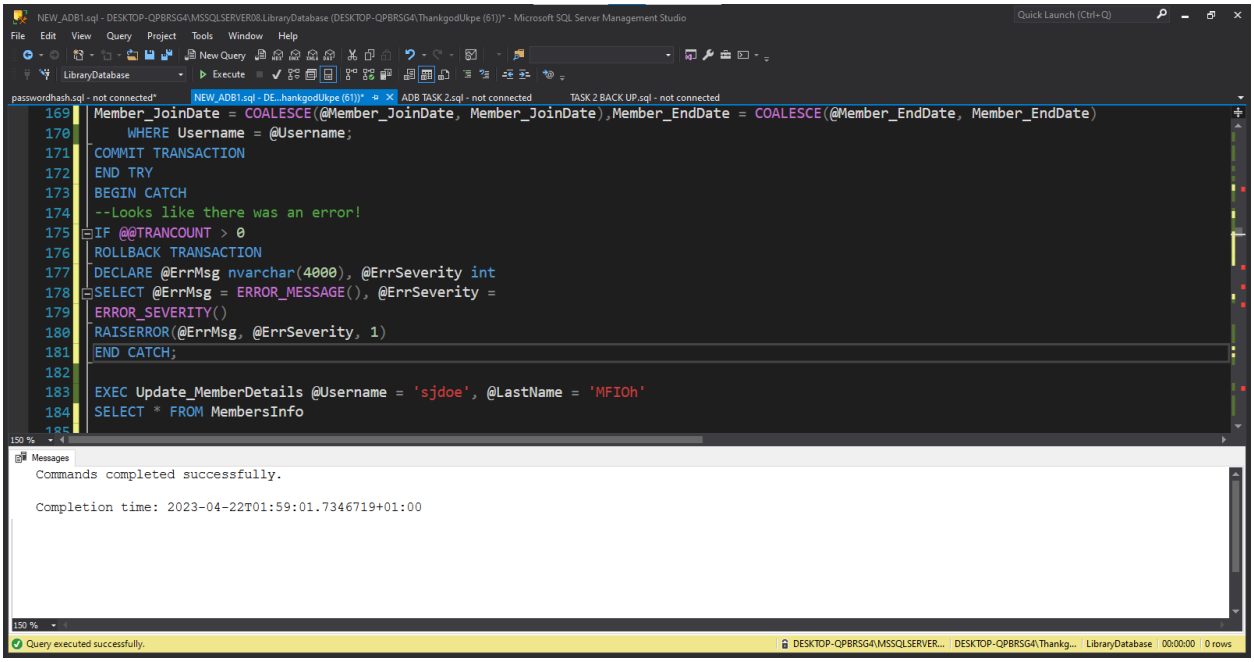


Fig 2d contd

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.

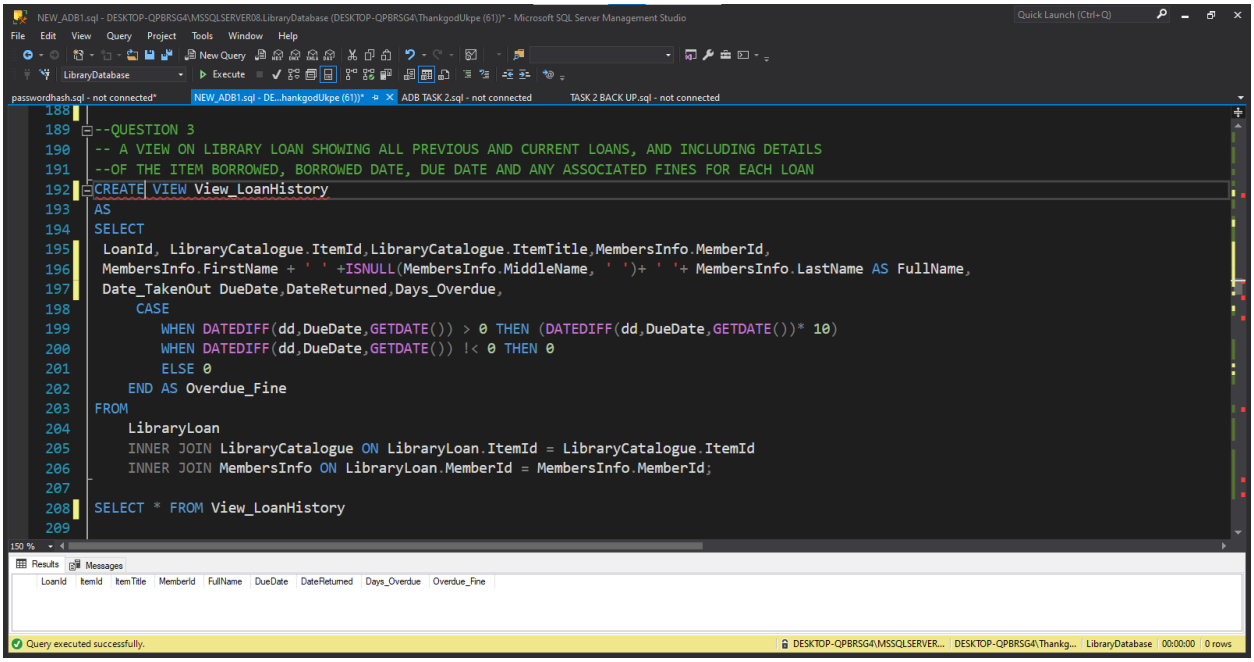


Fig 3

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.

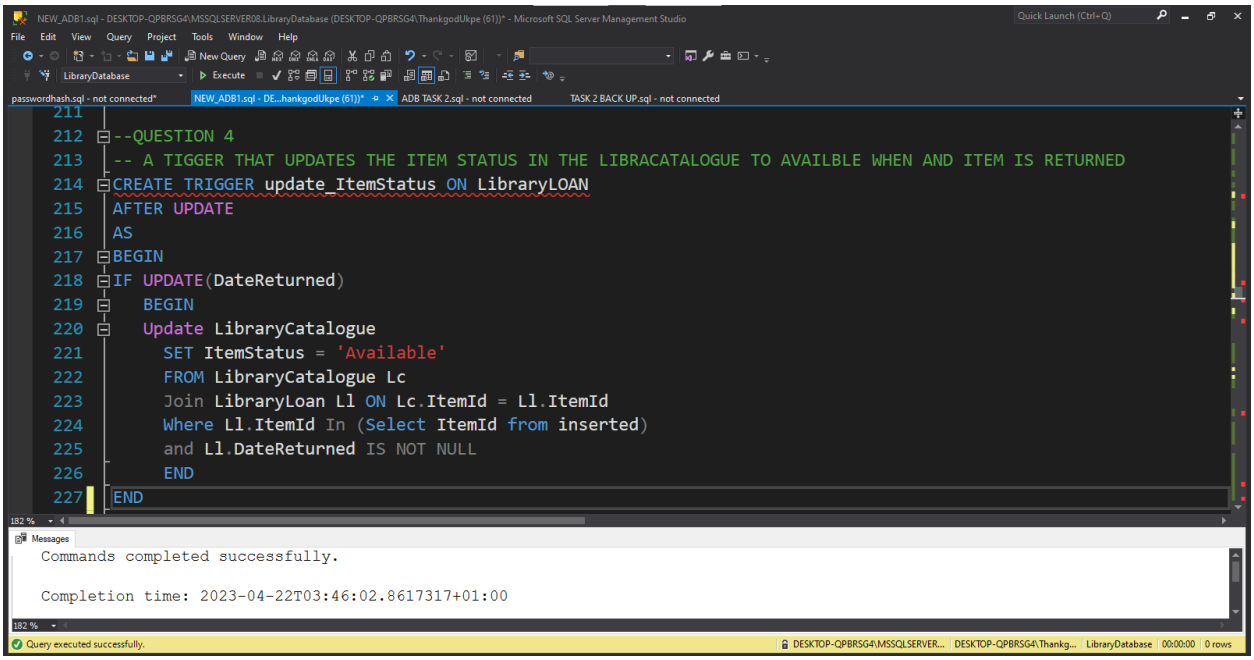


Fig 4

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_takenout 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.

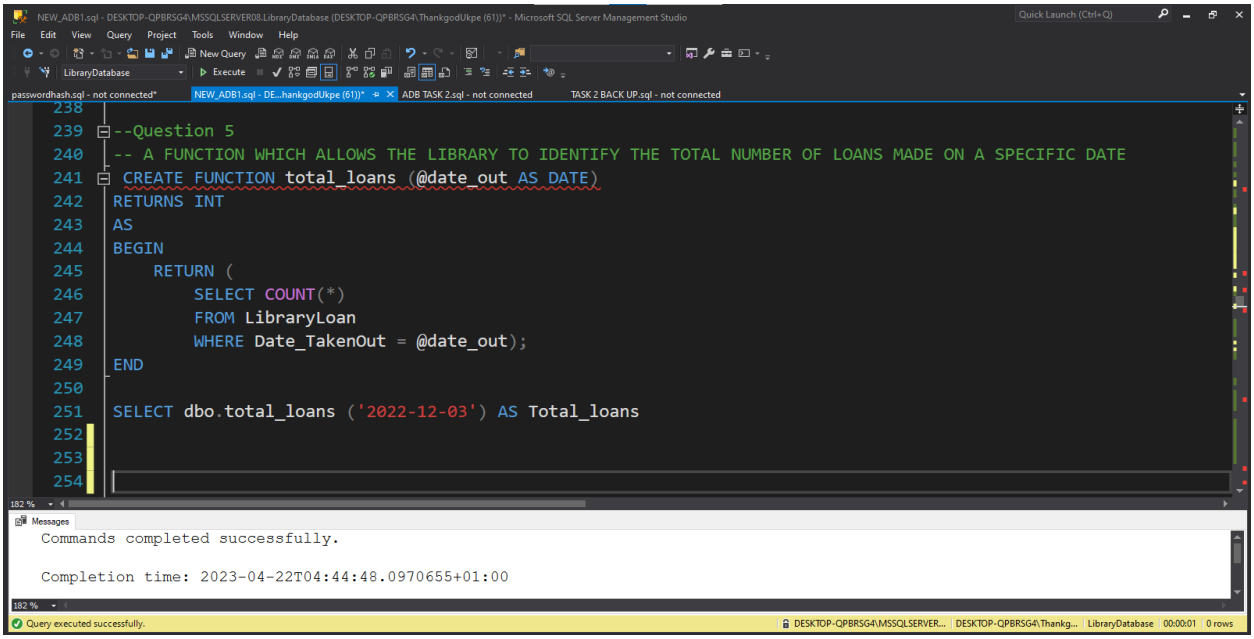
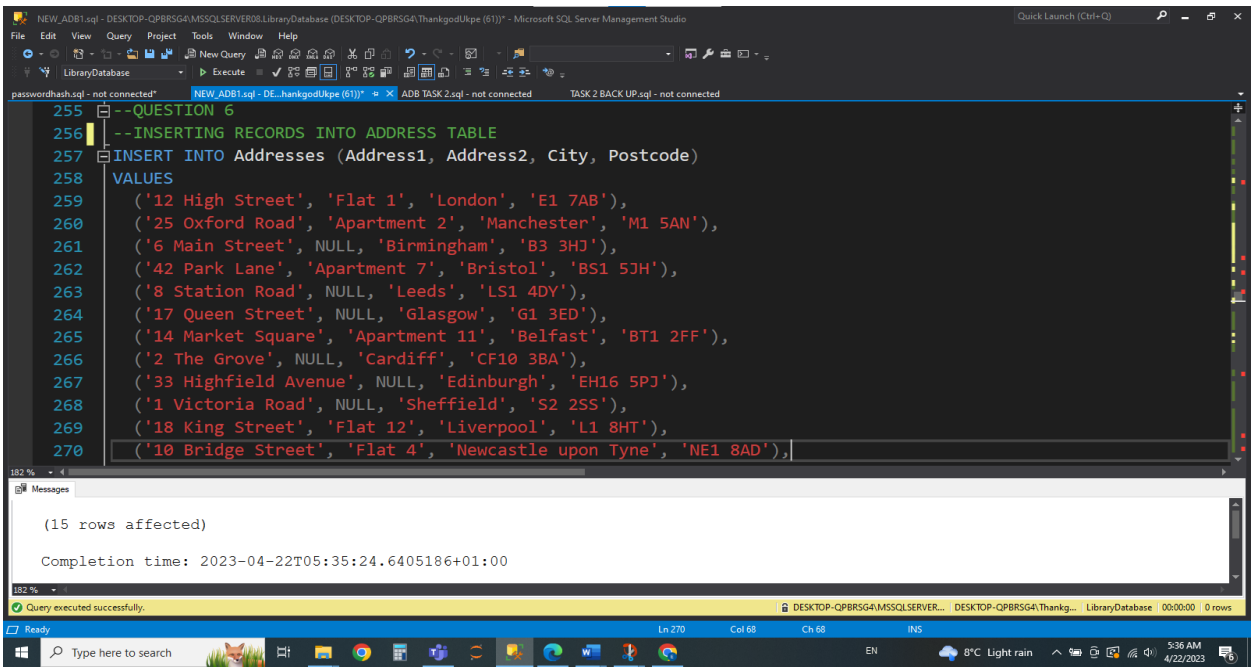


Fig 5

PART 6

6.1 Inserting Records into Tables



6.1.1 Fig 6.1.1 Inserting records into Address table

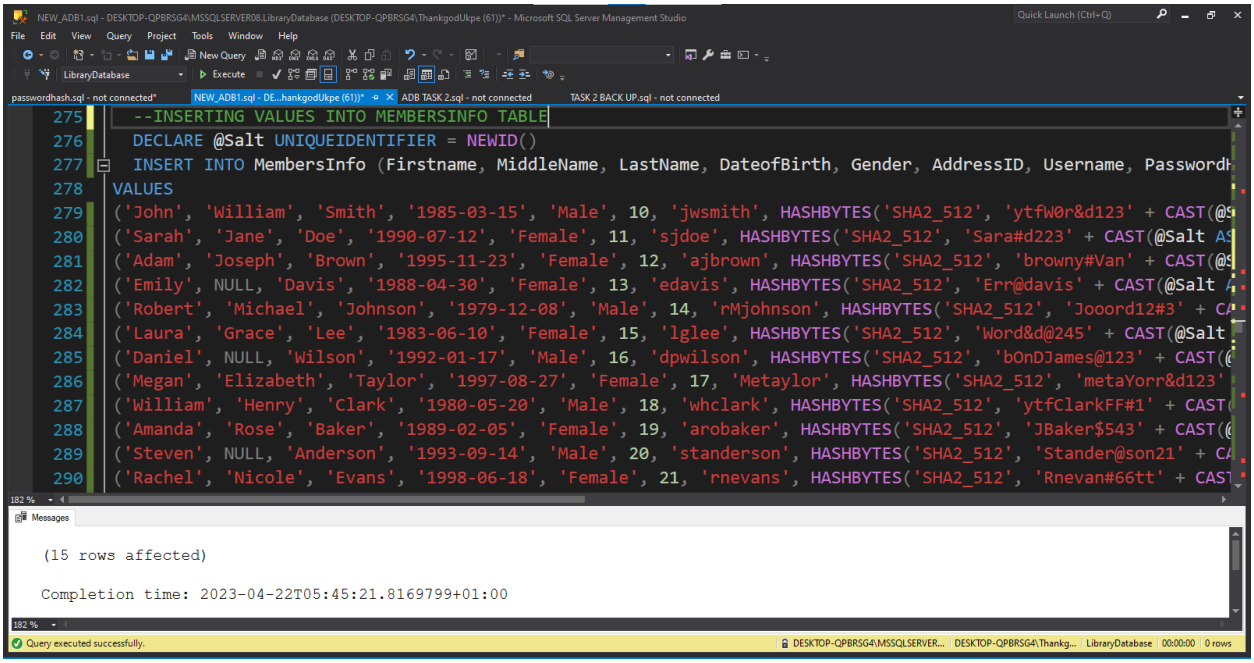


Fig 6.1.2 Inserting Records into MembersInfo table

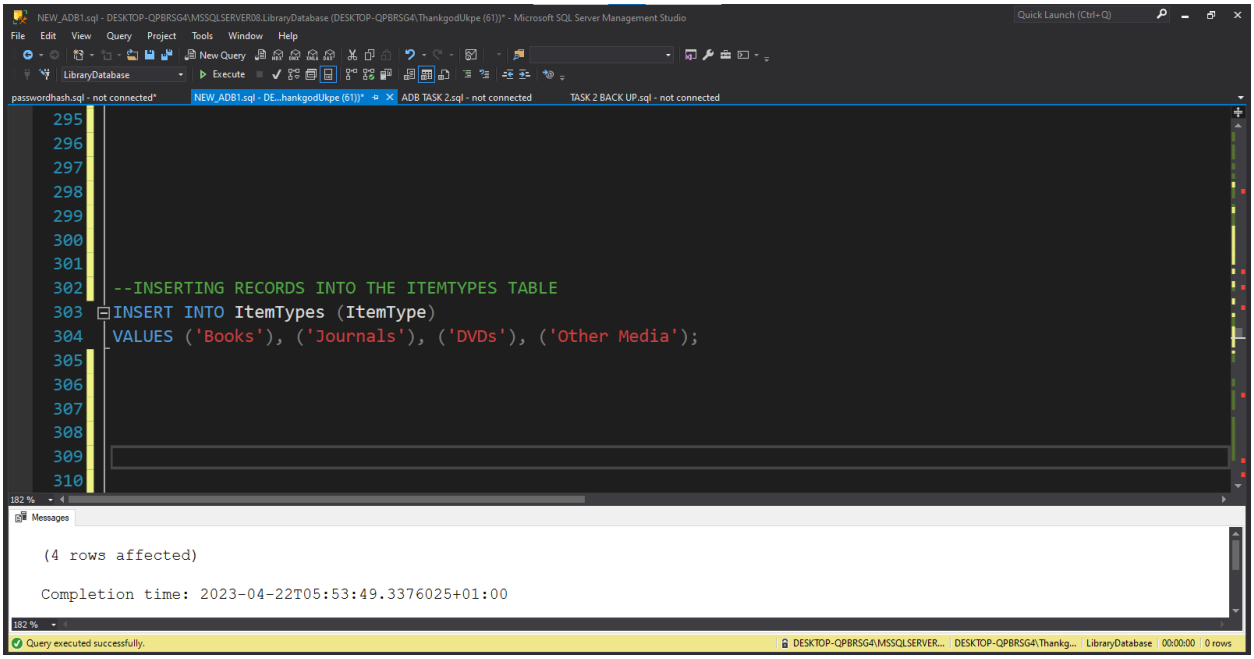


Fig 6.1.3 Inserting Records into ItemTypes table

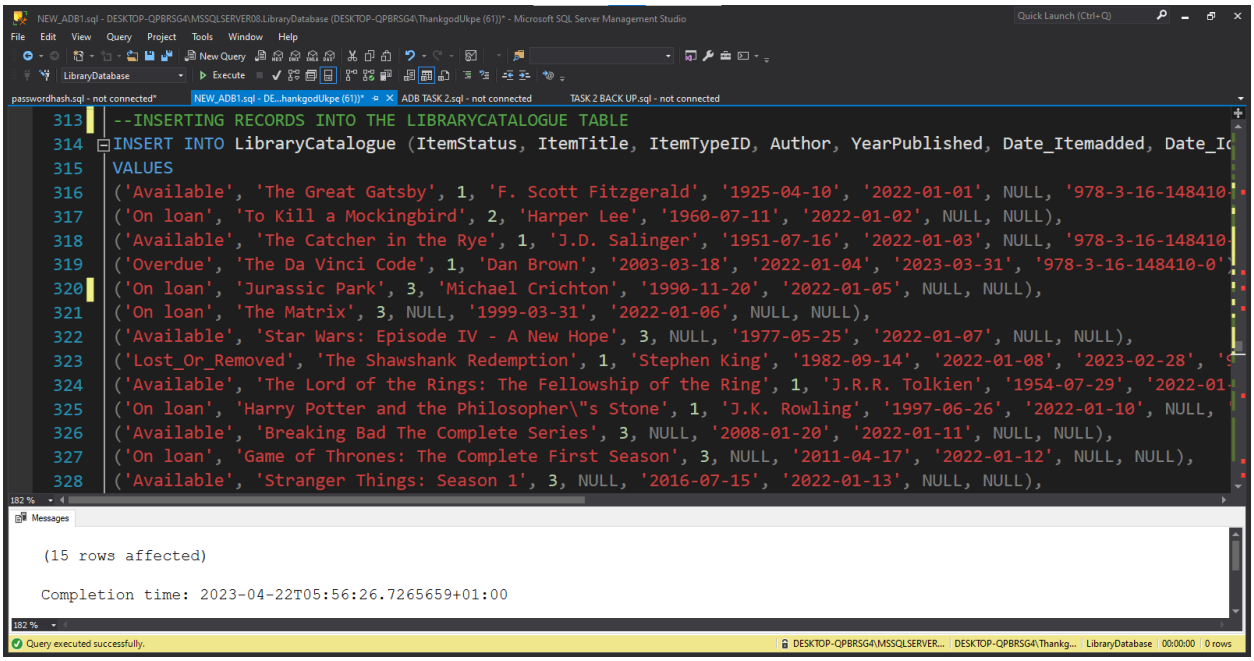


Fig 6.1.4 Inserting Records into LibraryCatalogue table

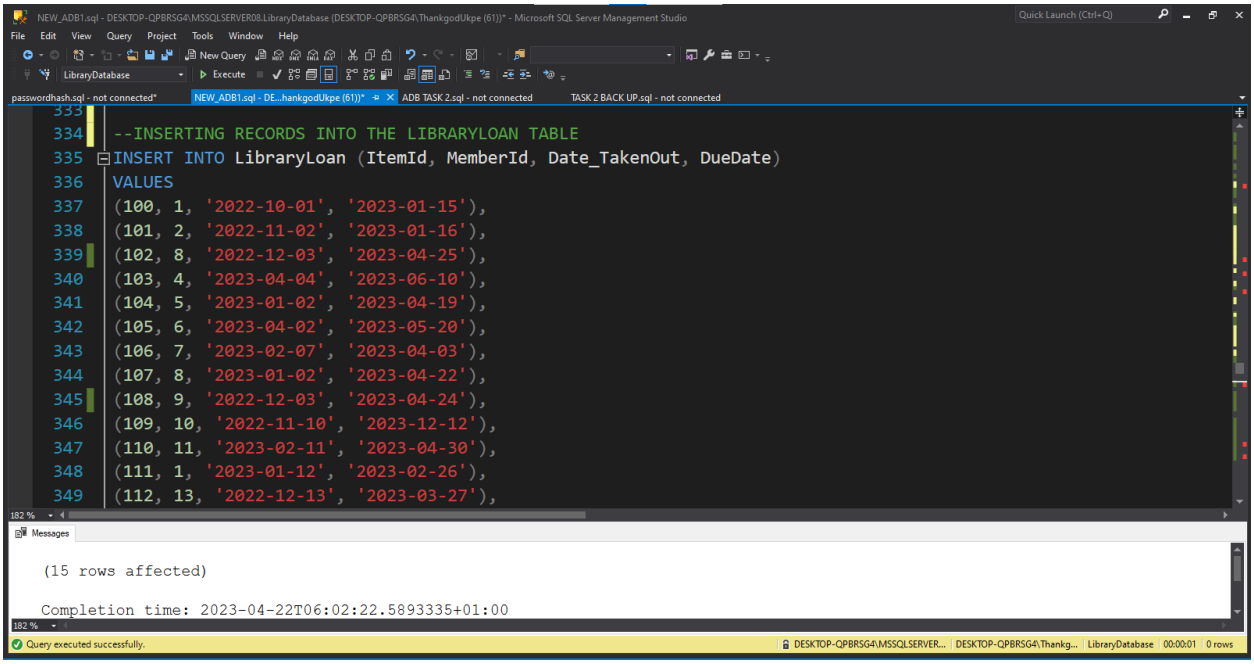


Fig 6.1.5 Inserting Records into LibraryLoan table

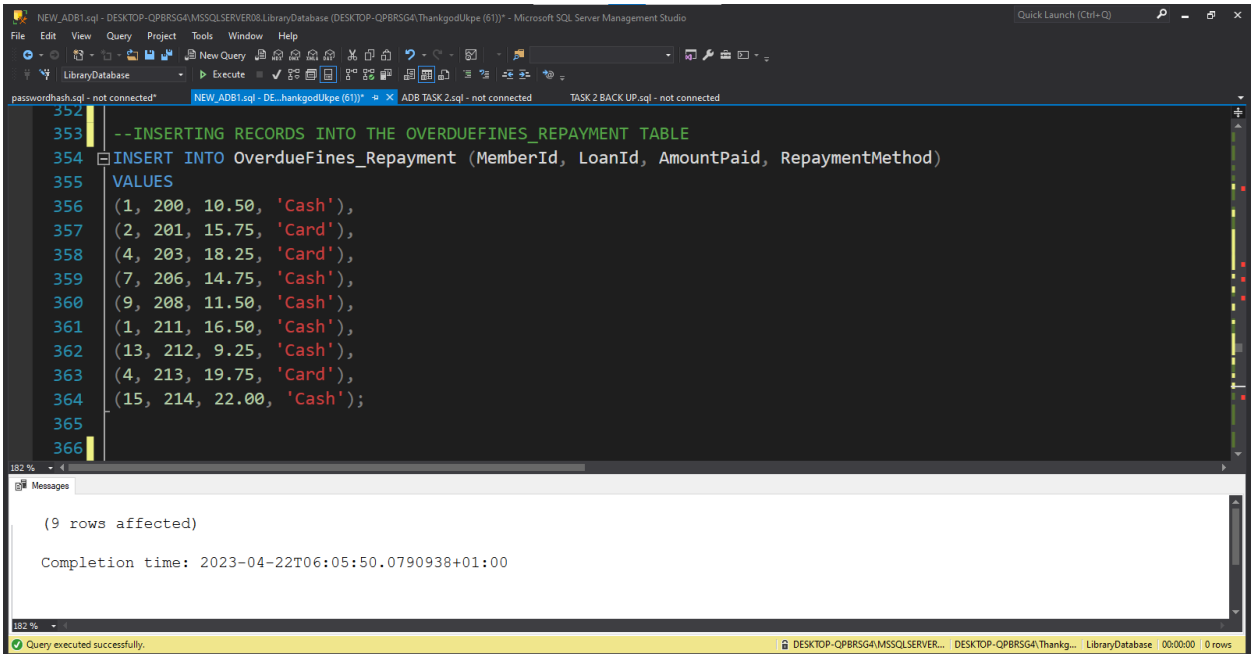


Fig 6.1.5 Inserting Records into OverdueFines_Repayment table

6.2 Displaying Inserted Records in the Tables Using Select Statement

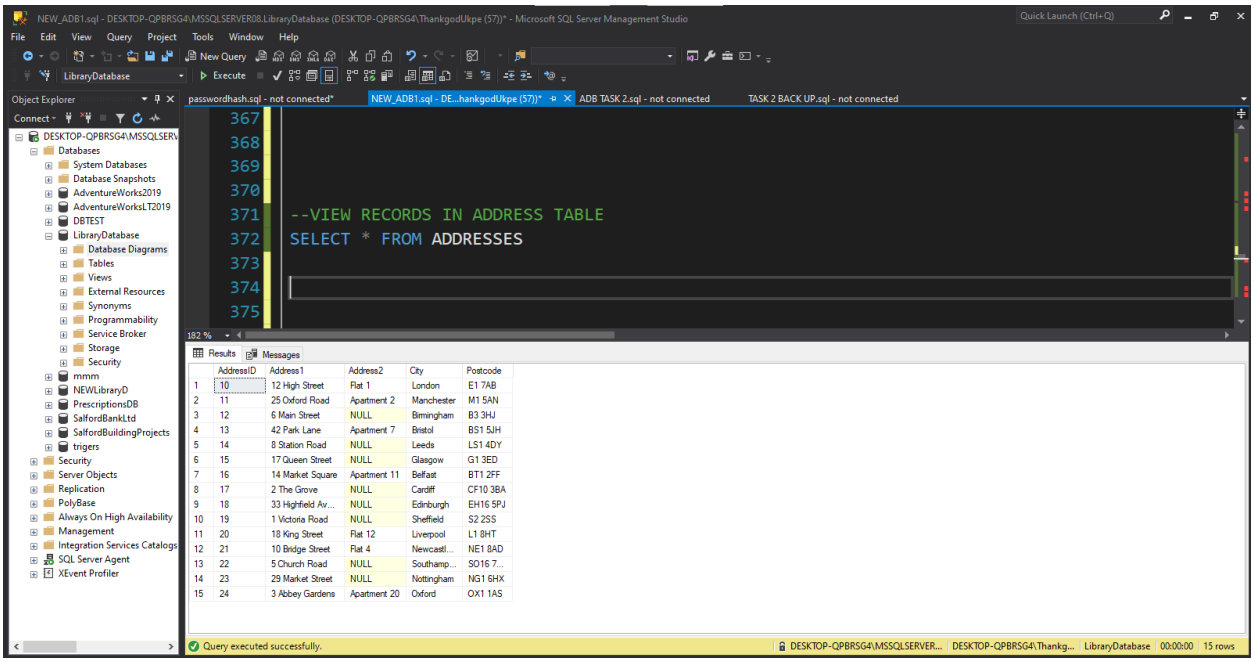


Fig 6.2.1 View records in the Addresses table

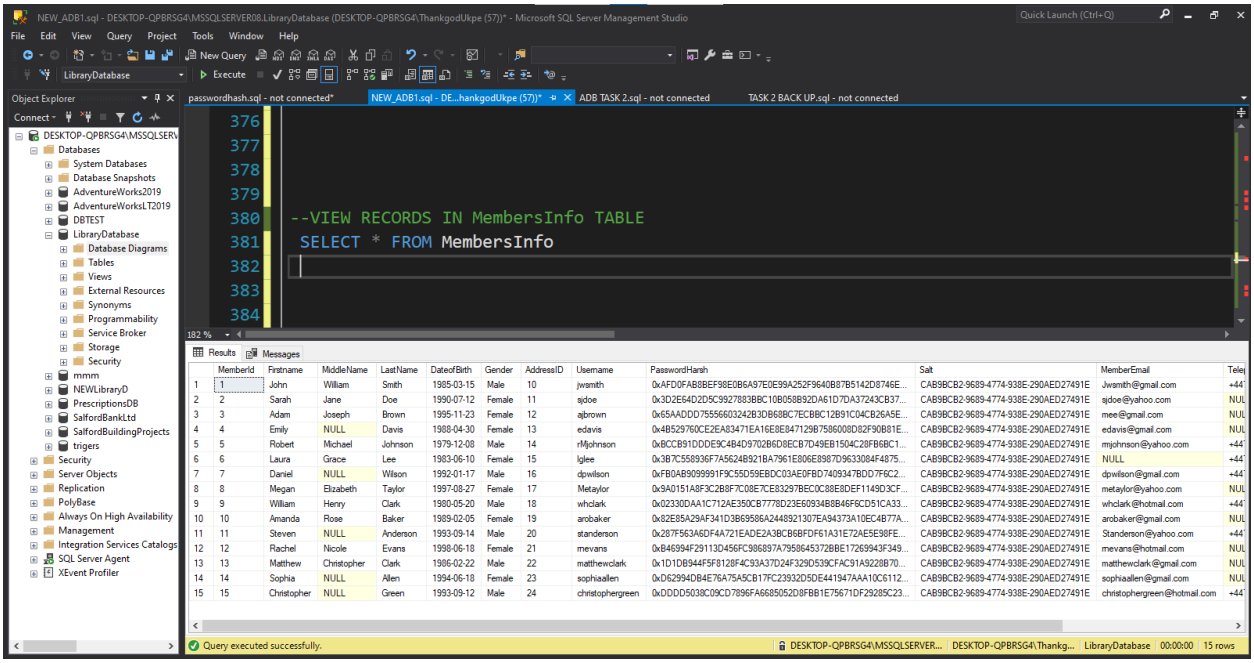


Fig 6.2.2 View records in the MembersInfo table

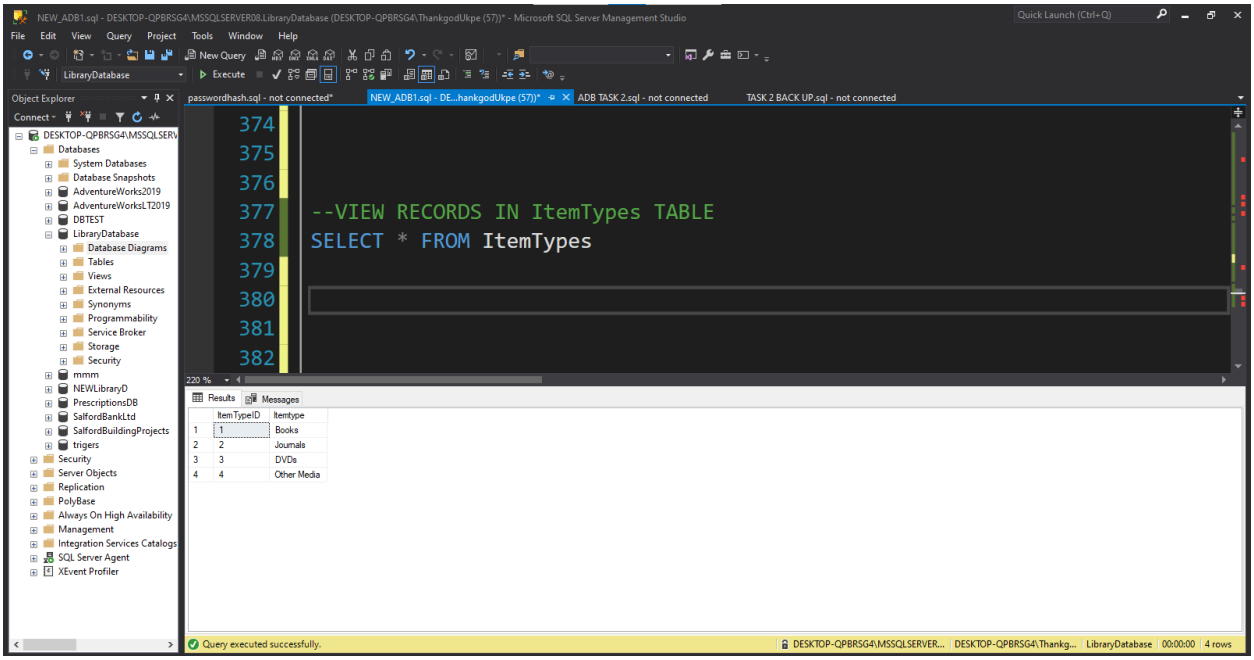
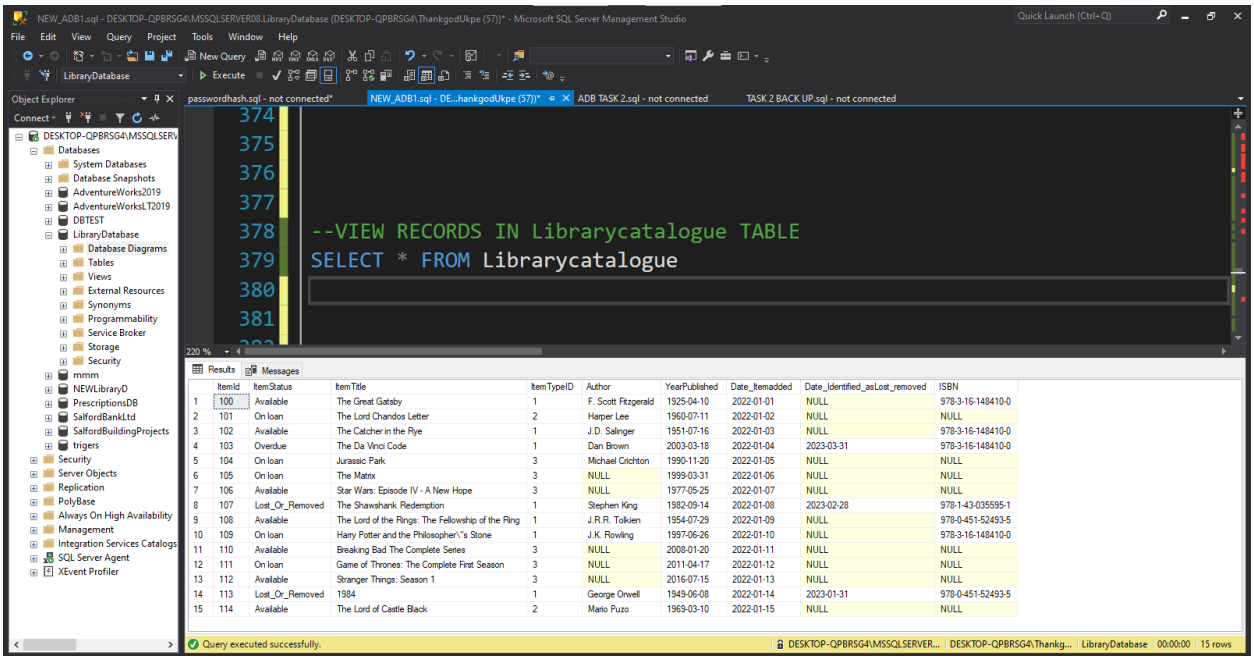
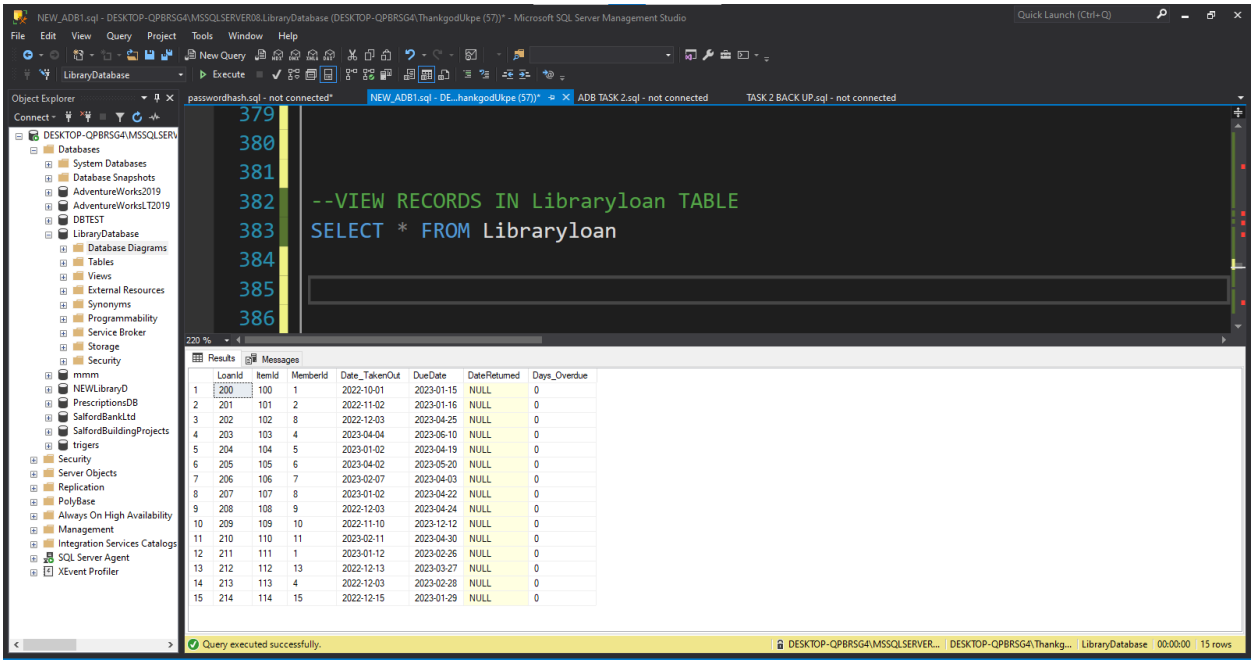


Fig 6.2.3 View records in the ItemTypes table



	ItemId	ItemStatus	ItemTitle	ItemTypeID	Author	YearPublished	Date_Itemadded	Date_Identified_asLost_removed	ISBN
1	100	Available	The Great Gatsby	1	F. Scott Fitzgerald	1925-04-10	2022-01-01	NULL	978-3-16-148410-0
2	101	On loan	The Lord Chandos Letter	2	Harper Lee	1960-07-11	2022-01-02	NULL	NULL
3	102	Available	The Catcher in the Rye	1	J.D. Salinger	1951-07-16	2022-01-03	NULL	978-3-16-148410-0
4	103	Overdue	The Da Vinci Code	1	Dan Brown	2003-03-18	2022-01-04	2023-03-31	978-3-16-148410-0
5	104	On loan	Jurassic Park	3	Michael Crichton	1990-11-20	2022-01-05	NULL	NULL
6	105	On loan	The Matrix	3	NULL	1999-03-31	2022-01-06	NULL	NULL
7	106	Available	Star Wars: Episode IV - A New Hope	3	NULL	1977-05-25	2022-01-07	NULL	NULL
8	107	Lost_Or_Removed	The Shawshank Redemption	1	Stephen King	1982-09-14	2022-01-08	2023-02-28	978-1-43-035595-1
9	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
10	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
11	110	Available	Breaking Bad The Complete Series	3	NULL	2008-01-20	2022-01-11	NULL	NULL
12	111	On loan	Game of Thrones: The Complete First Season	3	NULL	2011-04-17	2022-01-12	NULL	NULL
13	112	Available	Stranger Things: Season 1	3	NULL	2016-07-15	2022-01-13	NULL	NULL
14	113	Lost_Or_Removed	1984	1	George Orwell	1949-06-08	2022-01-14	2023-01-31	978-0-451-52493-5
15	114	Available	The Lord of Castle Black	2	Mario Puzo	1969-03-10	2022-01-15	NULL	NULL

Fig 6.2.4 View records in the LibraryCatalogue table



	LoanId	ItemId	MemberId	Date_TakenOut	DueDate	DateRetumed	Days_Overdue
1	200	100	1	2022-10-01	2023-01-15	NULL	0
2	201	101	2	2022-11-02	2023-01-16	NULL	0
3	202	102	8	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	NULL	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
9	208	108	9	2022-12-03	2023-04-24	NULL	0
10	209	109	10	2022-11-10	2023-12-12	NULL	0
11	210	110	11	2023-02-11	2023-04-30	NULL	0
12	211	111	1	2023-01-12	2023-02-26	NULL	0
13	212	112	13	2022-12-13	2023-03-27	NULL	0
14	213	113	4	2022-12-03	2023-02-28	NULL	0
15	214	114	15	2022-12-15	2023-01-29	NULL	0

Fig 6.2.5 View records in the LibraryLoan table

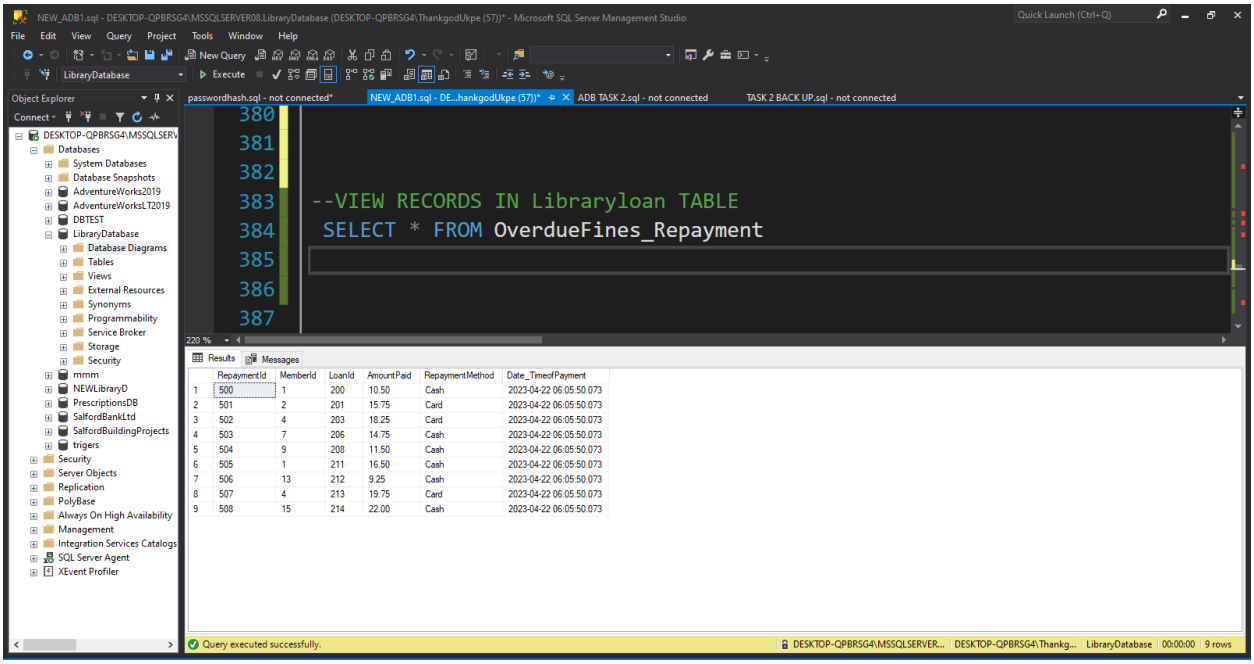


Fig 6.2.6 View records in the OverdueFines_Repayment table

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.

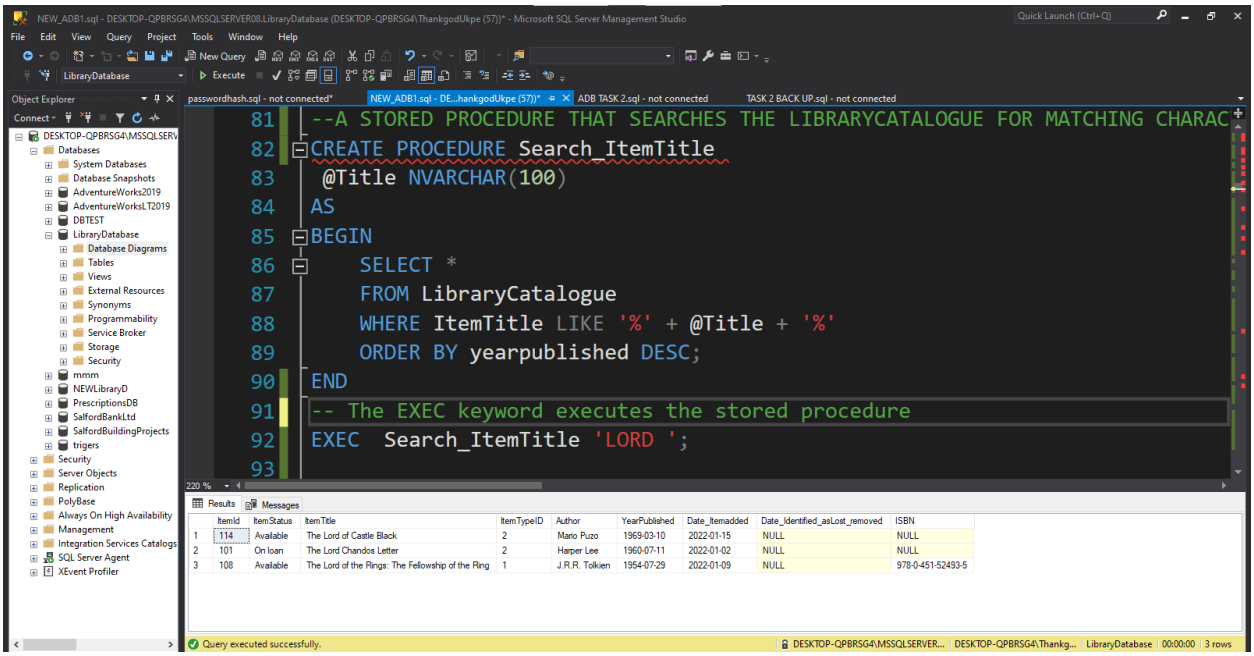


Fig 6.3.1 Executing the Stored procedure in Part 2a above.

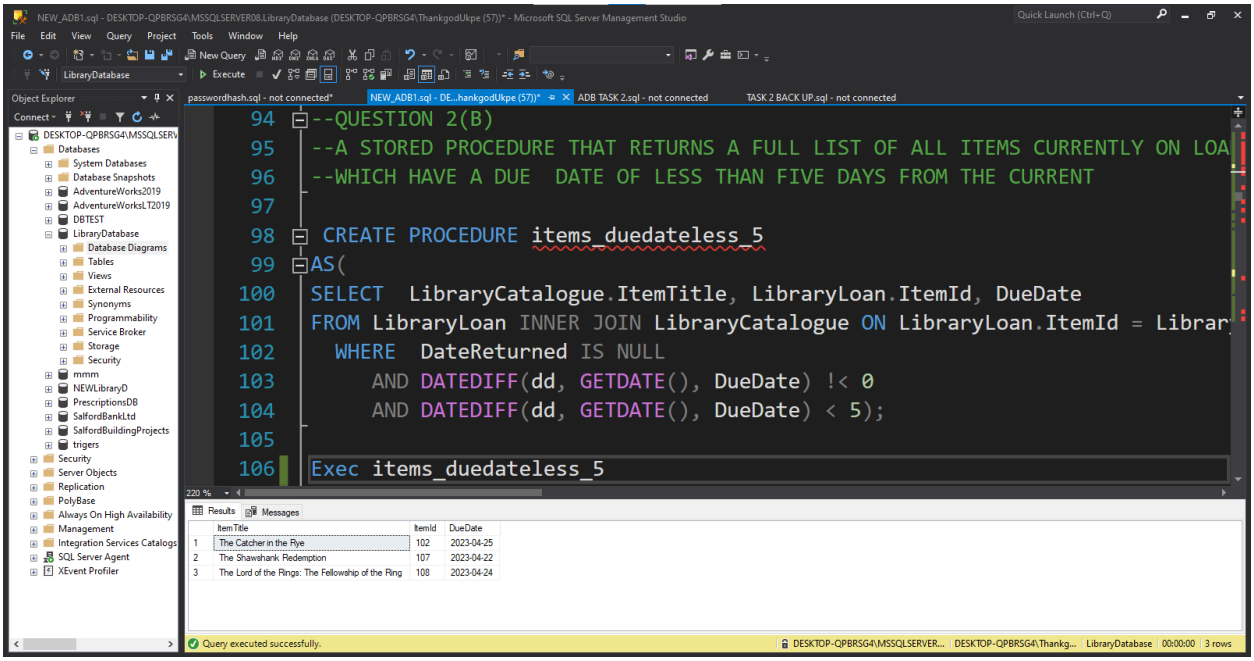


Fig 6.3.2 Executing the Stored procedure in Part 2b above.

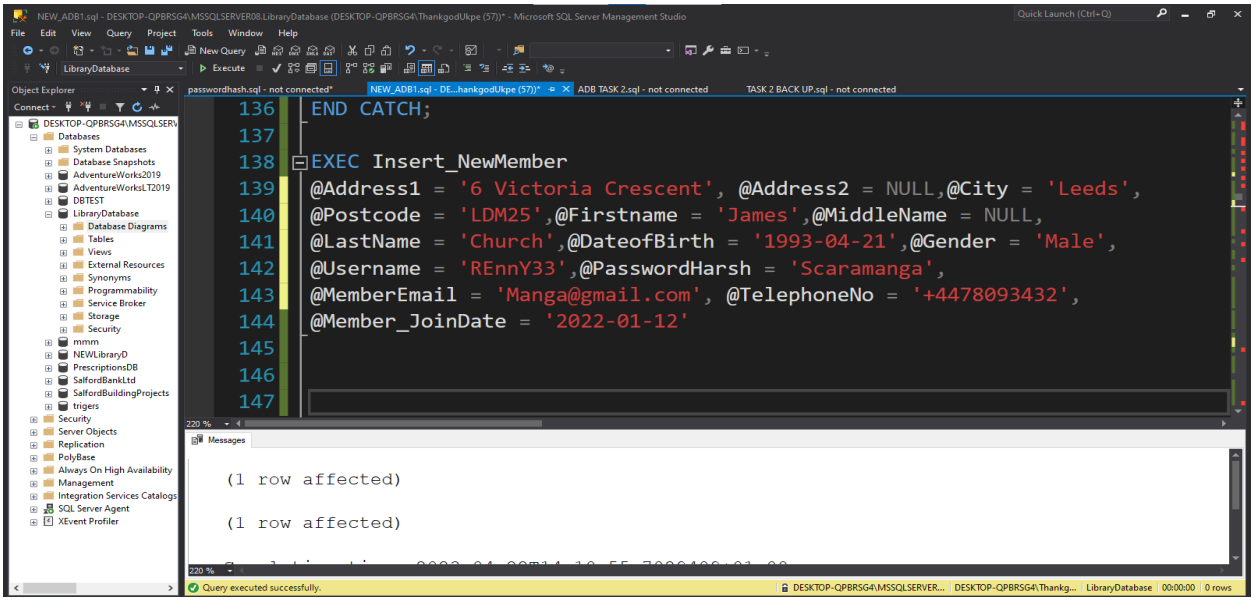


Fig 6.3.2 Executing the Stored procedure in Part 2c above.

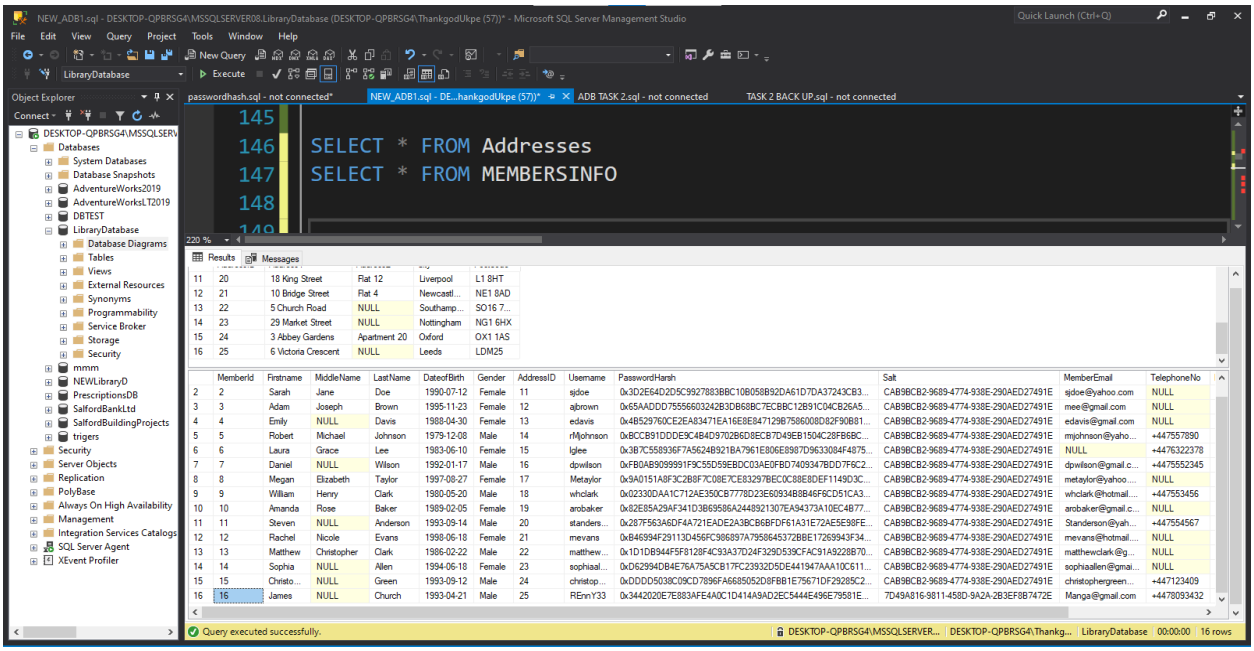


Fig 6.3.3 Select query on the two tables displays the new records inserted.

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.

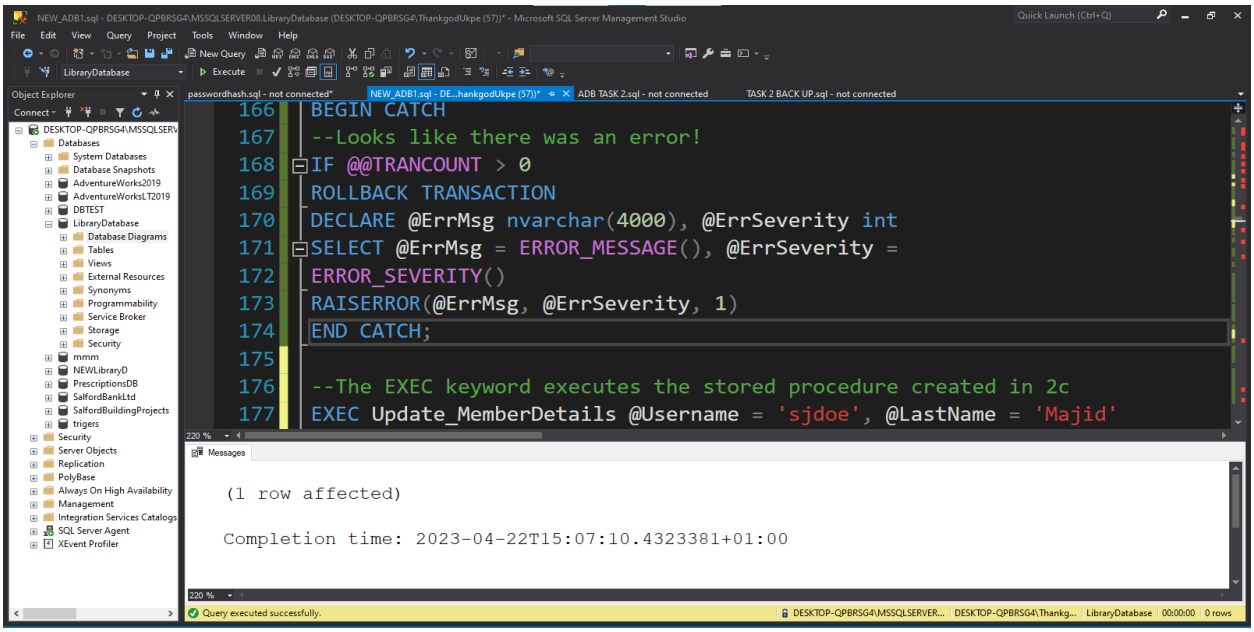


Fig 6.3.4 Executing the stored procedure in Part 2d.

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.

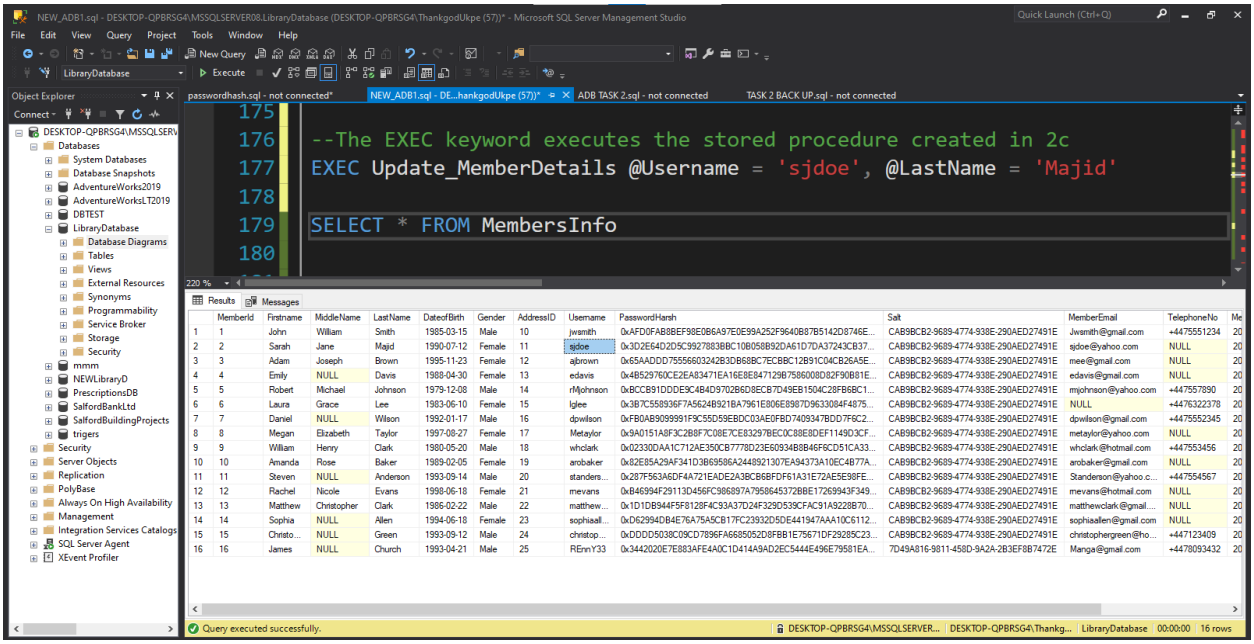


Fig 6.3.5 Select query that displays the affected row on the MembersInfo table.

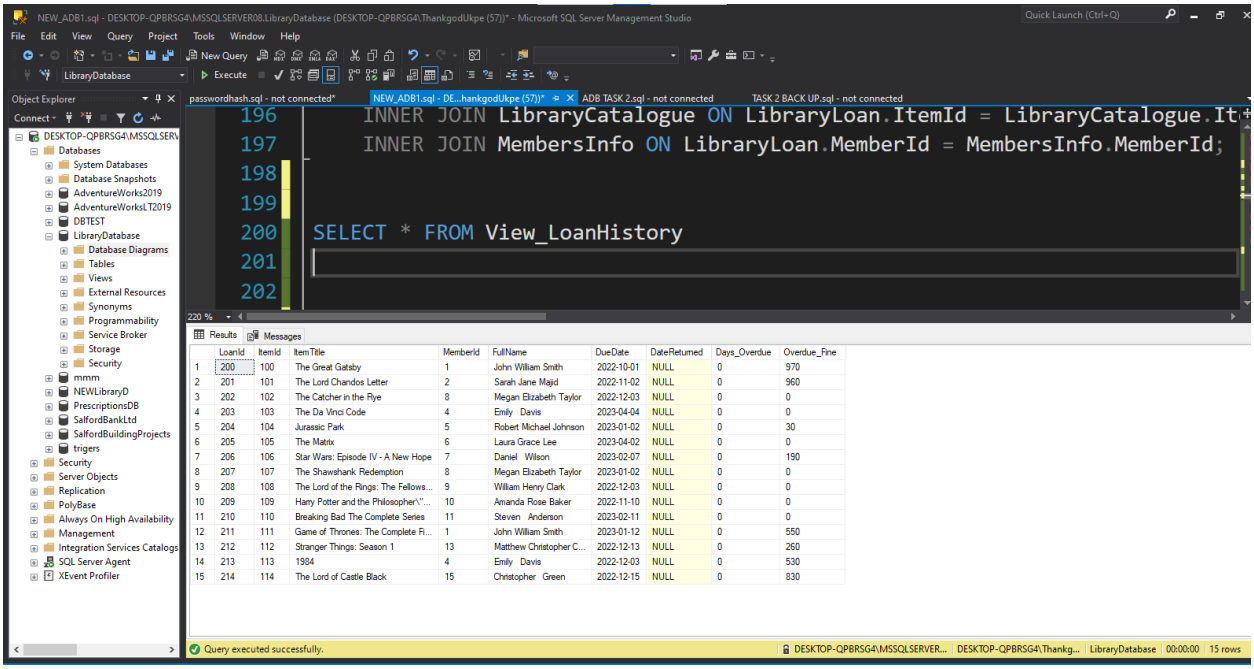


Fig 6.3.6 Select query that displays the created view in Part 3

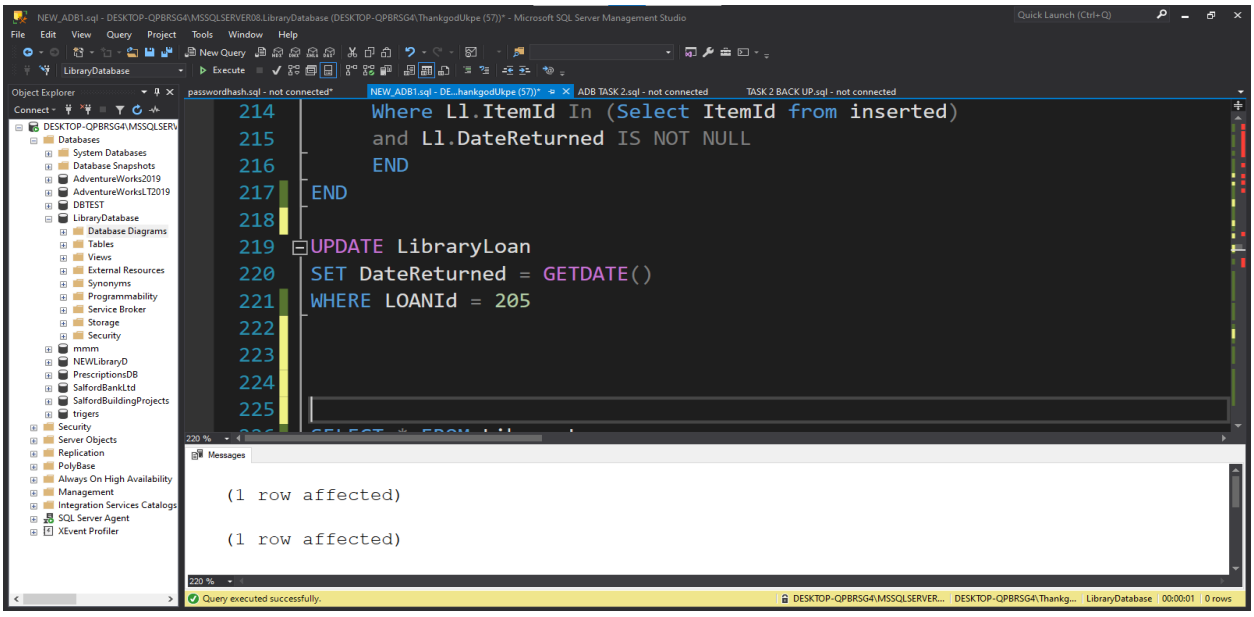


Fig 6.3.7 Update on the LibraryLoan table to fire the Trigger created in Part 4

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.

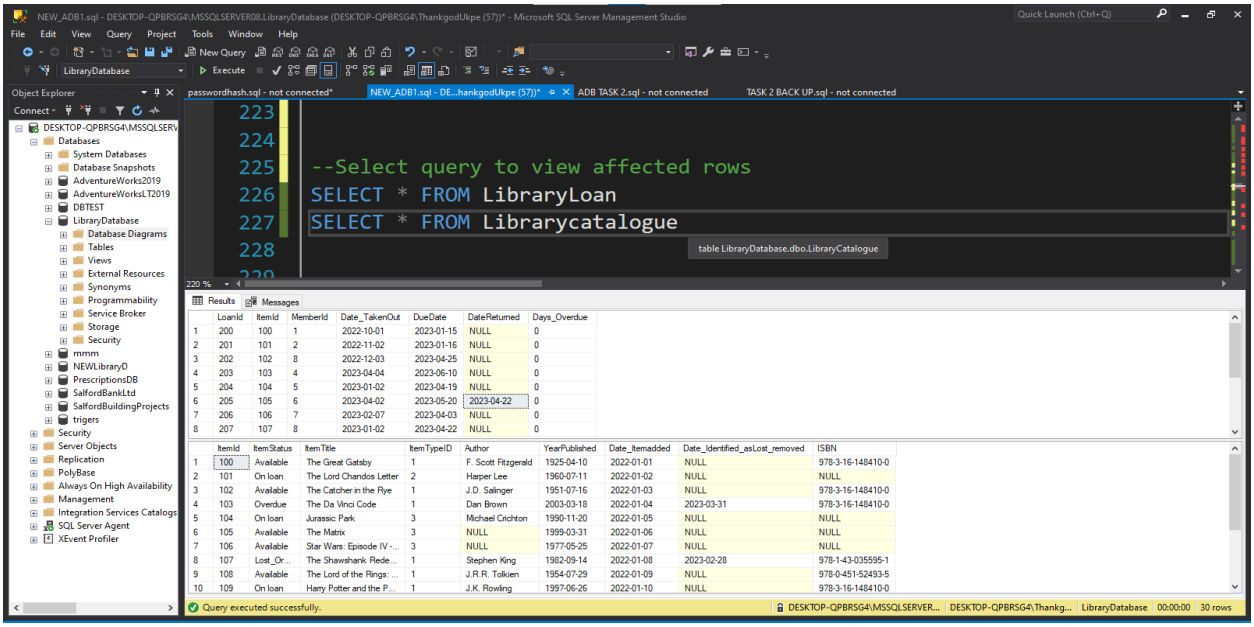


Fig 6.3.8 Select query to view the affected rows.

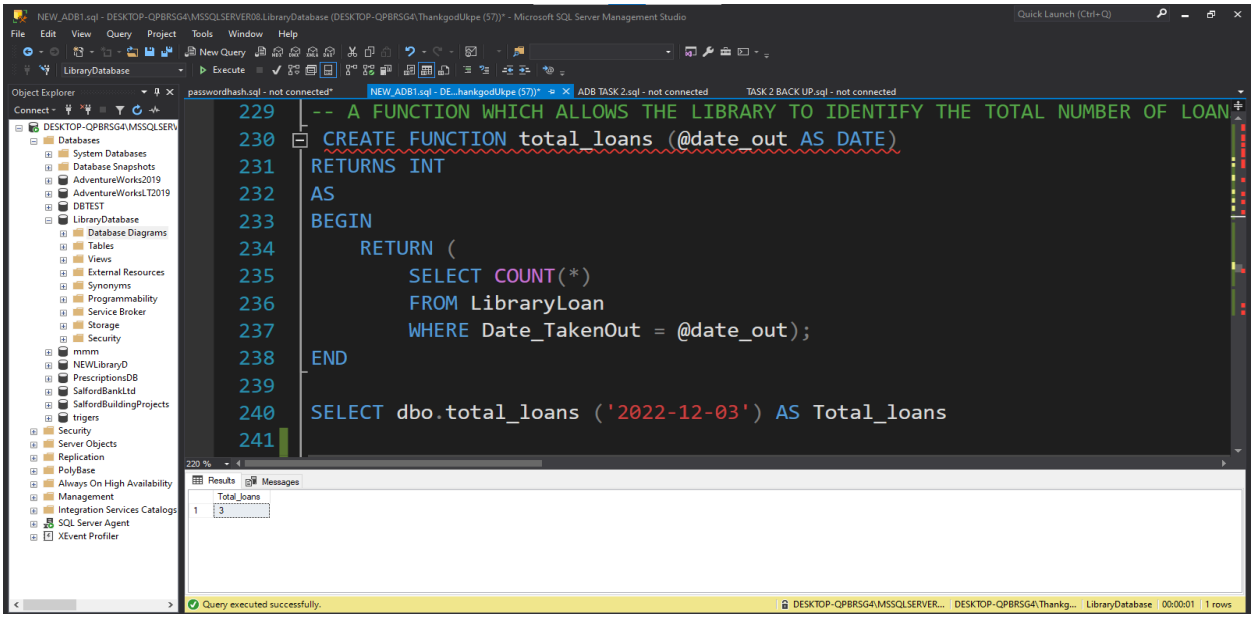


Fig 6.3.9 Select query calls the function created in Part 5 and returns the result.

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.

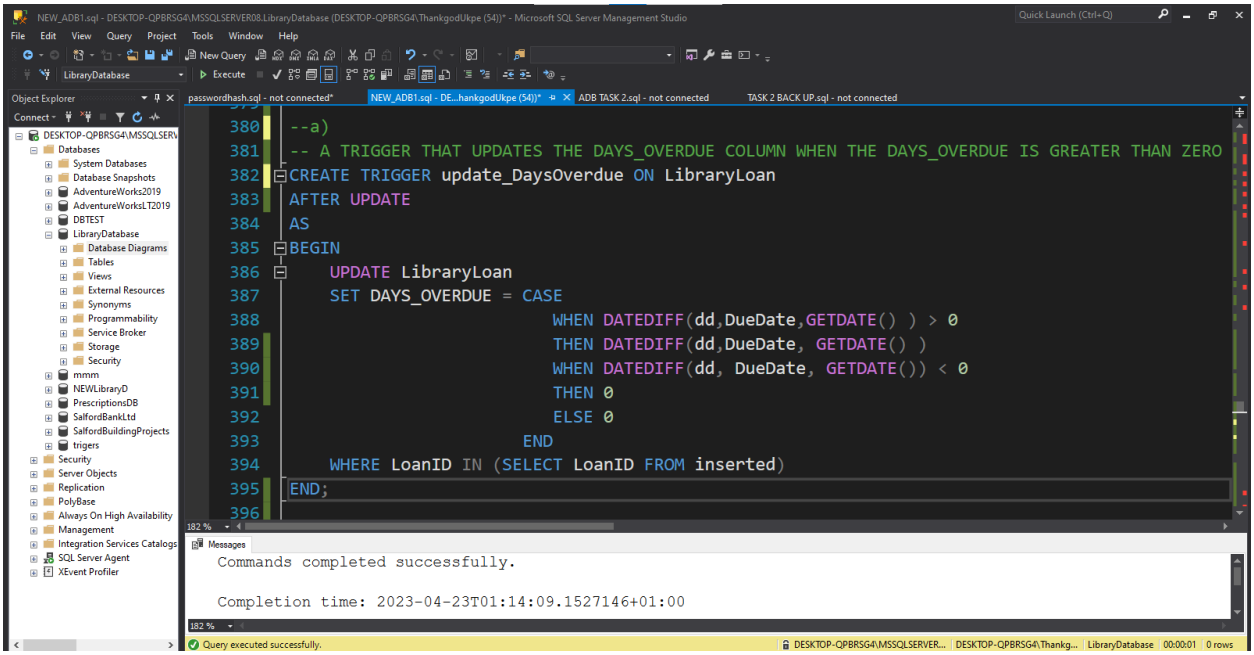


Fig 7a

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.

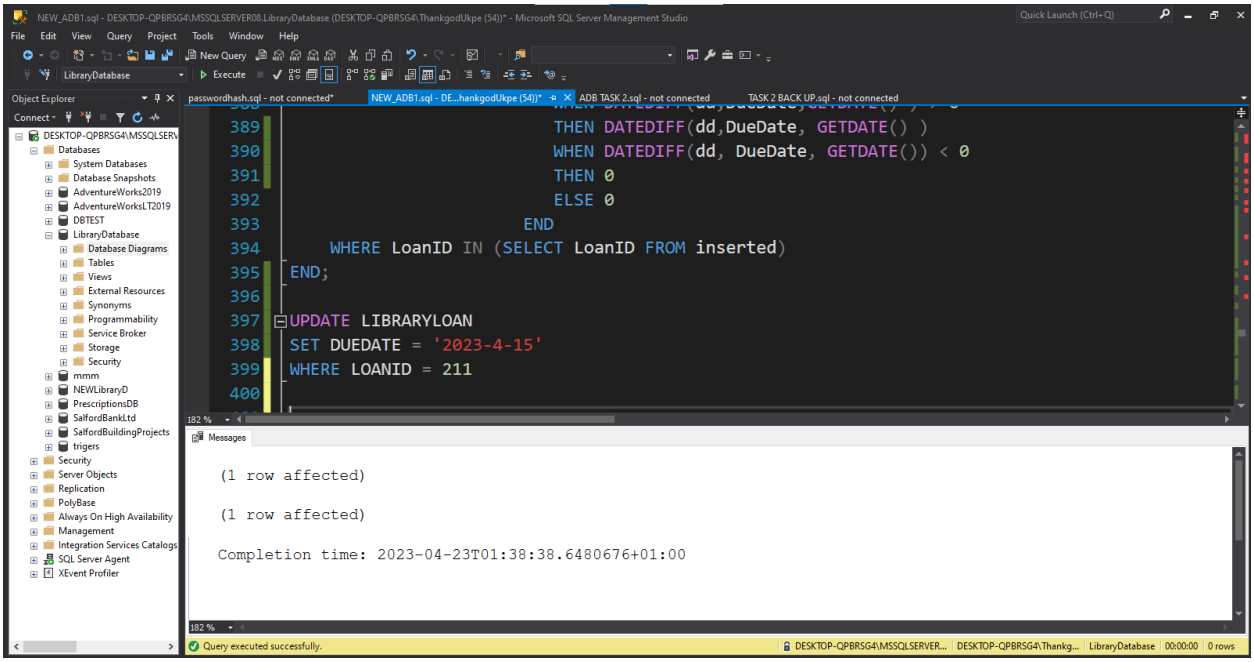


Fig 7a1

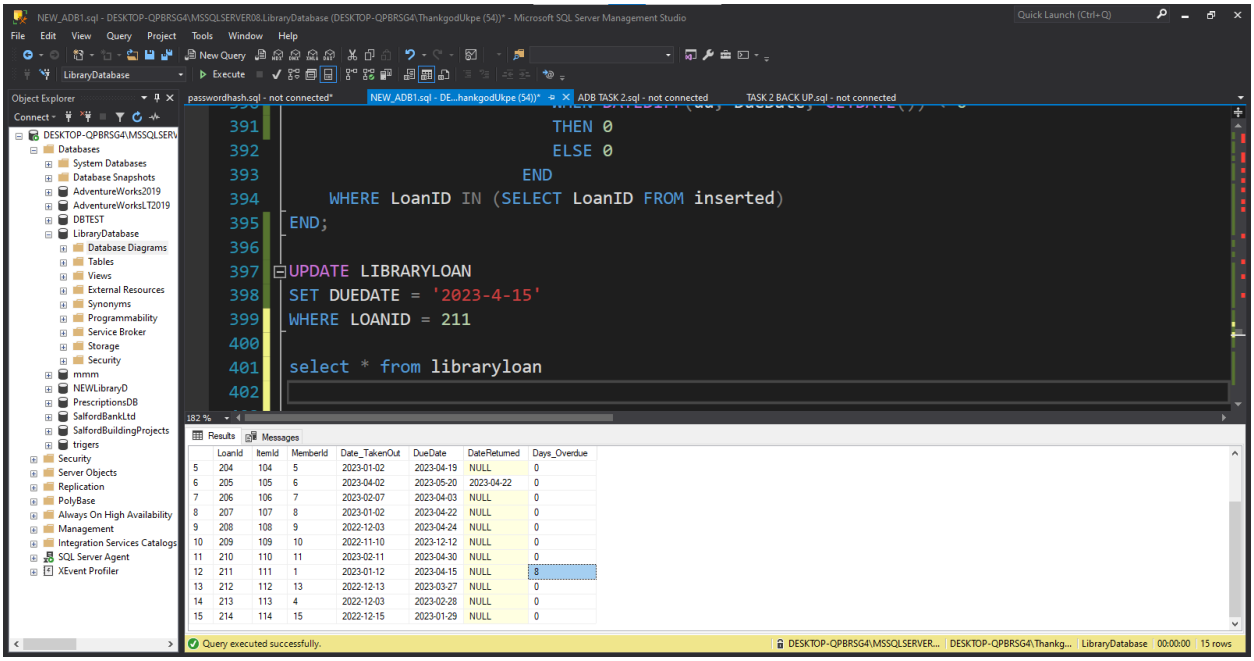


Fig 7a2

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.

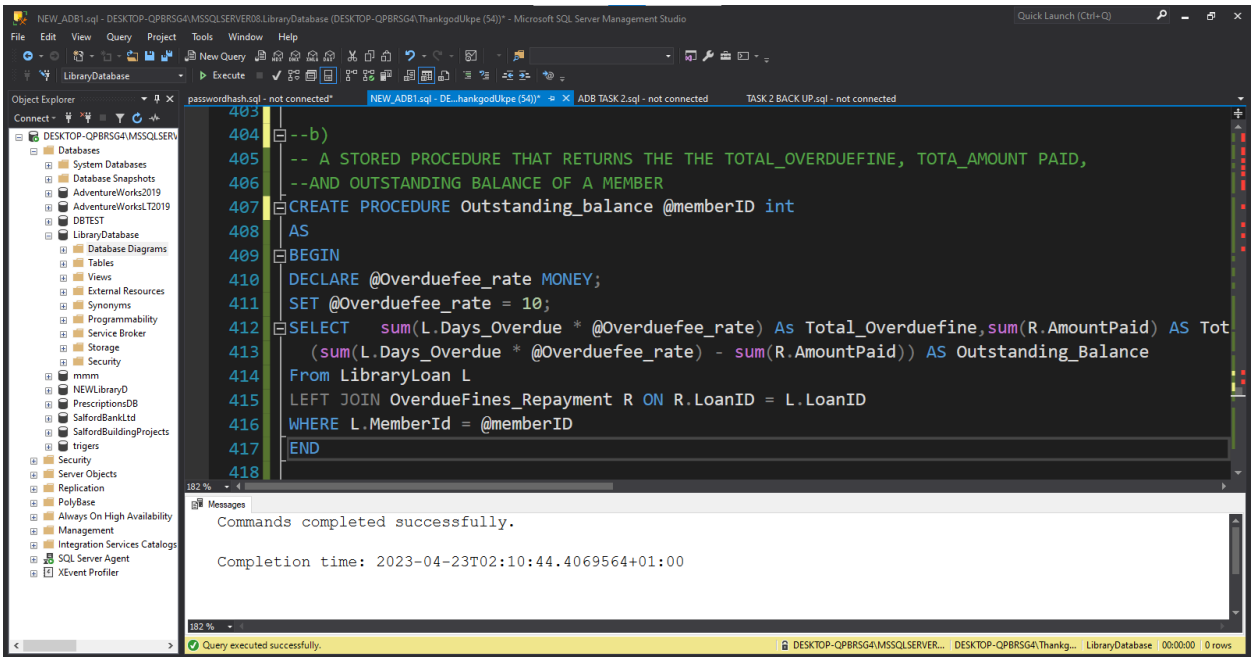


Fig 7b

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.

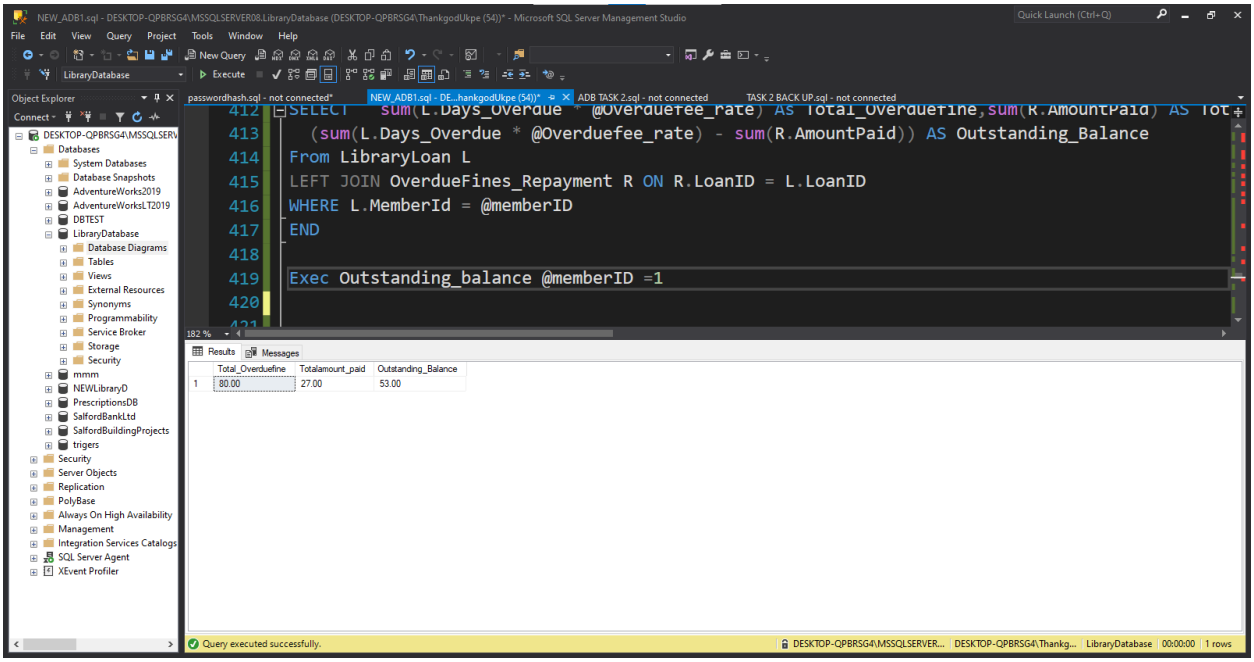


Fig 7b1

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.

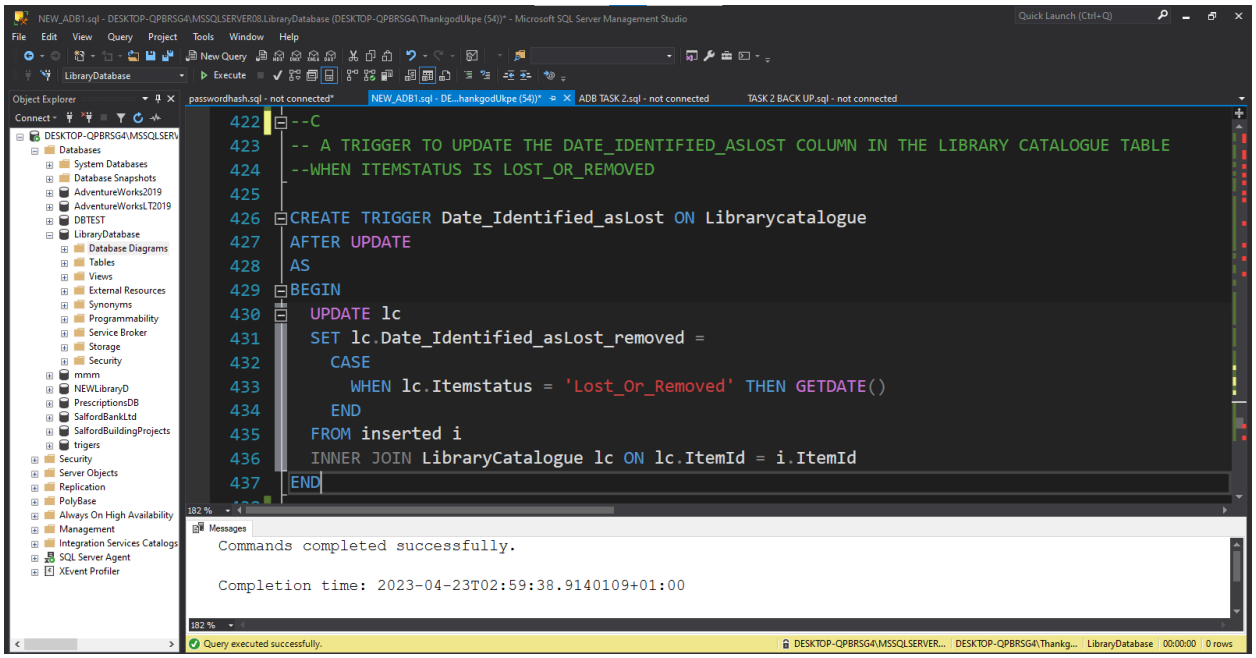


Fig 7c

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.

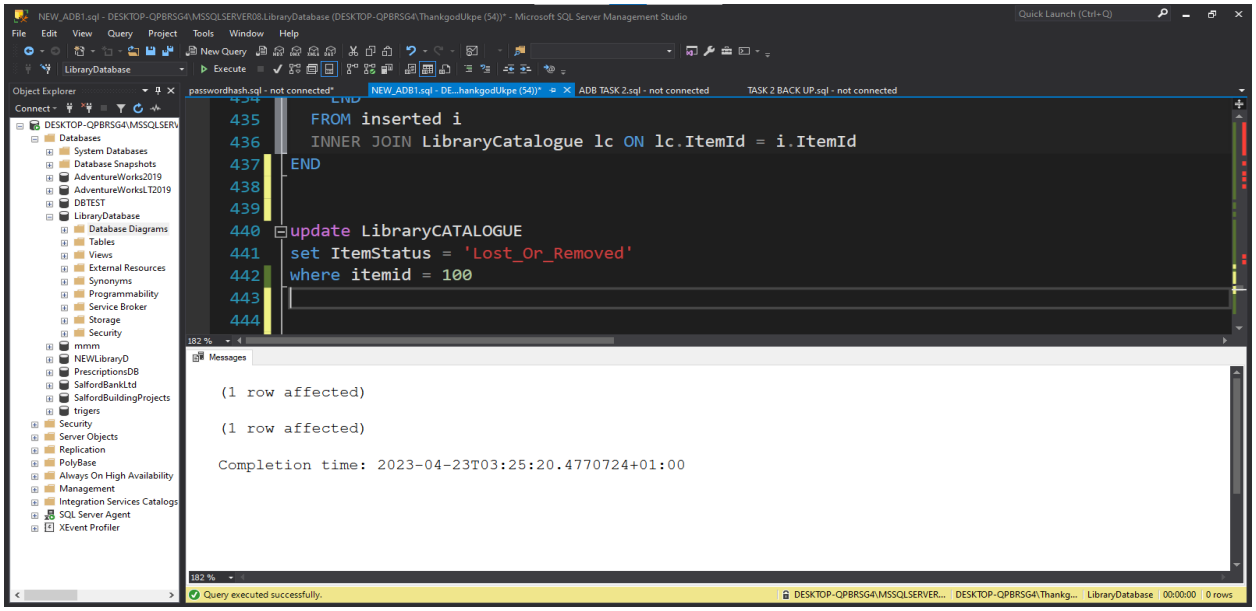


Fig 7c1

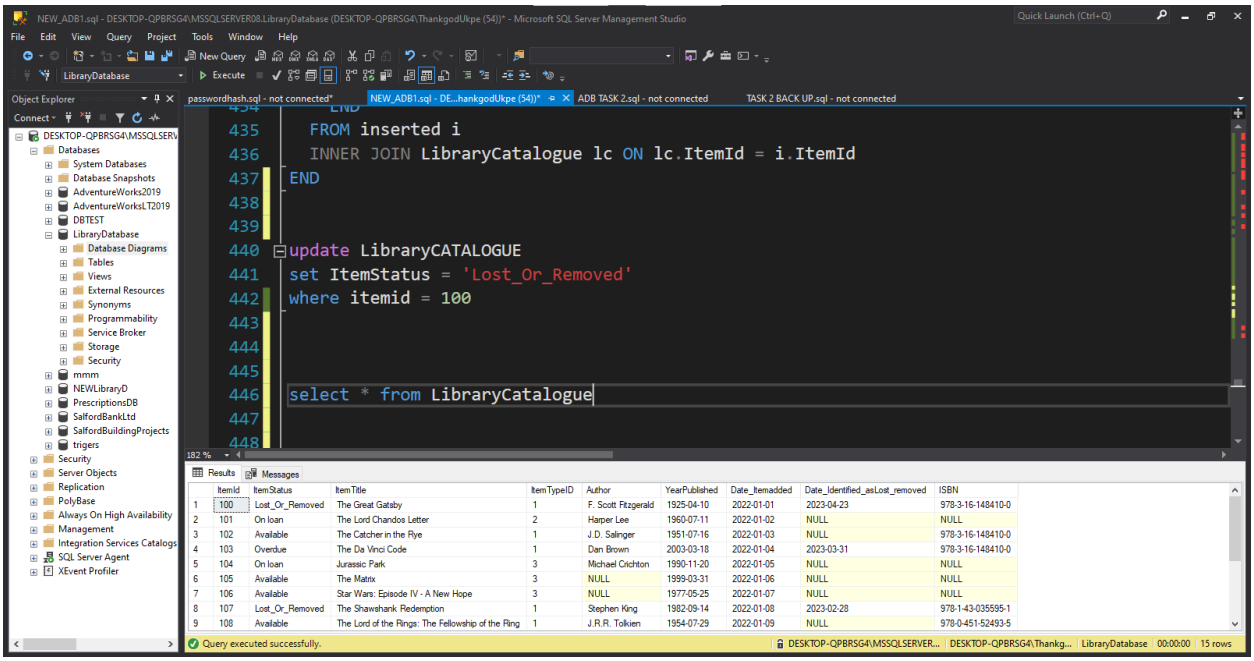


Fig 7c2

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.

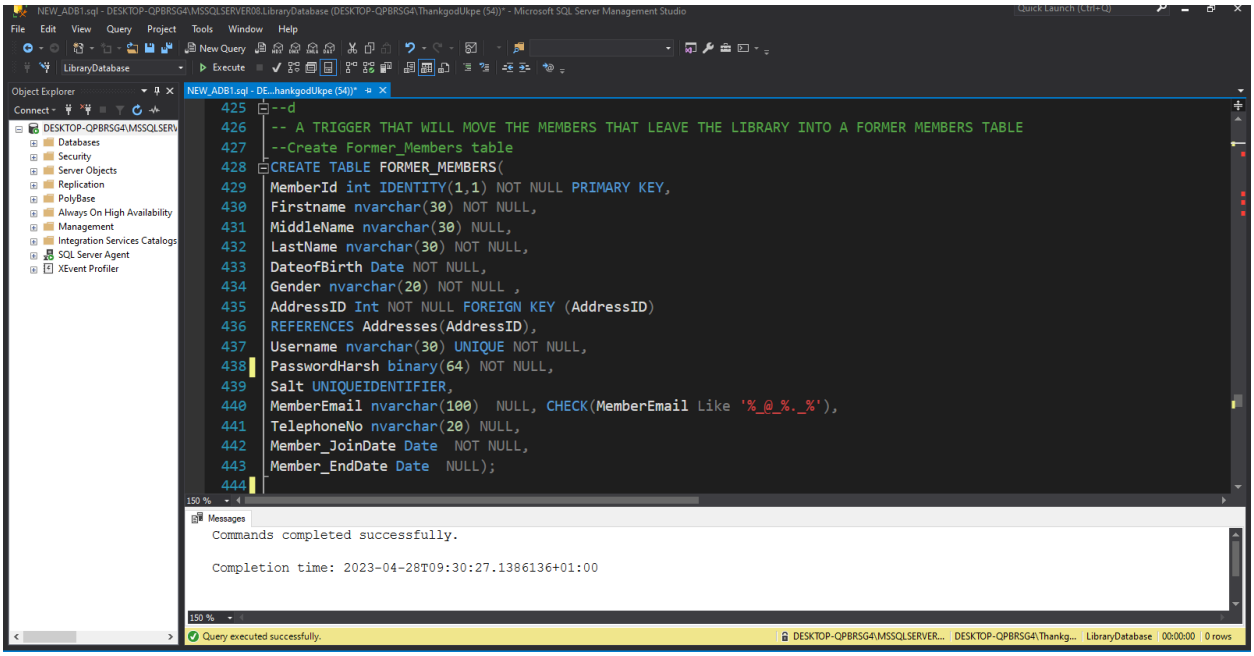


Fig 7d

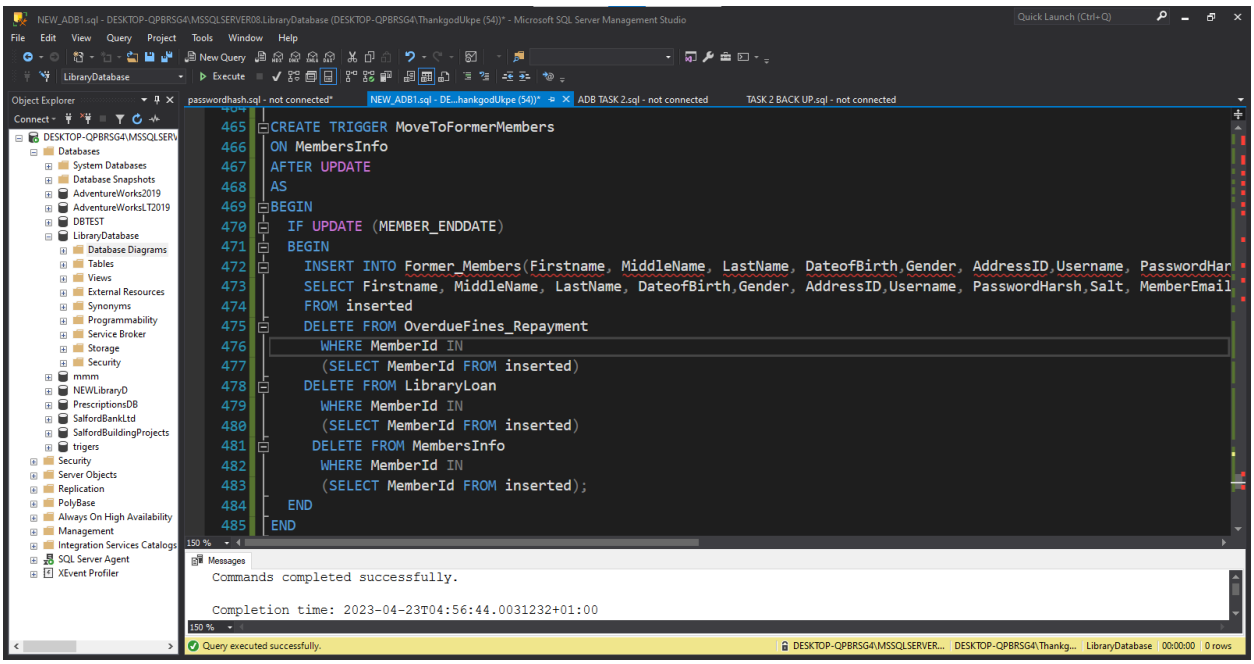


Fig 7d1

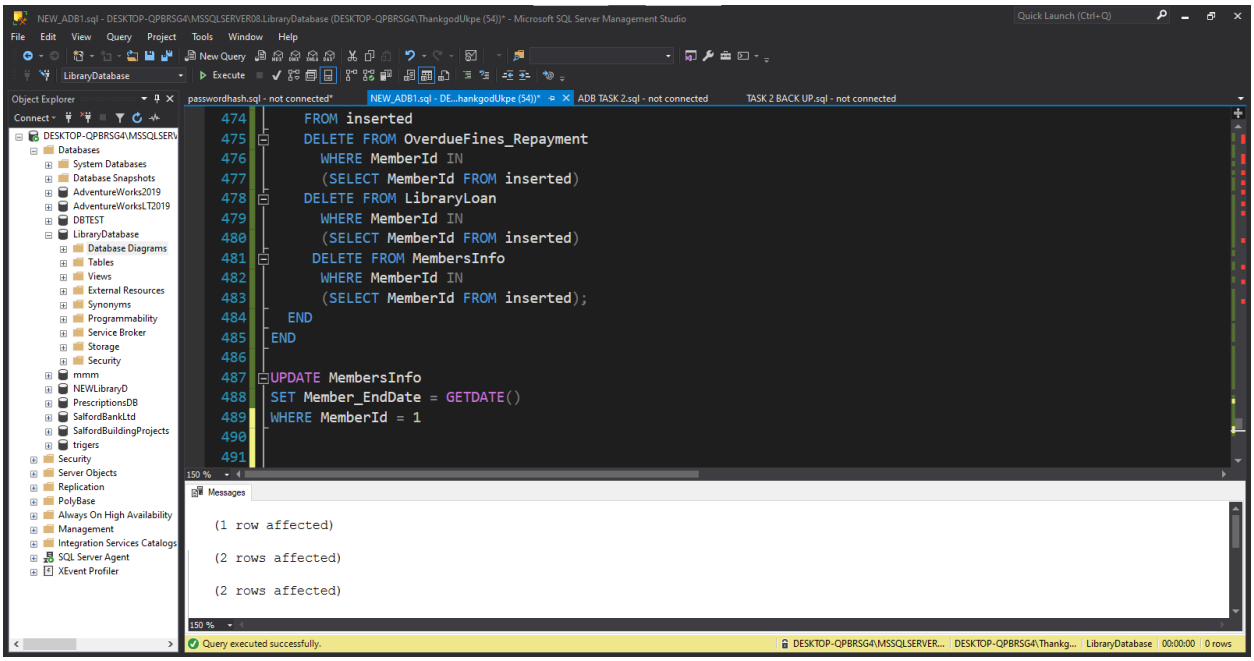


Fig 7d2

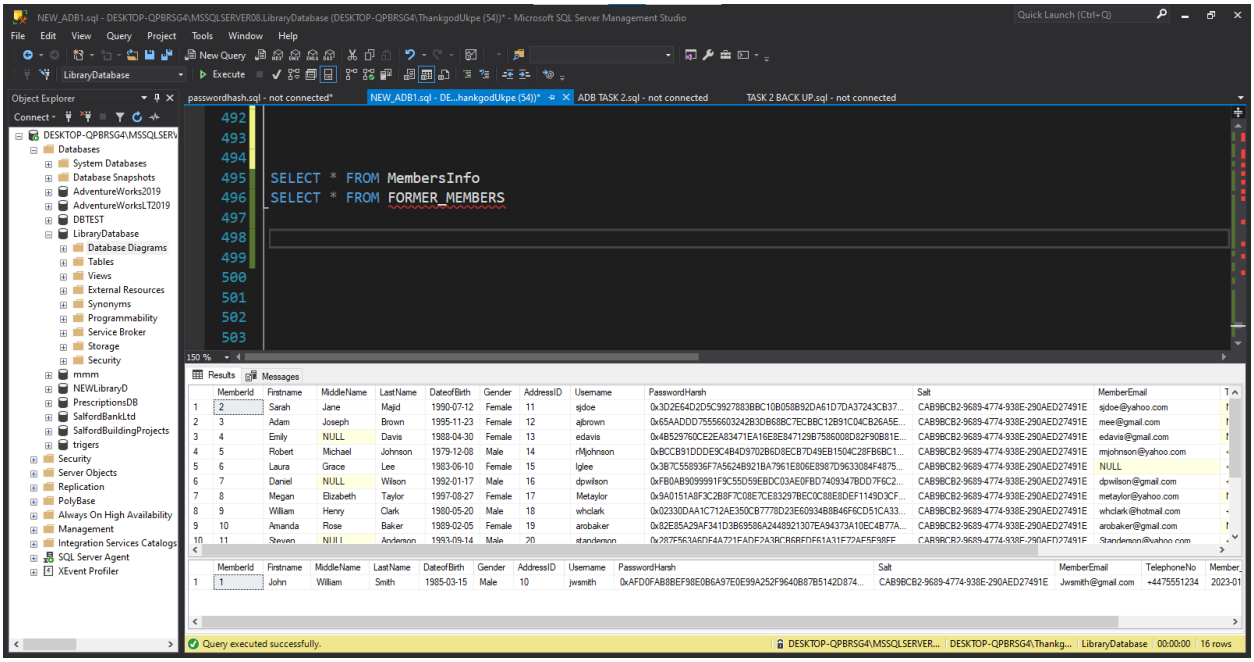


Fig 7d3

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 ‘passwordharsh’ 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 harshing mechanism is an adequate method of storing passwords as it is

computationally infeasible to reverse the harsh 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

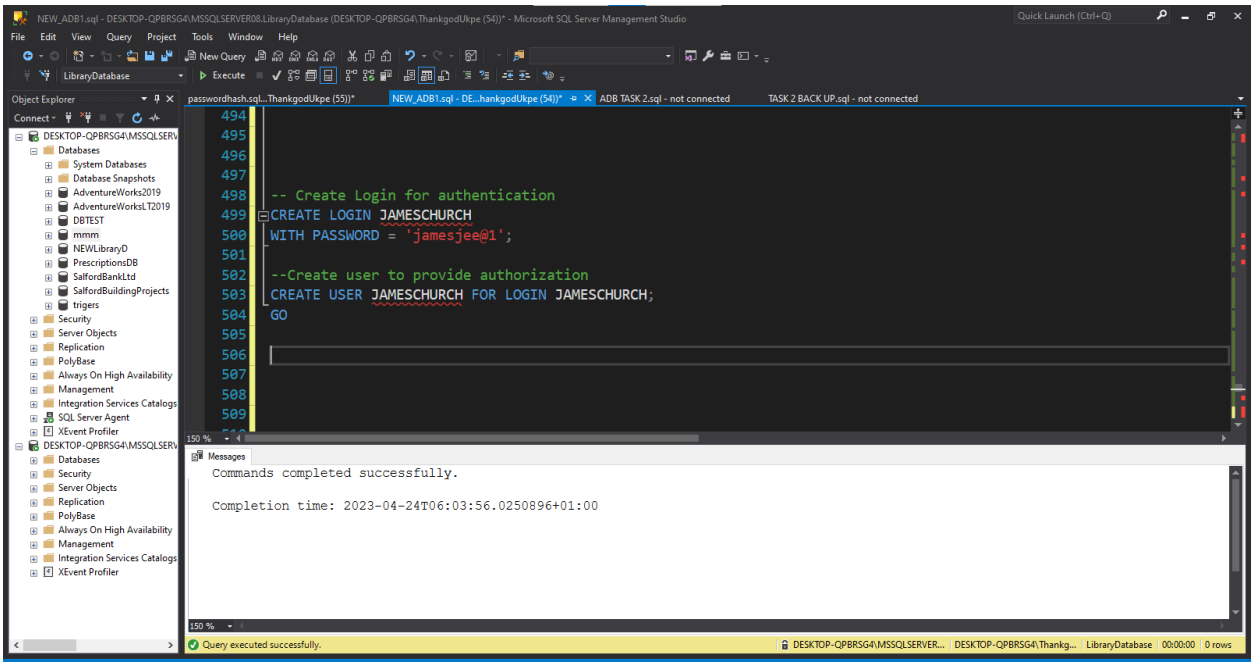


Fig 8.2.2

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.

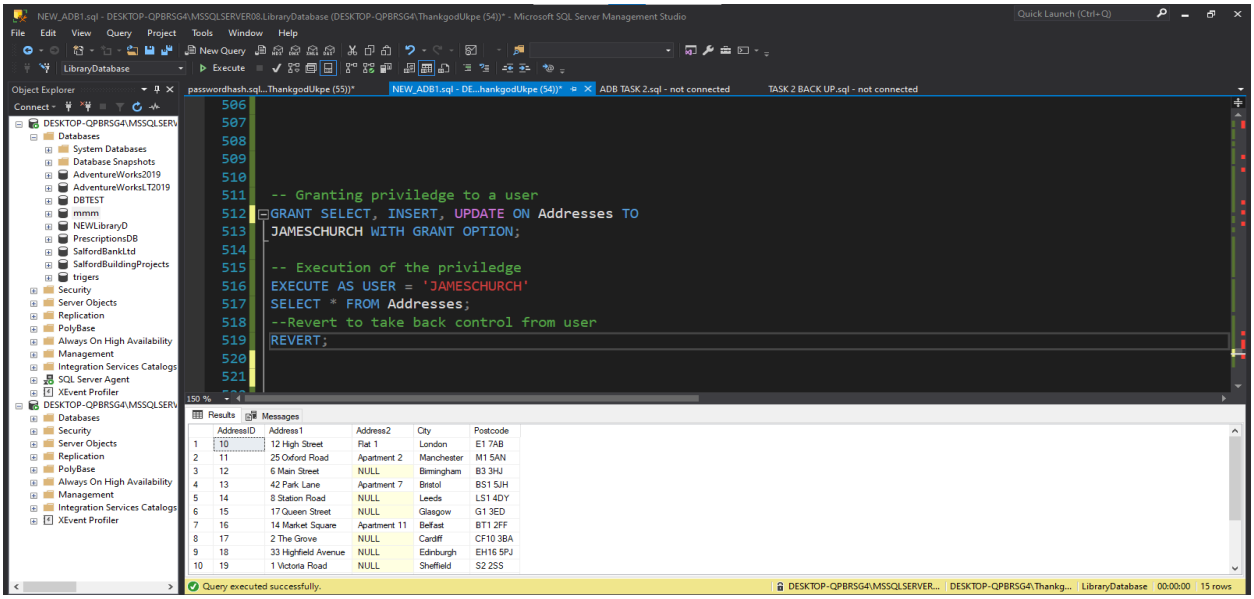


Fig 8.2.3.1

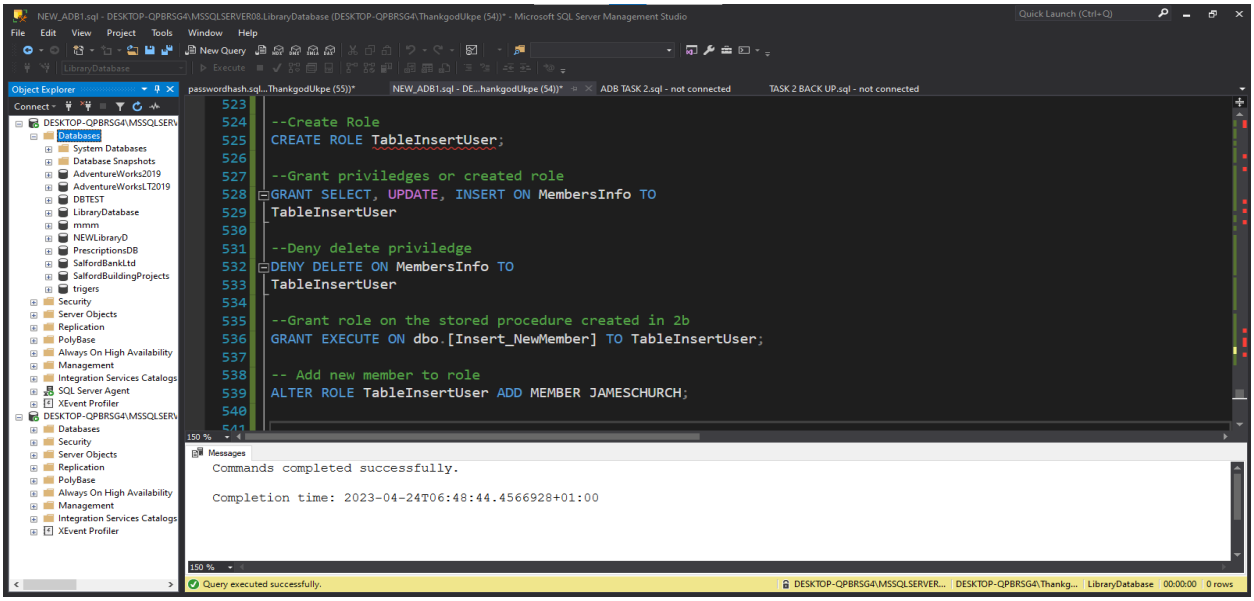


Fig 8.2.3.2

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.