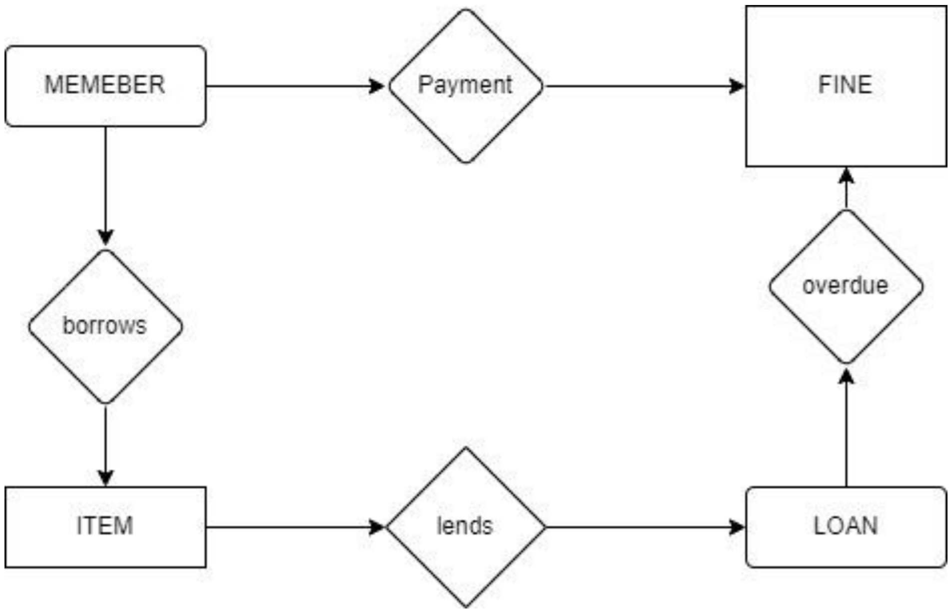
INTRODUCTION

As a database consultant, my aim is to design a database system, that can be used to store information on their members, their library catalogue, loan history and overdue fine repayments. To make sure the system is efficient the relationships between these elements will be accurately defined -linked .

The database will help manage what Item was available, on loan, overdue or removed/lost. This will also help the Library staff effectively and easily track loans and fine fees.

PART 1

According to my client's requirements, my database design model is built up from 4 initial entities.



After consultation with the clients, I had to add more entities to the database to make the design more efficient.

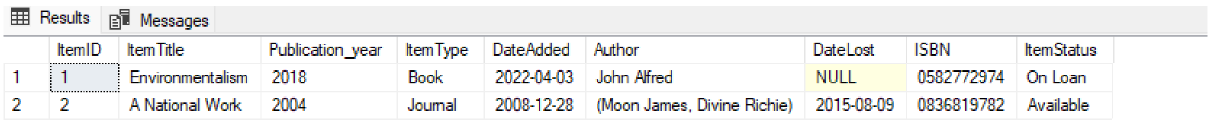
ENTITY	ATTRIBUTES
Member	MemberID, FirstName, LastName, Address, BirthDate, EmailAddress, PhoneNumber, EndDate, Status
Login_Details	UserID, MemberID, Username, Password
MemberArchive	MemberID, FirstName, LastName, Date of Birth, EmailAddress
Items	ItemID, ItemTitle, ItemType, Author, Year of publication, DateAdded, ItemStatus, DateLost/Removed, ISBN
Loans	LoanID, Item_loaned, DateLoaned, DueDate, DateReturned, OverdueFee
Fines	FineID, OverdueFee, Amount_paid, OutstandingBalance PaymentMethod, Date_time

NORMALIZATION OF DATABASE.

Normalization is a database principle that helps to organize data in a consistent way. This involves the creation of tables and establishing relationships between them. It reduces redundancy and maintaining database integrity.

Normalization to 1NF

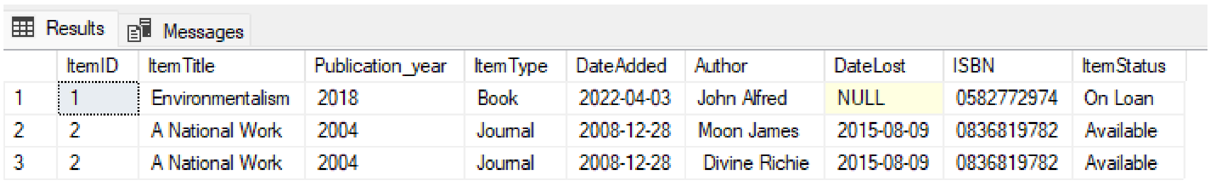
With the database above there are anomalies that violates the rules of first normal form. For instance, in the Item table; an Item may be written by more than one author hence more than one value in a cell.



Results | Messages

	ItemID	ItemTitle	Publication_year	ItemType	DateAdded	Author	DateLost	ISBN	ItemStatus
1	1	Environmentalism	2018	Book	2022-04-03	John Alfred	NULL	0582772974	On Loan
2	2	A National Work	2004	Journal	2008-12-28	(Moon James, Divine Richie)	2015-08-09	0836819782	Available

For a table to be in the first normal form it must no hold more than a single value in a cell.



Results | Messages

	ItemID	ItemTitle	Publication_year	ItemType	DateAdded	Author	DateLost	ISBN	ItemStatus
1	1	Environmentalism	2018	Book	2022-04-03	John Alfred	NULL	0582772974	On Loan
2	2	A National Work	2004	Journal	2008-12-28	Moon James	2015-08-09	0836819782	Available
3	2	A National Work	2004	Journal	2008-12-28	Divine Richie	2015-08-09	0836819782	Available

Reducing the complexity of columns such as the Address column in the Members table by creating an Address table and decomposing its attribute to different columns such as Address1, Address2, City, Postcode.

I identified each set of related data to a Primary Key in each table

I applied these rules to each table on the database.

Normalization to 2NF

There are still anomalies within the database after been normalised to the first normal form. For instance, in the Items table

- Item – (ItemID, ItemTitle, Type, Author, Year of publication, DateAdded, Status, DateLost, ISBN)

To add a new Status in the Item table I will have to add a new item to the table. This doesn't comply with the 2NF criteria because Status is functionally dependent.

Another anomality with the database in 1NF table is that deleting could cause unwanted loss. For instance, in the Item table if I decide to delete a row for a particular Status and Type, this might lead to me losing critical information.

Also applying changes or updating information in the Status and Type column will be difficult and may have overhead.

Hence, I split the Item table into smaller relations.

- Item – (Item_Id, ItemTitle, ItemTypeID, Author, Year of publication, DateAdded, ItemStatusID, DateLost, ISBN)
- ItemStatus – (ItemStatusID, ItemStatus)
- ItemType – (ItemTypeID, ItemType)

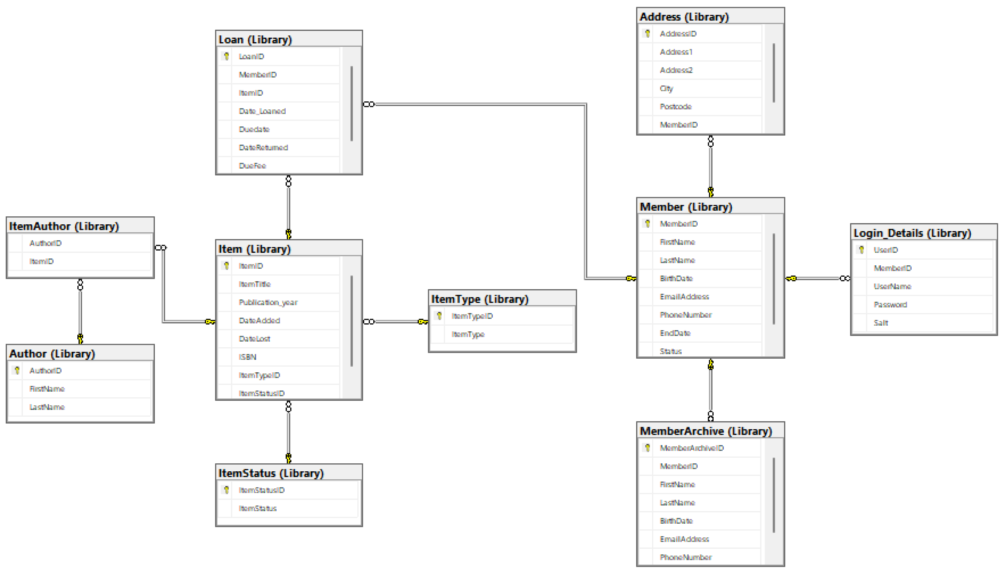
After I split the tables into smaller relations, added foreign keys to the keep the relationships between them

Making sure every table in my database meets the criteria for second normal form

Normalization to 3NF

After normalizing the database to 2NF there is still anomalies within it. An author can write more than one item and an item can be written by more than one author. This leads to difficulty when updating and redundancy, to fix this I created another table for Author and assigning an Id to each author name. I also created a connecting table that consists of the ItemID and AuthorID.

My database then transformed to this;



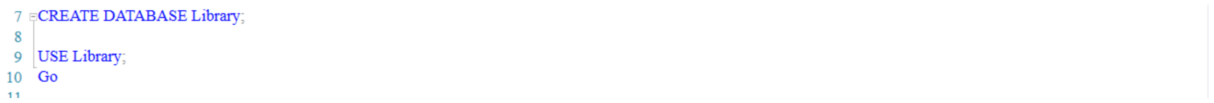
Entity related diagram

Relationship between entities

- One to One
  1. Member and Address: A member can have only one address and vice versa.
  2. Fine and Loan: A fine can belong to only one loan and a loan can only have one fine.
- One to Many
  1. Item to ItemType: An item can have only one item type and an item type can be assigned to more than one item.
  2. Item to ItemStatus:-An item can have only one item status but an item status can be assigned to more than one item.
  3. Member to Loan: A loan can belong to just one member, but a member can make more than one loan.
  4. Fine and Payment: A fine is associated to multiple payment, but a payment is associated to one fine.
- Many to Many
  1. Author to Item: An author can write more than one item and an item can be written by more than one author

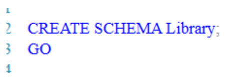
DATABASE DESIGN

I created the database and named it Library



```
CREATE DATABASE Library;

USE Library;
Go
```

I created a schema for my database called Library



```
CREATE SCHEMA Library;
GO
```

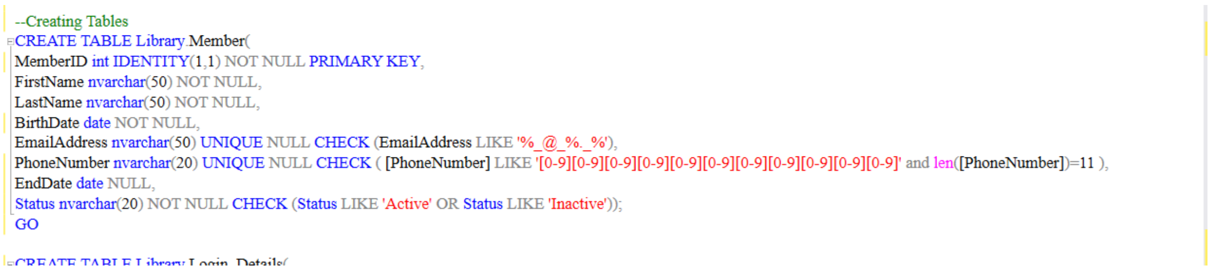
I created my tables for my entity and attributes.

Attributes datatype and description:

Datatype	Attributes	DESCRIPTION
nvarchar(50)	FirstName, LastName, EmailAddress, PhoneNumber, ItemType, AddressID, Address1, Address2, City, Postcode, ItemStatus, ItemTitle, Publication year, ISBN, PaymentMethod, UserName,	These columns contain have the datatype nvarchar because the length of the data entries can vary considerably
Date	BirthDate, EndDate, DateAdded, DateLost/Removed, DateLoaned, DueDate, DateReturned, Payment_Date	These columns are stored in date format because they consist of date at which events occurred
BINARY (64)	Password	This column is storing the member's password in hashed version,
Int	MemberID, ItemID, AddressID, ItemStatusID, LoanID, ItemTypeID, AuthorID,	This column contains numbers in the numerical

	ItemStatusID, FineID, UserID, MemberID	
Money	OverdueFee, OverdueFee, Amount_paid, OutstandingBalance, Repayment Amount	This column contains money
UNIQUEIDENTIFIER	Salt	This column is used in combination with the Password column to create hash valued password.

MEMBERS



```
--Creating Tables
CREATE TABLE Library.Member(
MemberID int IDENTITY(1,1) NOT NULL PRIMARY KEY,
FirstName nvarchar(50) NOT NULL,
LastName nvarchar(50) NOT NULL,
BirthDate date NOT NULL,
EmailAddress nvarchar(50) UNIQUE NULL CHECK (EmailAddress LIKE '%_@_%._%'),
PhoneNumber nvarchar(20) UNIQUE NULL CHECK ( [PhoneNumber] LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]' and len([PhoneNumber])=11 ),
EndDate date NULL,
Status nvarchar(20) NOT NULL CHECK (Status LIKE 'Active' OR Status LIKE 'Inactive'));
GO

CREATE TABLE Library.Login_Details(
```

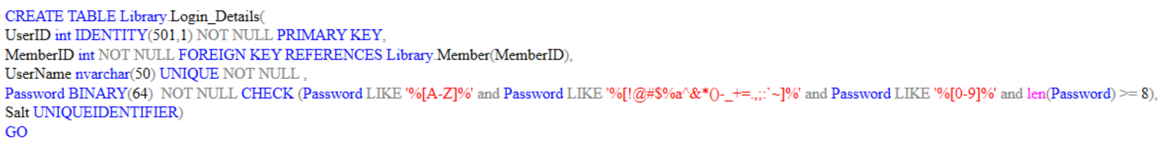
Assigned Keys

MembersID: it is the Primary key of the table and it’s not allowed to be a NULL

Constraints

- EmailAddress: A UNIQUE constraint that makes sure an email address is unique to every member. CHECK constraint helps to enforce some level of data integrity, by ensuring that email addresses adhere to the basic form required for a valid email address.
- PhoneNumber: CHECK constraint helps to enforce some level of data integrity, by ensuring that phone numbers adhere to a particular format and 11 digits long. UNIQUE constraint that makes sura and phone number is unique to every member.
- Status:With the CHECK constraints that makes sure it is active or inactive.

LOGIN_DETAILS



```
CREATE TABLE Library.Login_Details(
UserID int IDENTITY(501,1) NOT NULL PRIMARY KEY,
MemberID int NOT NULL FOREIGN KEY REFERENCES Library.Member(MemberID),
UserName nvarchar(50) UNIQUE NOT NULL ,
Password BINARY(64)  NOT NULL CHECK (Password LIKE '%[A-Z]%' and Password LIKE '%[!@#$%a^&*()-_+=.,;:`~]%' and Password LIKE '%[0-9]%' and len(Password) >= 8),
Salt UNIQUEIDENTIFIER)
GO
```

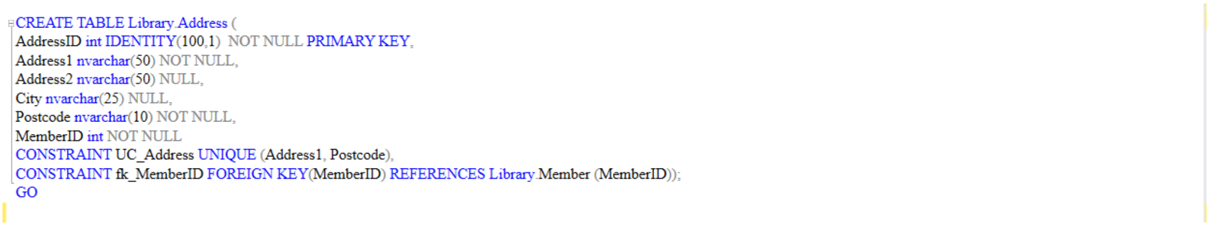
Assigned Keys

- UserID: it is the Primary key of the table and it's not allowed to be a NULL
- MemberID: it is a foreign key that references the member table

Constraints

- UserName:- 'UNIQUE' constraint to add an extra layer of security and uniquely identify each user. It is also assigned the datatype nvarchar(50) because the length of the data entries can vary considerably.
- Password: I also added the CHECK constraint that makes sure the password contains at least a capital letter, one digit, one special character from ('%[!@#$%a^&*()-_+=.,;:`~]%') and at least 8 character long.

ADDRESS



```
CREATE TABLE Library.Address (
AddressID int IDENTITY(100,1)  NOT NULL PRIMARY KEY,
Address1 nvarchar(50) NOT NULL,
Address2 nvarchar(50) NULL,
City nvarchar(25) NULL,
Postcode nvarchar(10) NOT NULL,
MemberID int NOT NULL
CONSTRAINT UC_Address UNIQUE (Address1, Postcode),
CONSTRAINT fk_MemberID FOREIGN KEY(MemberID) REFERENCES Library.Member (MemberID));
GO
```

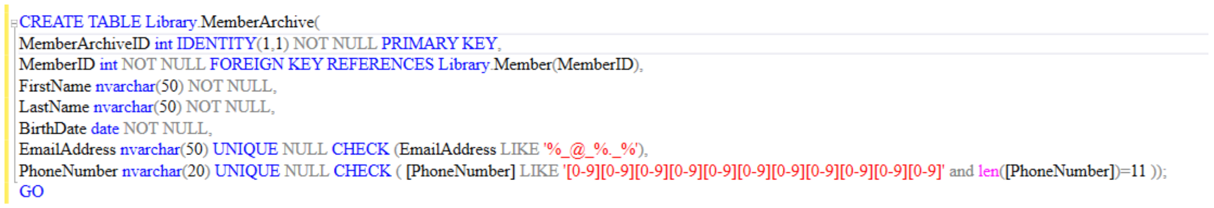
Assigned Keys

- AddressID: it is the Primary key of the table and it's not allowed to be a NULL
- MemberID: it is a foreign key that references the member table

Constraints

- UNIQUE constraint was added to Address1 and Postcode columns combined because in the UK, they uniquely determine the address

MEMBERARCHIVE



```
CREATE TABLE Library.MemberArchive(
MemberArchiveID int IDENTITY(1,1) NOT NULL PRIMARY KEY,
MemberID int NOT NULL FOREIGN KEY REFERENCES Library.Member(MemberID),
FirstName nvarchar(50) NOT NULL,
LastName nvarchar(50) NOT NULL,
BirthDate date NOT NULL,
EmailAddress nvarchar(50) UNIQUE NULL CHECK (EmailAddress LIKE '%_@_%._%'),
PhoneNumber nvarchar(20) UNIQUE NULL CHECK ( [PhoneNumber] LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]' and len([PhoneNumber])=11 ));
GO
```

Assigned Keys

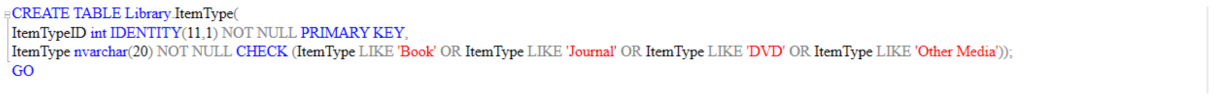
MembersArchiveID: it is the Primary key of the table and it's not allowed to be a NULL

MemberID: it is a foreign key that references the member table

Constraints

- EmailAddress: A UNIQUE constraint that makes sure an email address is unique to every member. CHECK constraint helps to enforce some level of data integrity, by ensuring that email addresses adhere to the basic form required for a valid email address.
- PhoneNumber: CHECK constraint helps to enforce some level of data integrity, by ensuring that phone numbers adhere to a particular format and 11 digits long. UNIQUE constraint that makes sura and phone number is unique to every member.

ITEMTYPE



```
CREATE TABLE Library.ItemType(
ItemTypeID int IDENTITY(11,1) NOT NULL PRIMARY KEY,
ItemType nvarchar(20) NOT NULL CHECK (ItemType LIKE 'Book' OR ItemType LIKE 'Journal' OR ItemType LIKE 'DVD' OR ItemType LIKE 'Other Media'));
GO
```

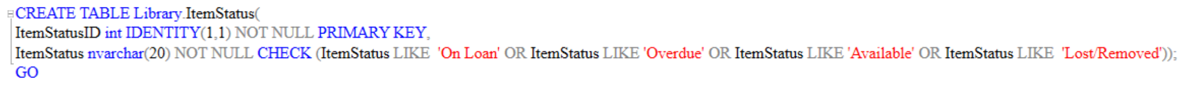
Assigned Keys

- ItemTypeID: it is the Primary key of the table and it's not allowed to be a NULL

Constraints

- ItemType:- CHECK constraints that makes sure it is either book, journal, DVD or other media

ITEMSTATUS



```
CREATE TABLE Library.ItemStatus(
ItemStatusID int IDENTITY(1,1) NOT NULL PRIMARY KEY,
ItemStatus nvarchar(20) NOT NULL CHECK (ItemStatus LIKE  'On Loan' OR ItemStatus LIKE 'Overdue' OR ItemStatus LIKE 'Available' OR ItemStatus LIKE  'Lost/Removed'));
GO
```

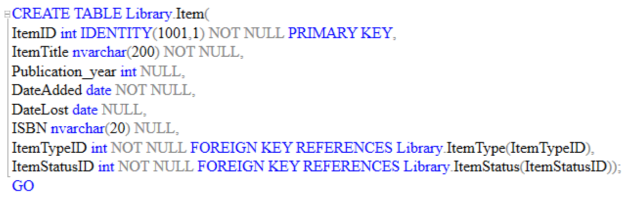
Assigned Keys

- ItemStatusID: it is the Primary key of the table and it’s not allowed to be a NULL

Constraints.

- ItemStatus:- CHECK constraints that makes sure it is either on loan, overdue, available or lost/removed.

ITEM

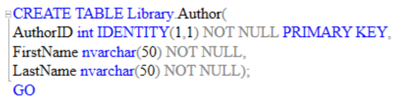


```
CREATE TABLE Library.Item(
ItemID int IDENTITY(1001,1) NOT NULL PRIMARY KEY,
ItemTitle nvarchar(200) NOT NULL,
Publication_year int NULL,
DateAdded date NOT NULL,
DateLost date NULL,
ISBN nvarchar(20) NULL,
ItemTypeID int NOT NULL FOREIGN KEY REFERENCES Library.ItemType(ItemTypeID),
ItemStatusID int NOT NULL FOREIGN KEY REFERENCES Library.ItemStatus(ItemStatusID));
GO
```

Assigned Keys

- ItemID: it is the Primary key of the table and it’s not allowed to be a NULL
- ItemTypeID: it is a foreign key that references the ItemType table
- ItemStatusID: it is a foreign key that references the ItemStatus table

AUTHOR

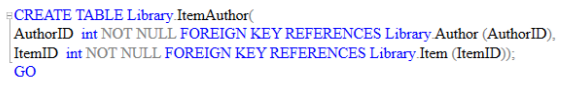


```
CREATE TABLE Library.Author(
AuthorID int IDENTITY(1,1) NOT NULL PRIMARY KEY,
FirstName nvarchar(50) NOT NULL,
LastName nvarchar(50) NOT NULL);
GO
```

Assigned Keys

- AuthorID: it is the Primary key of the table and it’s not allowed to be a NULL

ITEMAUTHOR



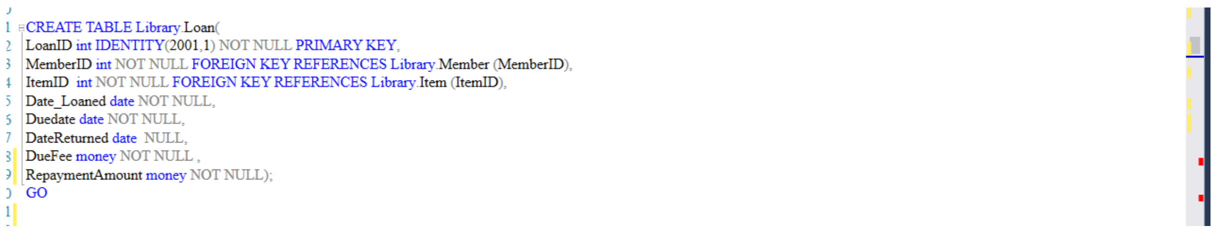
```
CREATE TABLE Library.ItemAuthor(
AuthorID  int NOT NULL FOREIGN KEY REFERENCES Library.Author (AuthorID),
ItemID  int NOT NULL FOREIGN KEY REFERENCES Library.Item (ItemID));
GO
```

Assigned Keys

- AuthorID: This column is a foreign key that references the AuthorID column in Author table. s.

- ItemID: This column is a foreign key that references the ItemID column in Item table.

LOAN

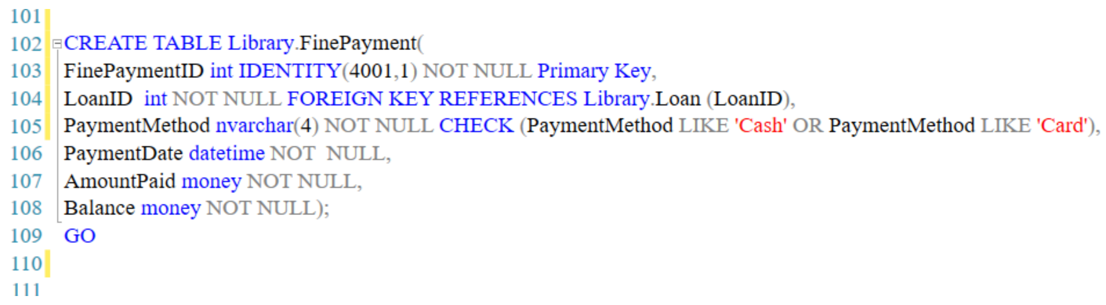


```
CREATE TABLE Library.Loan(
LoanID int IDENTITY(2001,1) NOT NULL PRIMARY KEY,
MemberID int NOT NULL FOREIGN KEY REFERENCES Library.Member (MemberID),
ItemID  int NOT NULL FOREIGN KEY REFERENCES Library.Item (ItemID),
Date_Loaned date NOT NULL,
Duedate date NOT NULL,
DateReturned date  NULL,
DueFee money NOT NULL ,
RepaymentAmount money NOT NULL);
GO
```

Assigned Keys

- AuthorID: it is the Primary key of the table and it's not allowed to be a NULL
- MemberID: This column is a foreign key that references the MemberID
- ItemID:- This column is a foreign key that references the ItemID column in Item table.

FINEPAYMENTS



```
CREATE TABLE Library.FinePayment(
FinePaymentID int IDENTITY(4001,1) NOT NULL Primary Key,
LoanID  int NOT NULL FOREIGN KEY REFERENCES Library.Loan (LoanID),
PaymentMethod nvarchar(4) NOT NULL CHECK (PaymentMethod LIKE 'Cash' OR PaymentMethod LIKE 'Card'),
PaymentDate datetime NOT  NULL,
AmountPaid money NOT NULL,
Balance money NOT NULL);
GO
```

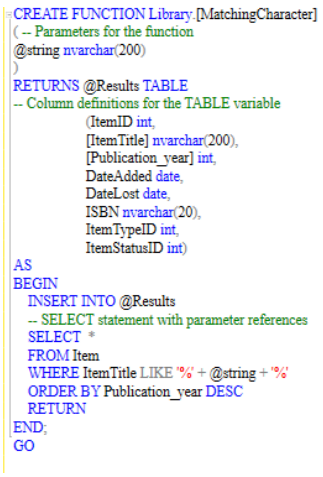
Assigned Keys

- PaymentID: i t is the Primary key of the table and it's not allowed to be a NULL
- FineID:- This column is a foreign key that references the FineID column in Fine table.

Constraints

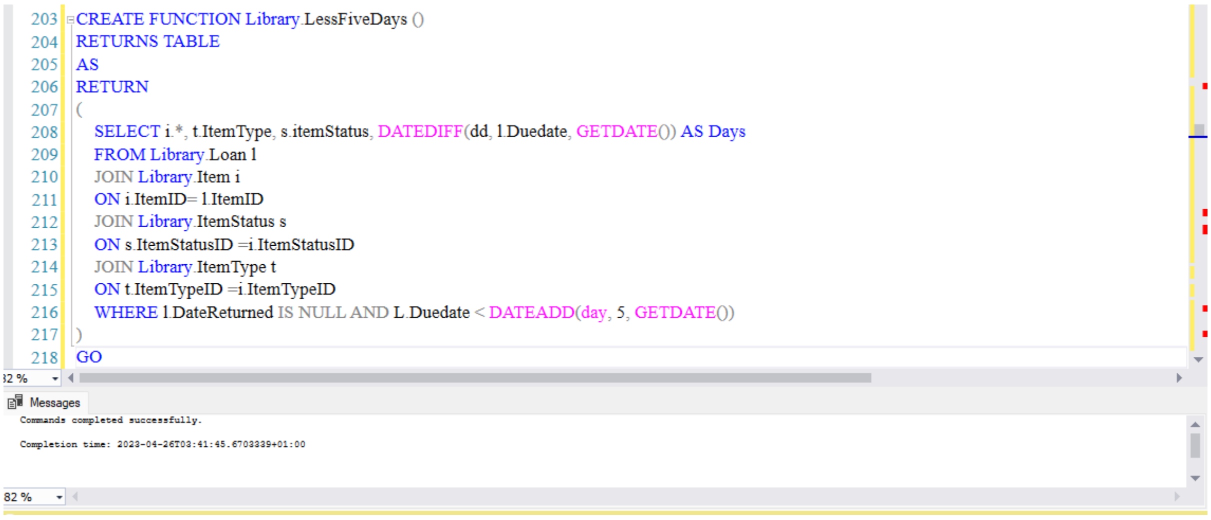
- PaymentMethod: With the CHECK constraints that makes sure it is either cash, or card, the NOT NULL constraint is used to ensure that the column does not contain null values.

Created a function called the MatchingCharacter. This function the accepts one parameter named @string of type nvarchar(200). It returns the @Results table after looking through the ItemTitle column in the Items table for titles which has the same characters as @string anywhere in them. @Results table is ordered by Publication_year column from with the most recent publication first.



```
CREATE FUNCTION Library.[MatchingCharacter]
( -- Parameters for the function
@string nvarchar(200)
)
RETURNS @Results TABLE
-- Column definitions for the TABLE variable
                (ItemID int,
                [ItemTitle] nvarchar(200),
                [Publication_year] int,
                DateAdded date,
                DateLost date,
                ISBN nvarchar(20),
                ItemTypeID int,
                ItemStatusID int)
AS
BEGIN
   INSERT INTO @Results
   -- SELECT statement with parameter references
   SELECT  *
   FROM Item
   WHERE ItemTitle LIKE '%' + @string + '%'
   ORDER BY Publication_year DESC
   RETURN
END;
GO
```

I created the function LessFiveDays(): this function returns a table that consists of the columns in the Item table, the ItemType from the ItemType table the ItemStatus from the ItemStatus table and the difference between current date and the DueDate column in Loans Table in Days. Where the Returned Date in Loans table is NULL and the Duedate in Loans table is less than 5 plus the current date .



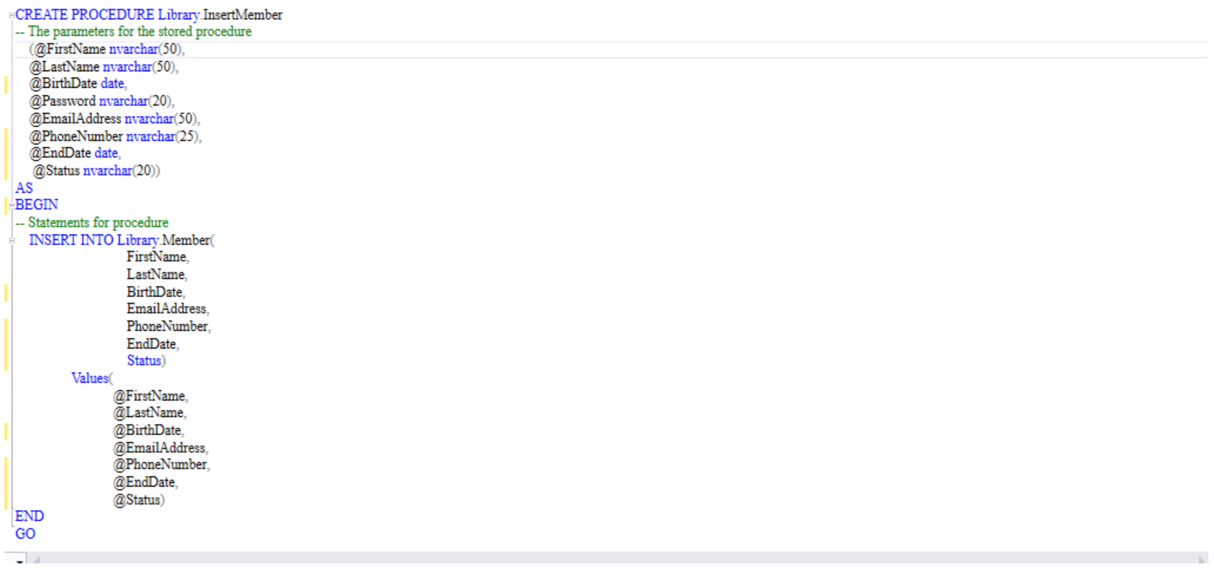
```
CREATE FUNCTION Library.LessFiveDays ()
RETURNS TABLE
AS
RETURN
(
   SELECT i.*, t.ItemType, s.itemStatus, DATEDIFF(dd, l.Duedate, GETDATE()) AS Days
   FROM Library.Loan l
   JOIN Library.Item i
   ON i.ItemID= l.ItemID
   JOIN Library.ItemStatus s
   ON s.ItemStatusID =i.ItemStatusID
   JOIN Library.ItemType t
   ON t.ItemTypeID =i.ItemTypeID
   WHERE l.DateReturned IS NULL AND L.Duedate < DATEADD(day, 5, GETDATE())
)
GO
```

```
Commands completed successfully.

Completion time: 2023-04-26T03:41:45.6703339+01:00
```

Using the CREATE PROCEDURE statement I created a procedure named InsertMember. This procedure's use is to insert new members into the the database.

The parameters for this stored procedures includes values for each column in the members table such as the FirstName, LastName, BirthDate, UserName, Password, MemberAddressID, EmailAddress, PhoneNumber.

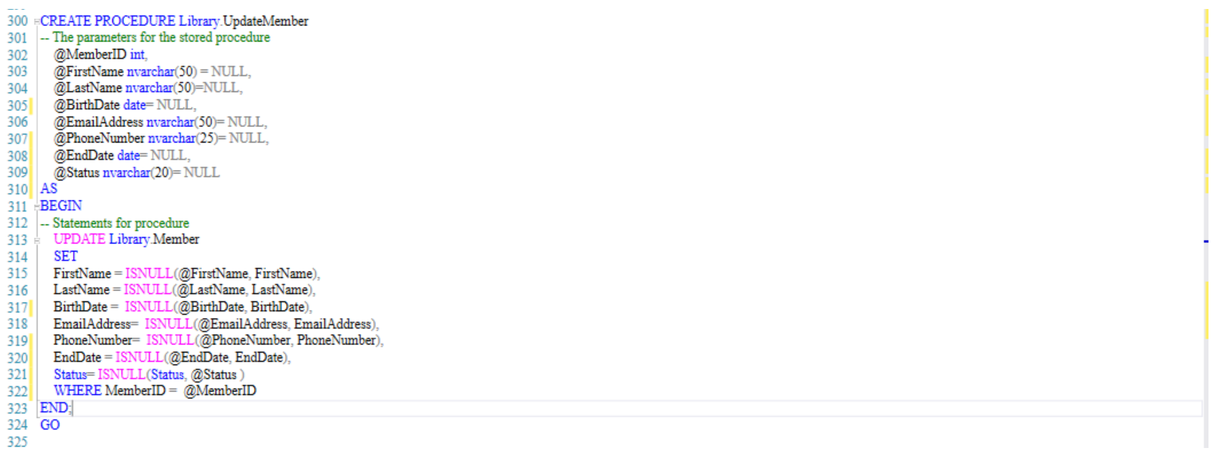
I use the INSERT INTO statement to list the columns I am interested in filling in the Members table and insert each parameters as values into these columns.



```sql
CREATE PROCEDURE Library.InsertMember
-- The parameters for the stored procedure
   (@FirstName nvarchar(50),
   @LastName nvarchar(50),
   @BirthDate date,
   @Password nvarchar(20),
   @EmailAddress nvarchar(50),
   @PhoneNumber nvarchar(25),
   @EndDate date,
    @Status nvarchar(20))
AS
BEGIN
-- Statements for procedure
   INSERT INTO Library.Member(
                        FirstName,
                        LastName,
                        BirthDate,
                        EmailAddress,
                        PhoneNumber,
                        EndDate,
                        Status)
            Values(
                      @FirstName,
                      @LastName,
                      @BirthDate,
                      @EmailAddress,
                      @PhoneNumber,
                      @EndDate,
                      @Status)
END
GO
```

Using the CREATE PROCEDURE statement I created a procedure named UpdateMember. This procedure updates the details of an existing member.

When listing the parameters for procedure I stated that each parameter could be NULL except the MemberID. This means that whenever a member's detail is to be updated the MemberID is compulsory and the details of the column to be updated.

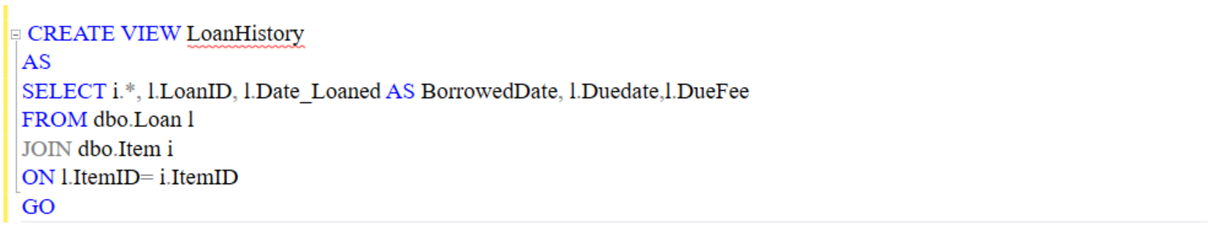


```sql
CREATE PROCEDURE Library.UpdateMember
-- The parameters for the stored procedure
   @MemberID int,
   @FirstName nvarchar(50) = NULL,
   @LastName nvarchar(50)=NULL,
   @BirthDate date= NULL,
   @EmailAddress nvarchar(50)= NULL,
   @PhoneNumber nvarchar(25)= NULL,
   @EndDate date= NULL,
   @Status nvarchar(20)= NULL
AS
BEGIN
-- Statements for procedure
   UPDATE Library.Member
   SET
   FirstName = ISNULL(@FirstName, FirstName),
   LastName = ISNULL(@LastName, LastName),
   BirthDate = ISNULL(@BirthDate, BirthDate),
   EmailAddress= ISNULL(@EmailAddress, EmailAddress),
   PhoneNumber= ISNULL(@PhoneNumber, PhoneNumber),
   EndDate = ISNULL(@EndDate, EndDate),
   Status= ISNULL(Status, @Status )
   WHERE MemberID = @MemberID
END;
GO
```

Using the Create CREATE VIEW statement, I created a view called the LoanHistory. This view returns a table that shows both the current and previous loans including details of the items borrowed.

I did this by including a SELECT statement that returns the all the columns in the Items table as well as the LoanID, BorrowedDate, Duedate and DueFee from the Loan table.

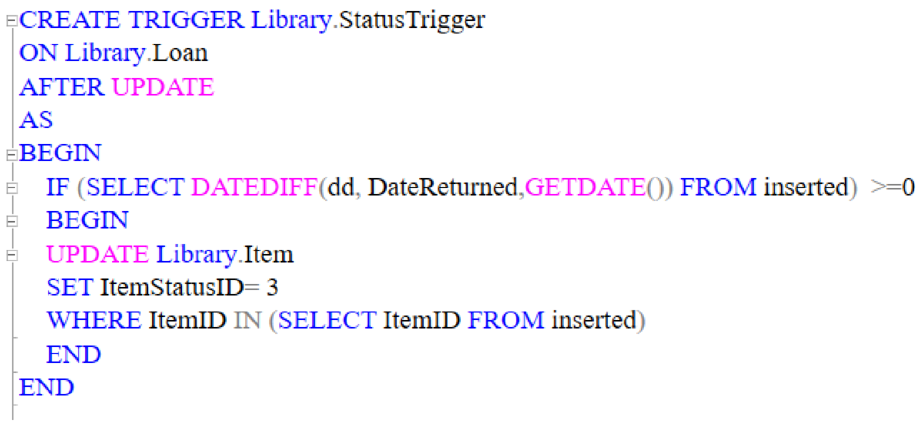
I joined the Items and Loan tables using the JOIN statement on the ItemID column which is present in both tables.



```
CREATE VIEW LoanHistory
AS
SELECT i.*, l.LoanID, l.Date_Loaned AS BorrowedDate, l.Duedate,l.DueFee
FROM dbo.Loan l
JOIN dbo.Item i
ON l.ItemID= i.ItemID
GO
```

I created a trigger called Library.StatusTrigger, the trigger update the Item table to ‘Available’ when the difference btw the current date and DateReturned column equivalent to zero or greater.

In this case ‘Available’ is identified as 3, because it is represented by a unique identifier in the Item table.

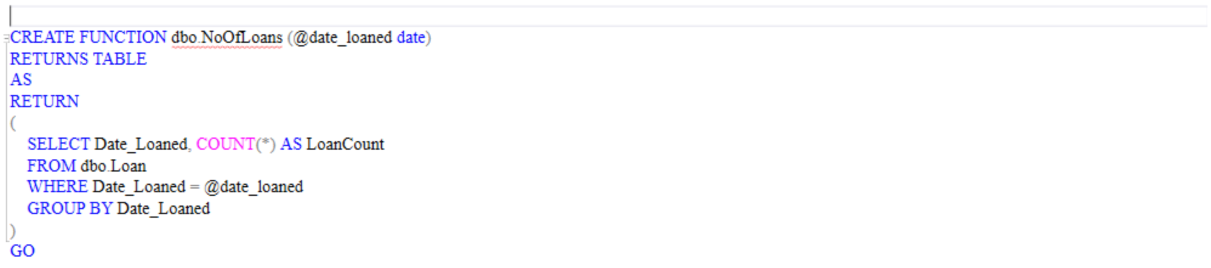


```
CREATE TRIGGER Library.StatusTrigger
ON Library.Loan
AFTER UPDATE
AS
BEGIN
   IF (SELECT DATEDIFF(dd, DateReturned,GETDATE()) FROM inserted)  >=0
   BEGIN
   UPDATE Library.Item
   SET ItemStatusID= 3
   WHERE ItemID IN (SELECT ItemID FROM inserted)
   END
END
```

Using the CREATE FUNCTION statement I created a function called the NoOfLoans function. This function returns total number of loans made an specified date.

NoOfLoans accepts only one parameter @date_loaned of the type date. The SELECT statement within the function returns the Date_Loaned, and uses the

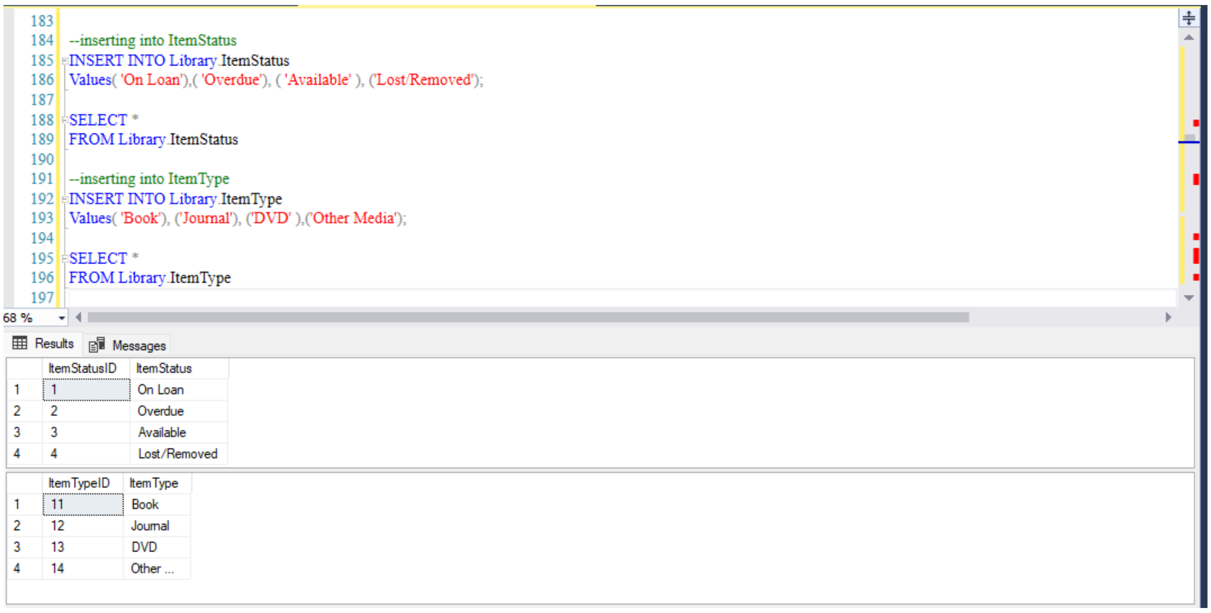
COUNT statement to count the number of Loans made on that date from the Loan table WHERE Date_Loaned is the same as @date_loaned.



```
CREATE FUNCTION dbo.NoOfLoans (@date_loaned date)
RETURNS TABLE
AS
RETURN
(
   SELECT Date_Loaned, COUNT(*) AS LoanCount
   FROM dbo.Loan
   WHERE Date_Loaned = @date_loaned
   GROUP BY Date_Loaned
)
GO
```

To demonstrate how the procedures, views and functions I created earlier works, I inserted values into my tables.

Inserted values into ItemStatus table and the ItemType table so their Primary Key ID can be generated .



```
--inserting into ItemStatus
INSERT INTO Library.ItemStatus
Values( 'On Loan'),( 'Overdue'), ( 'Available' ), ('Lost/Removed');

SELECT *
FROM Library.ItemStatus

--inserting into ItemType
INSERT INTO Library.ItemType
Values( 'Book'), ('Journal'), ('DVD' ),('Other Media');

SELECT *
FROM Library.ItemType
```

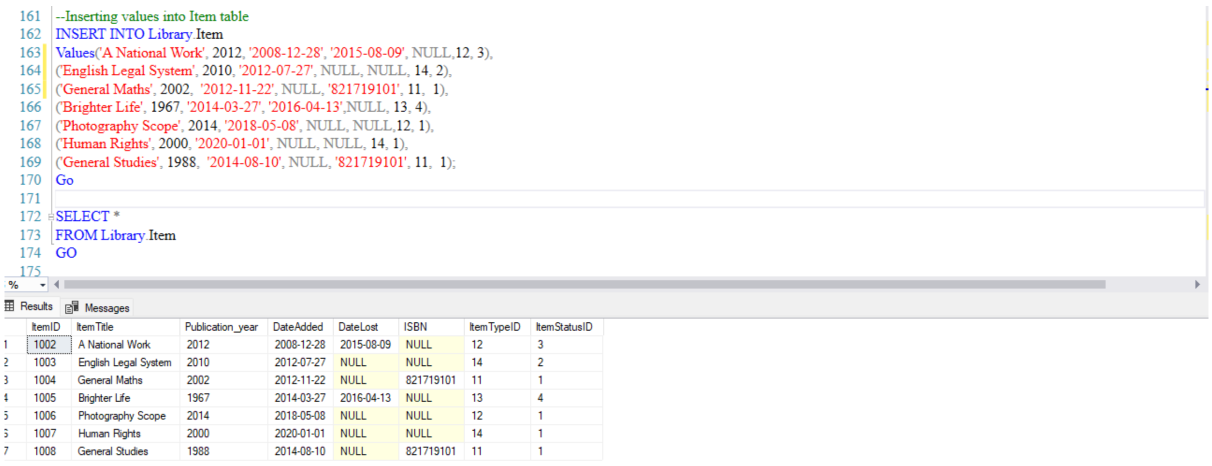
68 %

Results | Messages

	ItemStatusID	ItemStatus
1	1	On Loan
2	2	Overdue
3	3	Available
4	4	Lost/Removed

	ItemTypeID	ItemType
1	11	Book
2	12	Journal
3	13	DVD
4	14	Other ...

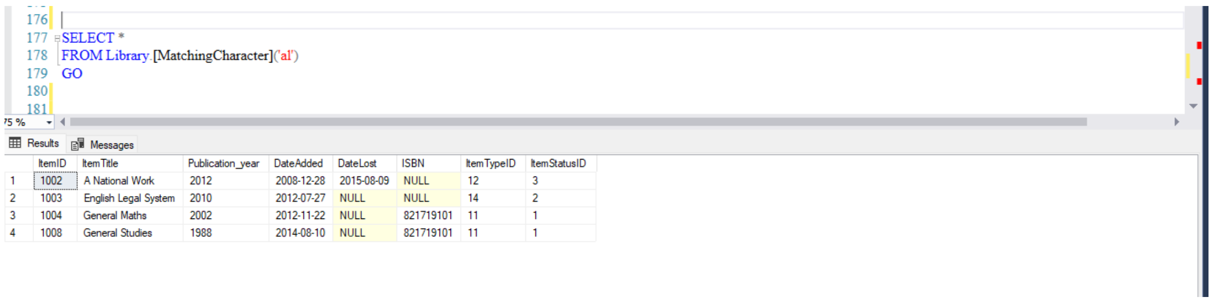
I went ahead and inserted values into my items table.



```sql
--Inserting values into Item table
INSERT INTO Library.Item
Values('A National Work', 2012, '2008-12-28', '2015-08-09', NULL,12, 3),
('English Legal System', 2010, '2012-07-27', NULL, NULL, 14, 2),
('General Maths', 2002,  '2012-11-22', NULL, '821719101', 11,  1),
('Brighter Life', 1967, '2014-03-27', '2016-04-13',NULL, 13, 4),
('Photography Scope', 2014, '2018-05-08', NULL, NULL,12, 1),
('Human Rights', 2000, '2020-01-01', NULL, NULL, 14, 1),
('General Studies', 1988,  '2014-08-10', NULL, '821719101', 11,  1);
Go

SELECT *
FROM Library.Item
GO
```

	ItemID	ItemTitle	Publication_year	DateAdded	DateLost	ISBN	ItemTypeID	ItemStatusID
1	1002	A National Work	2012	2008-12-28	2015-08-09	NULL	12	3
2	1003	English Legal System	2010	2012-07-27	NULL	NULL	14	2
3	1004	General Maths	2002	2012-11-22	NULL	821719101	11	1
4	1005	Brighter Life	1967	2014-03-27	2016-04-13	NULL	13	4
5	1006	Photography Scope	2014	2018-05-08	NULL	NULL	12	1
6	1007	Human Rights	2000	2020-01-01	NULL	NULL	14	1
7	1008	General Studies	1988	2014-08-10	NULL	821719101	11	1

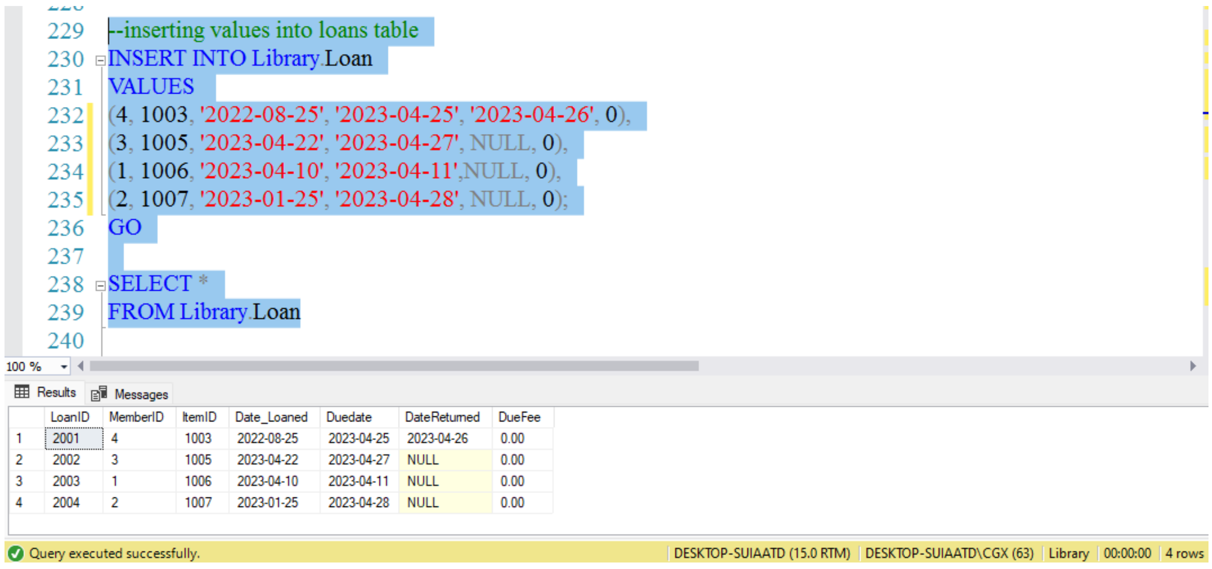
Demonstrated the use of the Library.MatchingCharacter function, using SELECT statement to get all rows where the sting ‘al’ appears in their Item title



```sql
SELECT *
FROM Library.[MatchingCharacter]('al')
GO
```

	ItemID	ItemTitle	Publication_year	DateAdded	DateLost	ISBN	ItemTypeID	ItemStatusID
1	1002	A National Work	2012	2008-12-28	2015-08-09	NULL	12	3
2	1003	English Legal System	2010	2012-07-27	NULL	NULL	14	2
3	1004	General Maths	2002	2012-11-22	NULL	821719101	11	1
4	1008	General Studies	1988	2014-08-10	NULL	821719101	11	1

Inserted the values in Loan table.

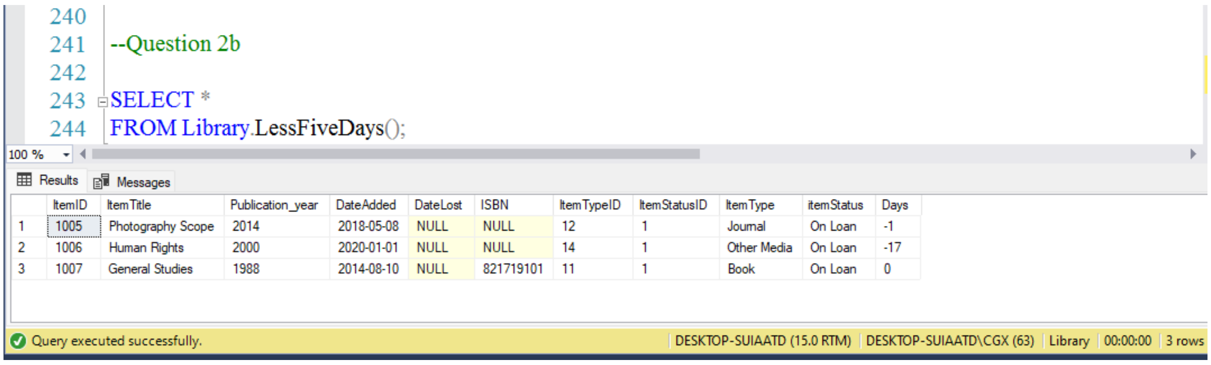


```sql
--inserting values into loans table
INSERT INTO Library.Loan
VALUES
(4, 1003, '2022-08-25', '2023-04-25', '2023-04-26', 0),
(3, 1005, '2023-04-22', '2023-04-27', NULL, 0),
(1, 1006, '2023-04-10', '2023-04-11',NULL, 0),
(2, 1007, '2023-01-25', '2023-04-28', NULL, 0);
GO

SELECT *
FROM Library.Loan
```

	LoanID	MemberID	ItemID	Date_Loaned	Duedate	DateReturned	DueFee
1	2001	4	1003	2022-08-25	2023-04-25	2023-04-26	0.00
2	2002	3	1005	2023-04-22	2023-04-27	NULL	0.00
3	2003	1	1006	2023-04-10	2023-04-11	NULL	0.00
4	2004	2	1007	2023-01-25	2023-04-28	NULL	0.00

```
Query executed successfully.    DESKTOP-SUIAATD (15.0 RTM) | DESKTOP-SUIAATD\CGX (63) | Library | 00:00:00 | 4 rows
```

Using the LessFiveDays function I was able to get retrieve details about items on loan with less than five days due.

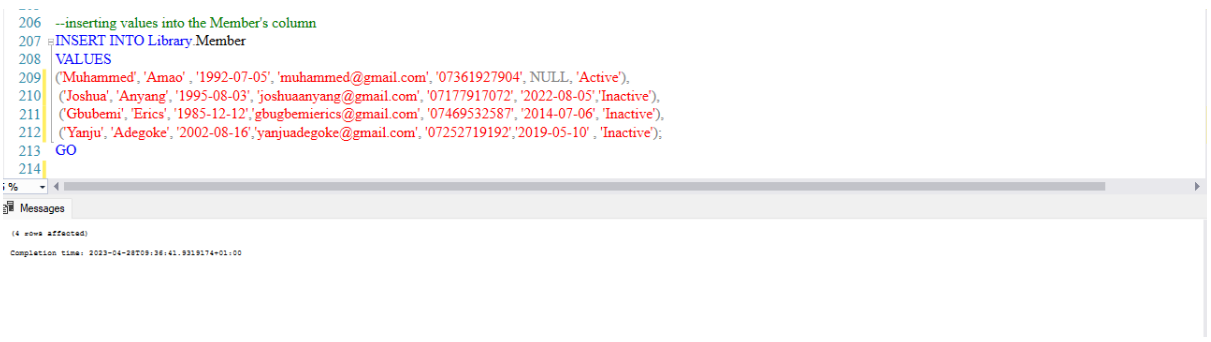


```sql
--Question 2b

SELECT *
FROM Library.LessFiveDays();
```

	ItemID	ItemTitle	Publication_year	DateAdded	DateLost	ISBN	ItemTypeID	ItemStatusID	ItemType	itemStatus	Days
1	1005	Photography Scope	2014	2018-05-08	NULL	NULL	12	1	Journal	On Loan	-1
2	1006	Human Rights	2000	2020-01-01	NULL	NULL	14	1	Other Media	On Loan	-17
3	1007	General Studies	1988	2014-08-10	NULL	821719101	11	1	Book	On Loan	0

```
Query executed successfully.   DESKTOP-SUIAATD (15.0 RTM) | DESKTOP-SUIAATD\CGX (63) | Library | 00:00:00 | 3 rows
```

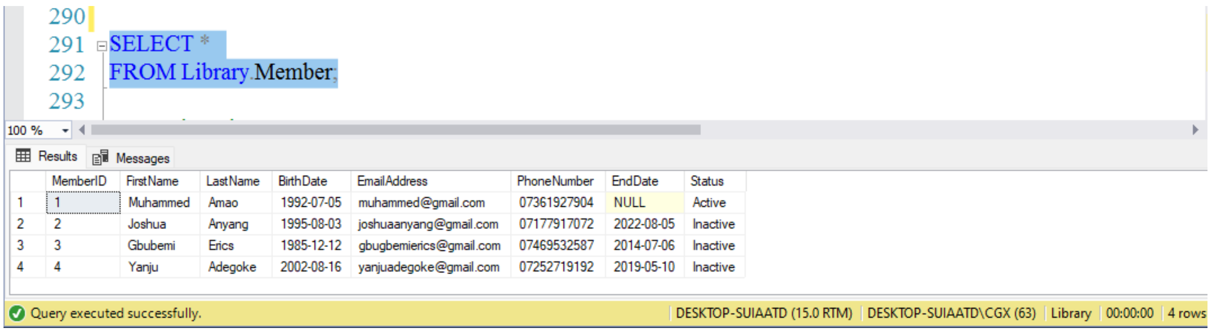
I inserted values into the Member table



```sql
--inserting values into the Member's column
INSERT INTO Library.Member
VALUES
('Muhammed', 'Amao' , '1992-07-05', 'muhammed@gmail.com', '07361927904', NULL, 'Active'),
 ('Joshua', 'Anyang', '1995-08-03', 'joshuaanyang@gmail.com', '07177917072', '2022-08-05','Inactive'),
 ('Gbubemi', 'Erics', '1985-12-12','gbugbemierics@gmail.com', '07469532587', '2014-07-06', 'Inactive'),
 ('Yanju', 'Adegoke', '2002-08-16','yanjuadegoke@gmail.com', '07252719192','2019-05-10' , 'Inactive');
GO
```

```
(4 rows affected)

Completion time: 2023-04-28T09:36:41.9319174+01:00
```

Demonstrating the procedure Library.InsertMember created in Question 2c using the Member table. Before executing the function the Member table had 4 rows.

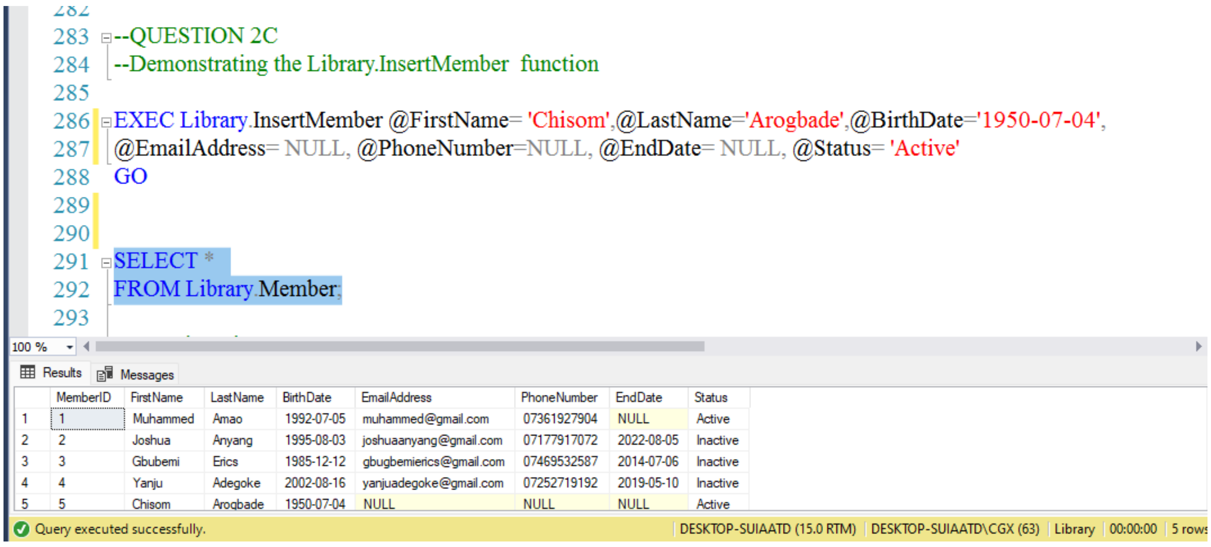


```sql
SELECT *
FROM Library.Member;
```

	MemberID	FirstName	LastName	BirthDate	EmailAddress	PhoneNumber	EndDate	Status
1	1	Muhammed	Amao	1992-07-05	muhammed@gmail.com	07361927904	NULL	Active
2	2	Joshua	Anyang	1995-08-03	joshuaanyang@gmail.com	07177917072	2022-08-05	Inactive
3	3	Gbubemi	Erics	1985-12-12	gbugbemierics@gmail.com	07469532587	2014-07-06	Inactive
4	4	Yanju	Adegoke	2002-08-16	yanjuadegoke@gmail.com	07252719192	2019-05-10	Inactive

```
Query executed successfully.   DESKTOP-SUIAATD (15.0 RTM) | DESKTOP-SUIAATD\CGX (63) | Library | 00:00:00 | 4 rows
```

Executing the Library.InsertMember function and assigning values to each column, I successfully inserted the 5th row to the table



```sql
--QUESTION 2C
--Demonstrating the Library.InsertMember  function

EXEC Library.InsertMember @FirstName= 'Chisom',@LastName='Arogbade',@BirthDate='1950-07-04',
@EmailAddress= NULL, @PhoneNumber=NULL, @EndDate= NULL, @Status= 'Active'
GO


SELECT *
FROM Library.Member;
```

100 %

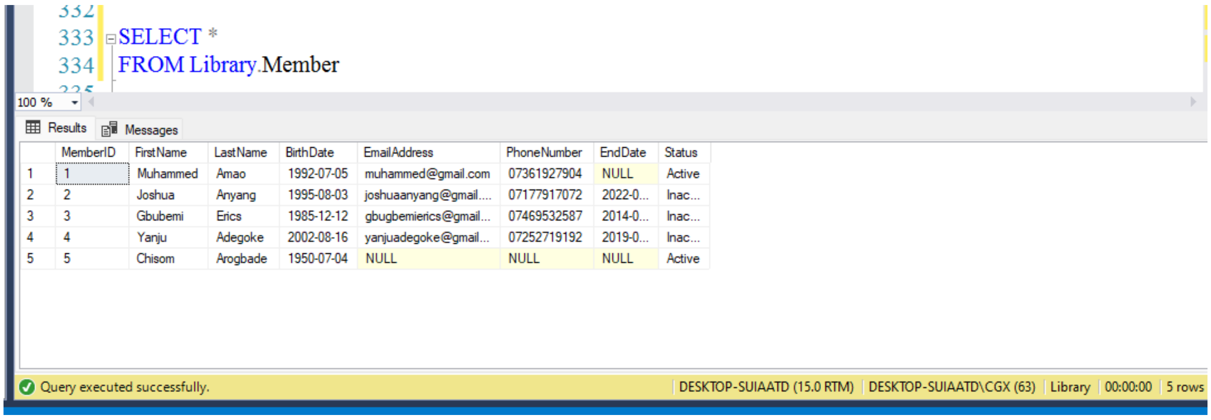
Results | Messages

	MemberID	FirstName	LastName	BirthDate	EmailAddress	PhoneNumber	EndDate	Status
1	1	Muhammed	Amao	1992-07-05	muhammed@gmail.com	07361927904	NULL	Active
2	2	Joshua	Anyang	1995-08-03	joshuaanyang@gmail.com	07177917072	2022-08-05	Inactive
3	3	Gbubemi	Erics	1985-12-12	gbugbemierics@gmail.com	07469532587	2014-07-06	Inactive
4	4	Yanju	Adegoke	2002-08-16	yanjuadegoke@gmail.com	07252719192	2019-05-10	Inactive
5	5	Chisom	Arogbade	1950-07-04	NULL	NULL	NULL	Active

```
Query executed successfully.    DESKTOP-SUIAATD (15.0 RTM) | DESKTOP-SUIAATD\CGX (63) | Library | 00:00:00 | 5 rows
```

I demonstrated the Library.UpdateMember function created in question 2d using the Member table. Before updating the table MemberID = 3, FirstName = ‘Gbugbemi’

Using the MemberID I was able to update and change the FirstName of MemberID 3 to ‘Francis’



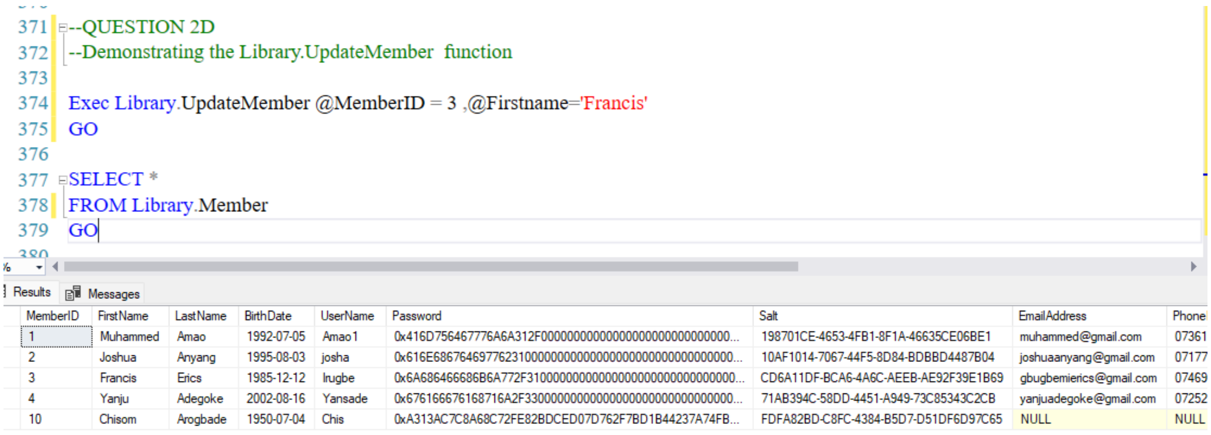
```sql
SELECT *
FROM Library.Member
```

100 %

Results | Messages

	MemberID	FirstName	LastName	BirthDate	EmailAddress	PhoneNumber	EndDate	Status
1	1	Muhammed	Amao	1992-07-05	muhammed@gmail.com	07361927904	NULL	Active
2	2	Joshua	Anyang	1995-08-03	joshuaanyang@gmail....	07177917072	2022-0...	Inac...
3	3	Gbubemi	Erics	1985-12-12	gbugbemierics@gmail...	07469532587	2014-0...	Inac...
4	4	Yanju	Adegoke	2002-08-16	yanjuadegoke@gmail...	07252719192	2019-0...	Inac...
5	5	Chisom	Arogbade	1950-07-04	NULL	NULL	NULL	Active

```
Query executed successfully.    DESKTOP-SUIAATD (15.0 RTM) | DESKTOP-SUIAATD\CGX (63) | Library | 00:00:00 | 5 rows
```

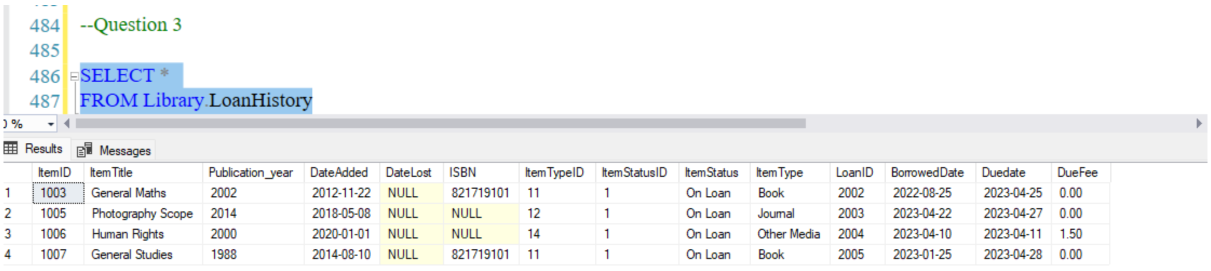


```
--QUESTION 2D
--Demonstrating the Library.UpdateMember  function

Exec Library.UpdateMember @MemberID = 3 ,@Firstname='Francis'
GO

SELECT *
FROM Library.Member
GO
```

Results | Messages

	MemberID	FirstName	LastName	BirthDate	UserName	Password	Salt	EmailAddress	Phone
	1	Muhammed	Amao	1992-07-05	Amao1	0x416D756467776A6A312F000000000000000000000000000000...	198701CE-4653-4FB1-8F1A-46635CE06BE1	muhammed@gmail.com	07361
	2	Joshua	Anyang	1995-08-03	josha	0x616E68676469776231000000000000000000000000000000000...	10AF1014-7067-44F5-8D84-BDBBD4487B04	joshuaanyang@gmail.com	07177
	3	Francis	Erics	1985-12-12	Irugbe	0x6A686466686B6A772F310000000000000000000000000000000...	CD6A11DF-BCA6-4A6C-AEEB-AE92F39E1B69	gbugbemierics@gmail.com	07469
	4	Yanju	Adegoke	2002-08-16	Yansade	0x676166676168716A2F330000000000000000000000000000000...	71AB394C-58DD-4451-A949-73C85343C2CB	yanjuadegoke@gmail.com	07252
	10	Chisom	Arogbade	1950-07-04	Chis	0xA313AC7C8A68C72FE82BDCED07D762F7BD1B44237A74FB...	FDFA82BD-C8FC-4384-B5D7-D51DF6D97C65	NULL	NULL

To demonstrate Question 3, I use the SELECT statement to select all columns from the view. Since it's a view, I decided to add the ItemStatus and ItemType columns as well.



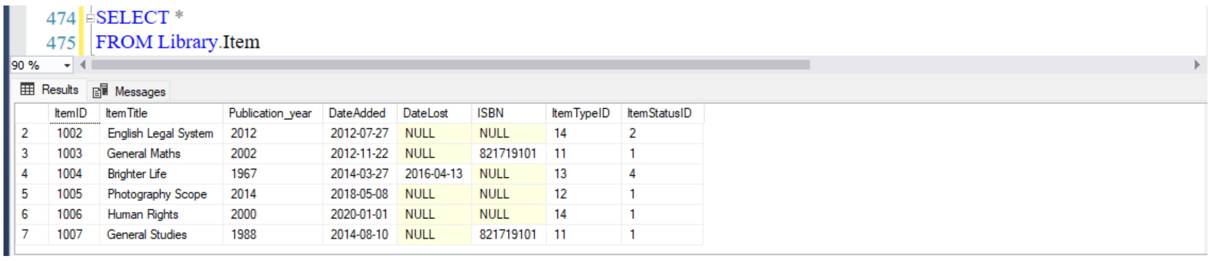
```
--Question 3

SELECT *
FROM Library.LoanHistory
```

Results | Messages

	ItemID	ItemTitle	Publication_year	DateAdded	DateLost	ISBN	ItemTypeID	ItemStatusID	ItemStatus	ItemType	LoanID	BorrowedDate	Duedate	DueFee
1	1003	General Maths	2002	2012-11-22	NULL	821719101	11	1	On Loan	Book	2002	2022-08-25	2023-04-25	0.00
2	1005	Photography Scope	2014	2018-05-08	NULL	NULL	12	1	On Loan	Journal	2003	2023-04-22	2023-04-27	0.00
3	1006	Human Rights	2000	2020-01-01	NULL	NULL	14	1	On Loan	Other Media	2004	2023-04-10	2023-04-11	1.50
4	1007	General Studies	1988	2014-08-10	NULL	821719101	11	1	On Loan	Book	2005	2023-01-25	2023-04-28	0.00

Demonstrating the Library.StatusTrigger that was created in question 4,

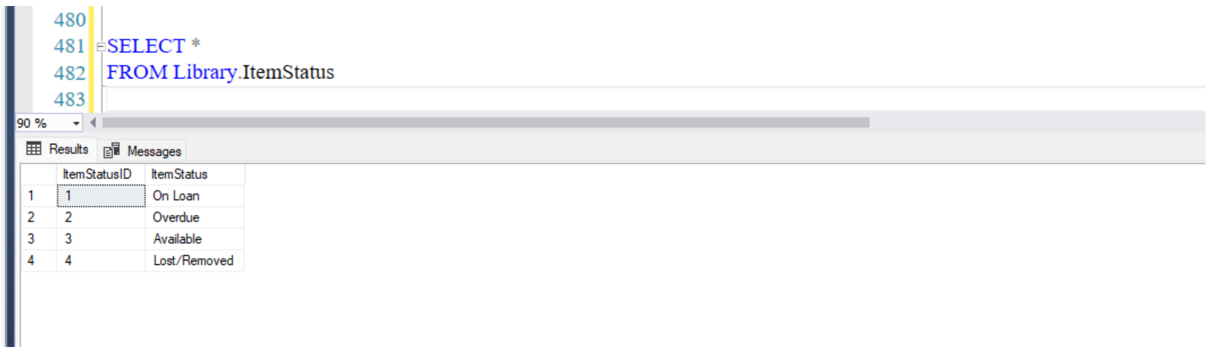
I retrieved all the columns in the Item table, ItemID 1003 has the ItemStatusID one(1), which represents 'On Loan' shown in the figure below.



```
SELECT *
FROM Library.Item
```

90 % | Results | Messages

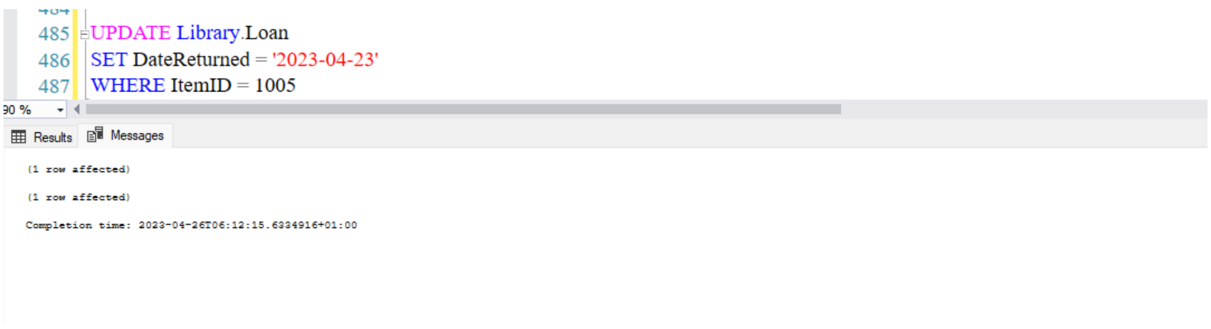
	ItemID	ItemTitle	Publication_year	DateAdded	DateLost	ISBN	ItemTypeID	ItemStatusID
2	1002	English Legal System	2012	2012-07-27	NULL	NULL	14	2
3	1003	General Maths	2002	2012-11-22	NULL	821719101	11	1
4	1004	Brighter Life	1967	2014-03-27	2016-04-13	NULL	13	4
5	1005	Photography Scope	2014	2018-05-08	NULL	NULL	12	1
6	1006	Human Rights	2000	2020-01-01	NULL	NULL	14	1
7	1007	General Studies	1988	2014-08-10	NULL	821719101	11	1



```sql
SELECT *
FROM Library.ItemStatus
```

	ItemStatusID	ItemStatus
1	1	On Loan
2	2	Overdue
3	3	Available
4	4	Lost/Removed

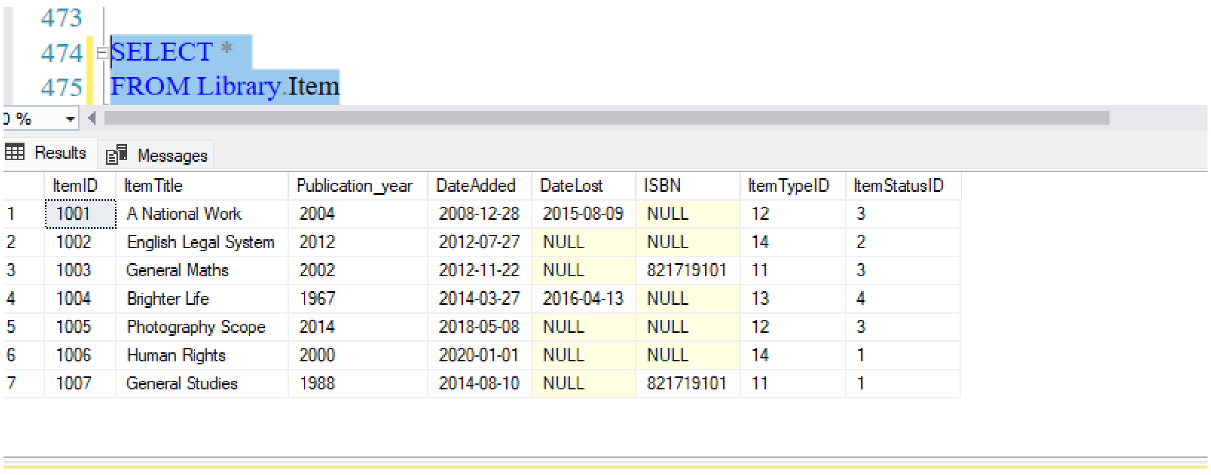
I updated the DateReturned column in the Loan table to a date older than the current date, where ItemID is 1005.



```sql
UPDATE Library.Loan
SET DateReturned = '2023-04-23'
WHERE ItemID = 1005
```

```
(1 row affected)

(1 row affected)

Completion time: 2023-04-26T06:12:15.6334916+01:00
```

Then retrieve all columns from Item table again, the ItemStatusID for ItemID 1003 successfully change to 3 which represents ‘Available’.

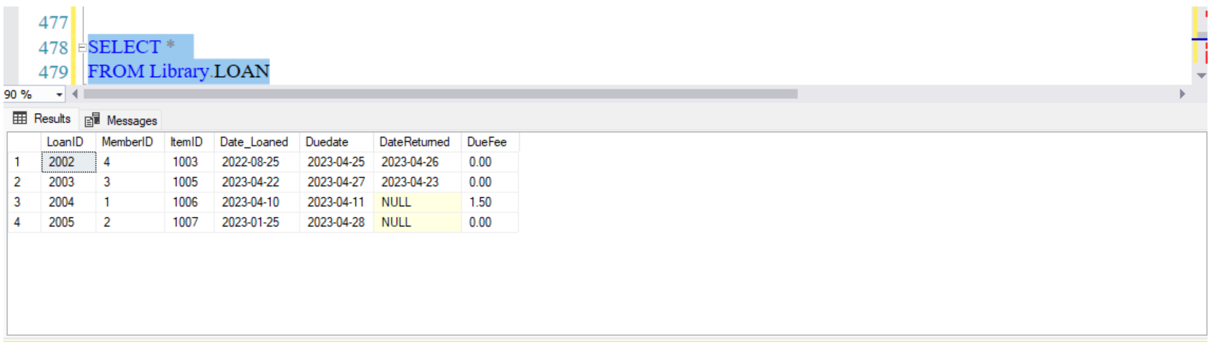


```sql
SELECT *
FROM Library.Item
```

	ItemID	ItemTitle	Publication_year	DateAdded	DateLost	ISBN	ItemTypeID	ItemStatusID
1	1001	A National Work	2004	2008-12-28	2015-08-09	NULL	12	3
2	1002	English Legal System	2012	2012-07-27	NULL	NULL	14	2
3	1003	General Maths	2002	2012-11-22	NULL	821719101	11	3
4	1004	Brighter Life	1967	2014-03-27	2016-04-13	NULL	13	4
5	1005	Photography Scope	2014	2018-05-08	NULL	NULL	12	3
6	1006	Human Rights	2000	2020-01-01	NULL	NULL	14	1
7	1007	General Studies	1988	2014-08-10	NULL	821719101	11	1

To demonstrate the NoOfLoans function created in question 5. The purpose of this function is to count the number of loans made on a specific date.

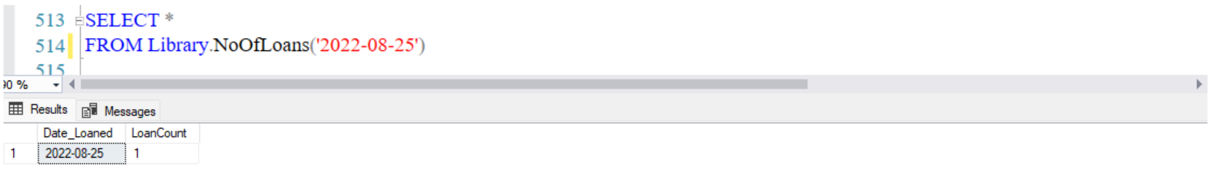
To show how this works, I retrieved the columns in the Loan table using the SELECT statement. In the Date_Loaned column there is only one row where loan was made on ‘2023-08-25’.



```sql
SELECT *
FROM Library.LOAN
```

	LoanID	MemberID	ItemID	Date_Loaned	Duedate	DateReturned	DueFee
1	2002	4	1003	2022-08-25	2023-04-25	2023-04-26	0.00
2	2003	3	1005	2023-04-22	2023-04-27	2023-04-23	0.00
3	2004	1	1006	2023-04-10	2023-04-11	NULL	1.50
4	2005	2	1007	2023-01-25	2023-04-28	NULL	0.00

Used the NoOfLoans function to count the number of time loans were made on ‘2022-08-05’ and the answer was 1.

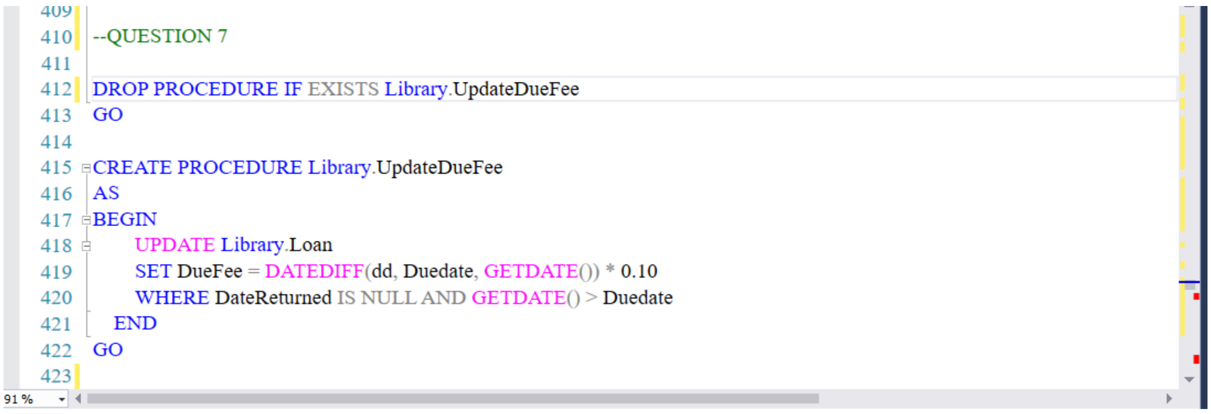


```sql
SELECT *
FROM Library.NoOfLoans('2022-08-25')
```

	Date_Loaned	LoanCount
1	2022-08-25	1

I created a procedure that updates the DueFee when the current date is greater than the due date.

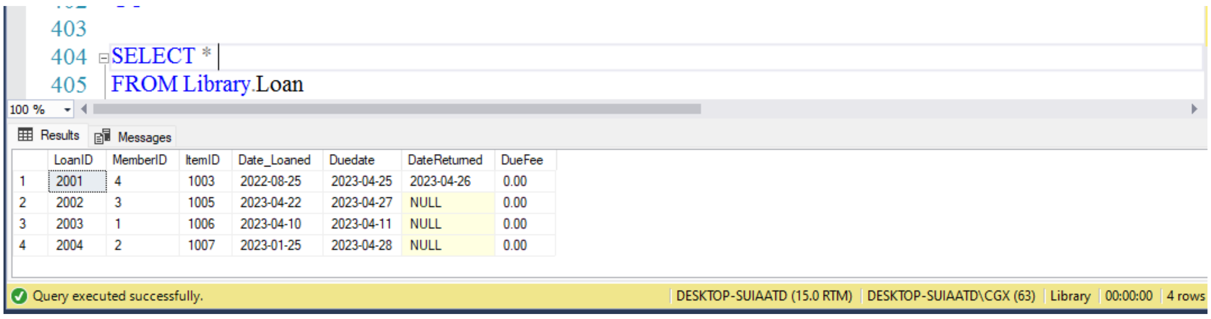
It multiplies the difference in days between the current date and due date by 0.10. I stored it in Library.UpdateDueFee.



```sql

--QUESTION 7

DROP PROCEDURE IF EXISTS Library.UpdateDueFee
GO

CREATE PROCEDURE Library.UpdateDueFee
AS
BEGIN
      UPDATE Library.Loan
      SET DueFee = DATEDIFF(dd, Duedate, GETDATE()) * 0.10
      WHERE DateReturned IS NULL AND GETDATE() > Duedate
   END
GO

```

Demonstrating the execution:-

All rows in DueFee column are currently 0.



```sql
SELECT *
FROM Library.Loan
```

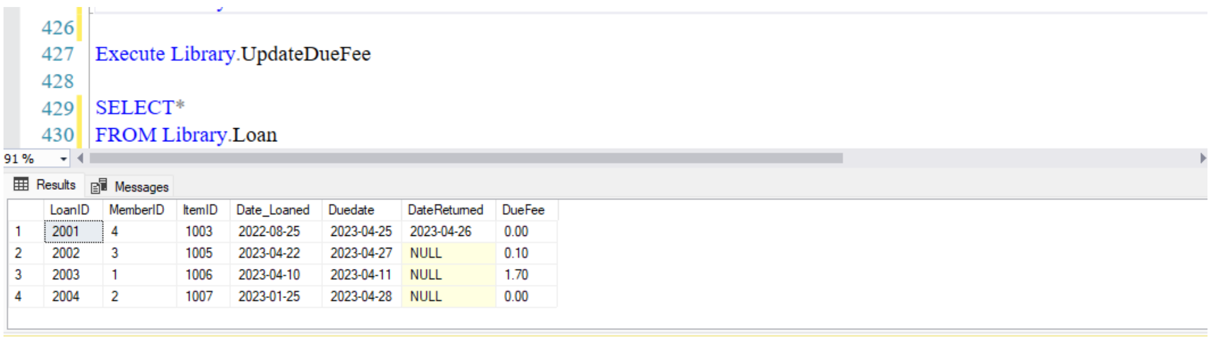
100 %

Results | Messages

	LoanID	MemberID	ItemID	Date_Loaned	Duedate	DateReturned	DueFee
1	2001	4	1003	2022-08-25	2023-04-25	2023-04-26	0.00
2	2002	3	1005	2023-04-22	2023-04-27	NULL	0.00
3	2003	1	1006	2023-04-10	2023-04-11	NULL	0.00
4	2004	2	1007	2023-01-25	2023-04-28	NULL	0.00

Query executed successfully. | DESKTOP-SUIAATD (15.0 RTM) | DESKTOP-SUIAATD\CGX (63) | Library | 00:00:00 | 4 rows

After executing the procedure the DueFee of 2002 changes to 0.10 and the due fee of 2003 changes to 1.70



```sql
Execute Library.UpdateDueFee

SELECT*
FROM Library.Loan
```

91 %

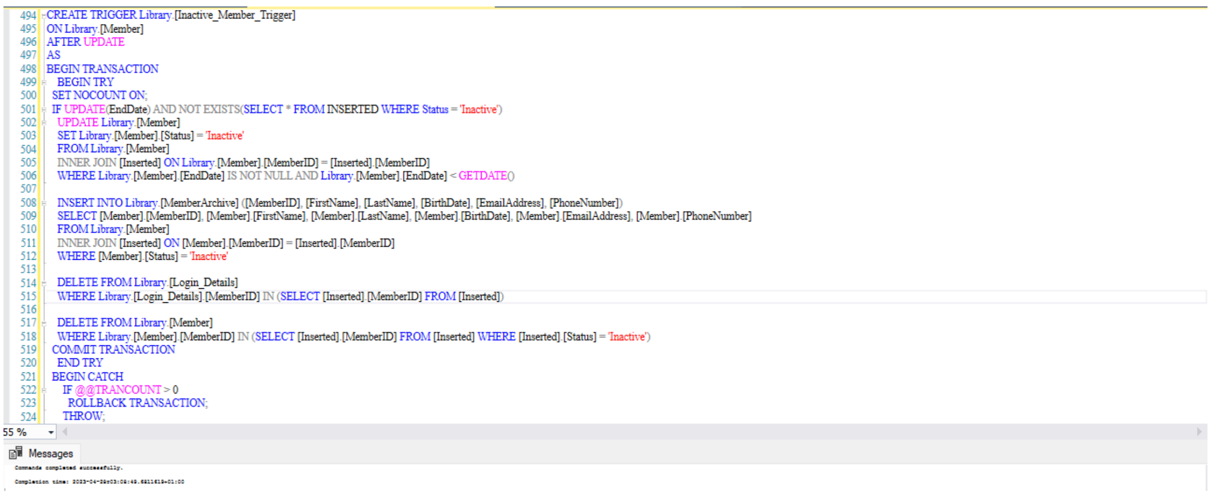
Results | Messages

	LoanID	MemberID	ItemID	Date_Loaned	Duedate	DateReturned	DueFee
1	2001	4	1003	2022-08-25	2023-04-25	2023-04-26	0.00
2	2002	3	1005	2023-04-22	2023-04-27	NULL	0.10
3	2003	1	1006	2023-04-10	2023-04-11	NULL	1.70
4	2004	2	1007	2023-01-25	2023-04-28	NULL	0.00

I created a trigger that sets the Status to Inactive when EndDate is updated and is greater than the current date in Member table.

It then takes the updated row and inserts it into the MemberArchive table to preserve the information of the member.

Joining on the MemberID it goes ahead and deletes the member's info from both the Login_Details and MemberID table where the MemberID is the same.



```sql
CREATE TRIGGER Library.[Inactive_Member_Trigger]
ON Library.[Member]
AFTER UPDATE
AS
BEGIN TRANSACTION
  BEGIN TRY
 SET NOCOUNT ON;
 IF UPDATE(EndDate) AND NOT EXISTS(SELECT * FROM INSERTED WHERE Status = 'Inactive')
  UPDATE Library.[Member]
  SET Library.[Member].[Status] = 'Inactive'
  FROM Library.[Member]
  INNER JOIN [Inserted] ON Library.[Member].[MemberID] = [Inserted].[MemberID]
  WHERE Library.[Member].[EndDate] IS NOT NULL AND Library.[Member].[EndDate] < GETDATE()

  INSERT INTO Library.[MemberArchive] ([MemberID], [FirstName], [LastName], [BirthDate], [EmailAddress], [PhoneNumber])
  SELECT [Member].[MemberID], [Member].[FirstName], [Member].[LastName], [Member].[BirthDate], [Member].[EmailAddress], [Member].[PhoneNumber]
  FROM Library.[Member]
  INNER JOIN [Inserted] ON [Member].[MemberID] = [Inserted].[MemberID]
  WHERE [Member].[Status] = 'Inactive'

  DELETE FROM Library.[Login_Details]
  WHERE Library.[Login_Details].[MemberID] IN (SELECT [Inserted].[MemberID] FROM [Inserted])

  DELETE FROM Library.[Member]
  WHERE Library.[Member].[MemberID] IN (SELECT [Inserted].[MemberID] FROM [Inserted] WHERE [Inserted].[Status] = 'Inactive')
 COMMIT TRANSACTION
  END TRY
 BEGIN CATCH
   IF @@TRANCOUNT > 0
    ROLLBACK TRANSACTION;
   THROW;
```

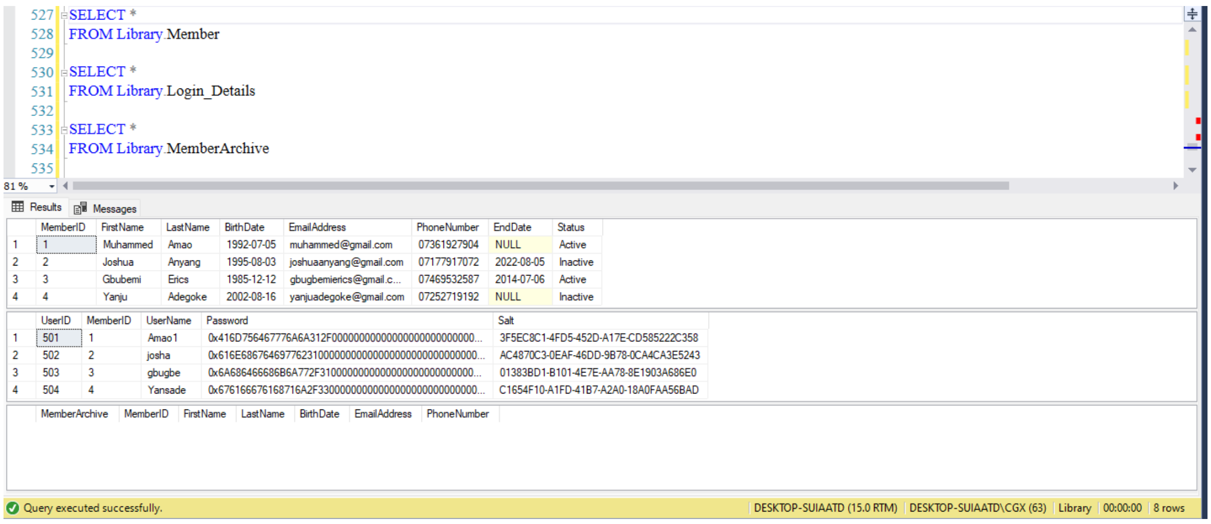
55 %

Messages

```
Commands completed successfully.

Completion time: 2023-04-28T03:08:49.6811619+01:00
```

Demonstrating the Library.Inactive_Member_Trigger; The figure below shows the data currently inserted into Member, Login_details, Member_Archive.

The Status for the Member ID one is currently 'Active'



```sql
SELECT *
FROM Library.Member

SELECT *
FROM Library.Login_Details

SELECT *
FROM Library.MemberArchive
```

Results | Messages

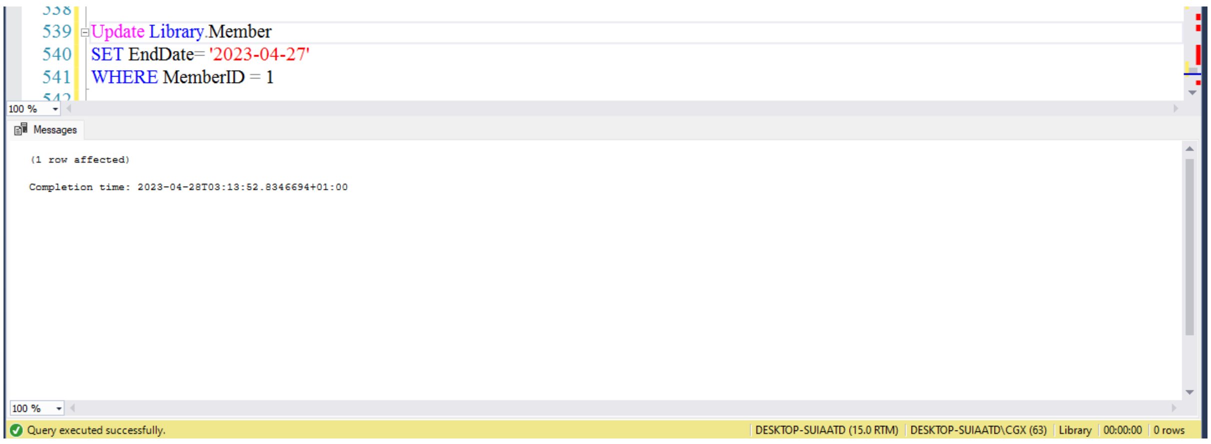
	MemberID	FirstName	LastName	BirthDate	EmailAddress	PhoneNumber	EndDate	Status
1	1	Muhammed	Amao	1992-07-05	muhammed@gmail.com	07361927904	NULL	Active
2	2	Joshua	Anyang	1995-08-03	joshuaanyang@gmail.com	07177917072	2022-08-05	Inactive
3	3	Gbubemi	Erics	1985-12-12	gbugbemierics@gmail.c...	07469532587	2014-07-06	Active
4	4	Yanju	Adegoke	2002-08-16	yanjuadegoke@gmail.com	07252719192	NULL	Inactive

	UserID	MemberID	UserName	Password	Salt
1	501	1	Amao1	0x416D756467776A6A312F000000000000000000000000000...	3F5EC8C1-4FD5-452D-A17E-CD585222C358
2	502	2	josha	0x616E686764697762310000000000000000000000000000000...	AC4870C3-0EAF-46DD-9B78-0CA4CA3E5243
3	503	3	gbugbe	0x6A686466686B6A772F3100000000000000000000000000...	01383BD1-B101-4E7E-AA78-8E1903A686E0
4	504	4	Yansade	0x676166676168716A2F330000000000000000000000000000...	C1654F10-A1FD-41B7-A2A0-18A0FAA56BAD

	MemberArchive	MemberID	FirstName	LastName	BirthDate	EmailAddress	PhoneNumber

Query executed successfully. | DESKTOP-SUIAATD (15.0 RTM) | DESKTOP-SUIAATD\CGX (63) | Library | 00:00:00 | 8 rows

To demonstrate the trigger I update MemberID that was Active EndDate



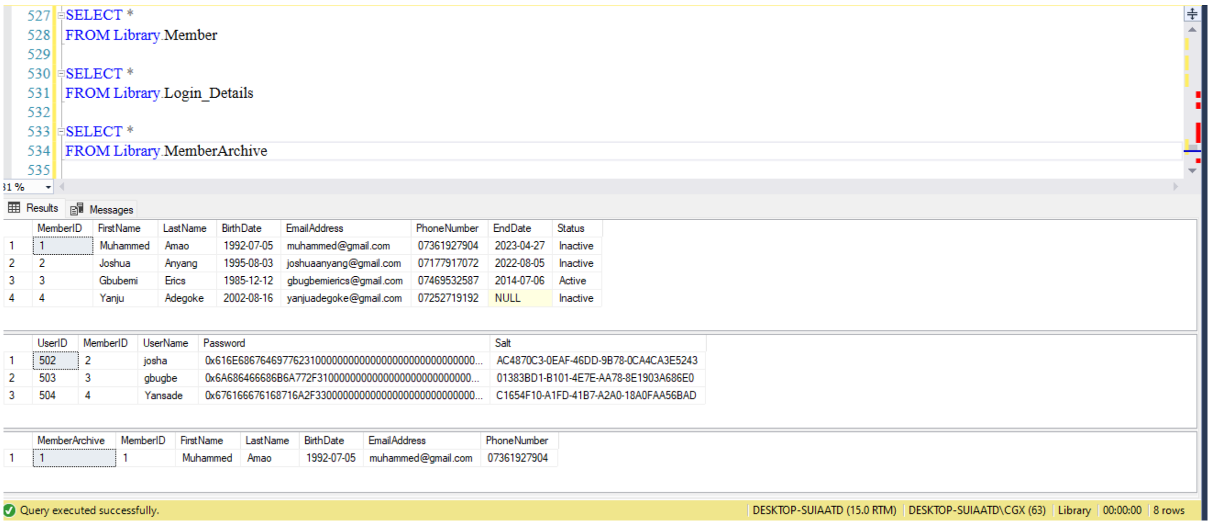
```sql
Update Library.Member
SET EndDate= '2023-04-27'
WHERE MemberID = 1
```

Messages

```
(1 row affected)

Completion time: 2023-04-28T03:13:52.8346694+01:00
```

Query executed successfully. | DESKTOP-SUIAATD (15.0 RTM) | DESKTOP-SUIAATD\CGX (63) | Library | 00:00:00 | 0 rows

It shows here that the MemberArchive Tavle has been updated with a row for the MemberID now inactive.



```sql
SELECT *
FROM Library.Member

SELECT *
FROM Library.Login_Details

SELECT *
FROM Library.MemberArchive
```

Results | Messages

	MemberID	FirstName	LastName	BirthDate	EmailAddress	PhoneNumber	EndDate	Status
1	1	Muhammed	Amao	1992-07-05	muhammed@gmail.com	07361927904	2023-04-27	Inactive
2	2	Joshua	Anyang	1995-08-03	joshuaanyang@gmail.com	07177917072	2022-08-05	Inactive
3	3	Gbubemi	Erics	1985-12-12	gbugbemierics@gmail.com	07469532587	2014-07-06	Active
4	4	Yanju	Adegoke	2002-08-16	yanjuadegoke@gmail.com	07252719192	NULL	Inactive

	UserID	MemberID	UserName	Password	Salt
1	502	2	josha	0x616E686764697762310000000000000000000000000000000...	AC4870C3-0EAF-46DD-9B78-0CA4CA3E5243
2	503	3	gbugbe	0x6A686466686B6A772F3100000000000000000000000000...	01383BD1-B101-4E7E-AA78-8E1903A686E0
3	504	4	Yansade	0x676166676168716A2F330000000000000000000000000000...	C1654F10-A1FD-41B7-A2A0-18A0FAA56BAD

	MemberArchive	MemberID	FirstName	LastName	BirthDate	EmailAddress	PhoneNumber
1	1	1	Muhammed	Amao	1992-07-05	muhammed@gmail.com	07361927904

Query executed successfully. | DESKTOP-SUIAATD (15.0 RTM) | DESKTOP-SUIAATD\CGX (63) | Library | 00:00:00 | 8 rows

ADVICE AND GUIDANCE

Data Integrity- This is ensuring the accuracy, completeness and consistency of the database. In order to accomplish this while designing this database, I did the following:

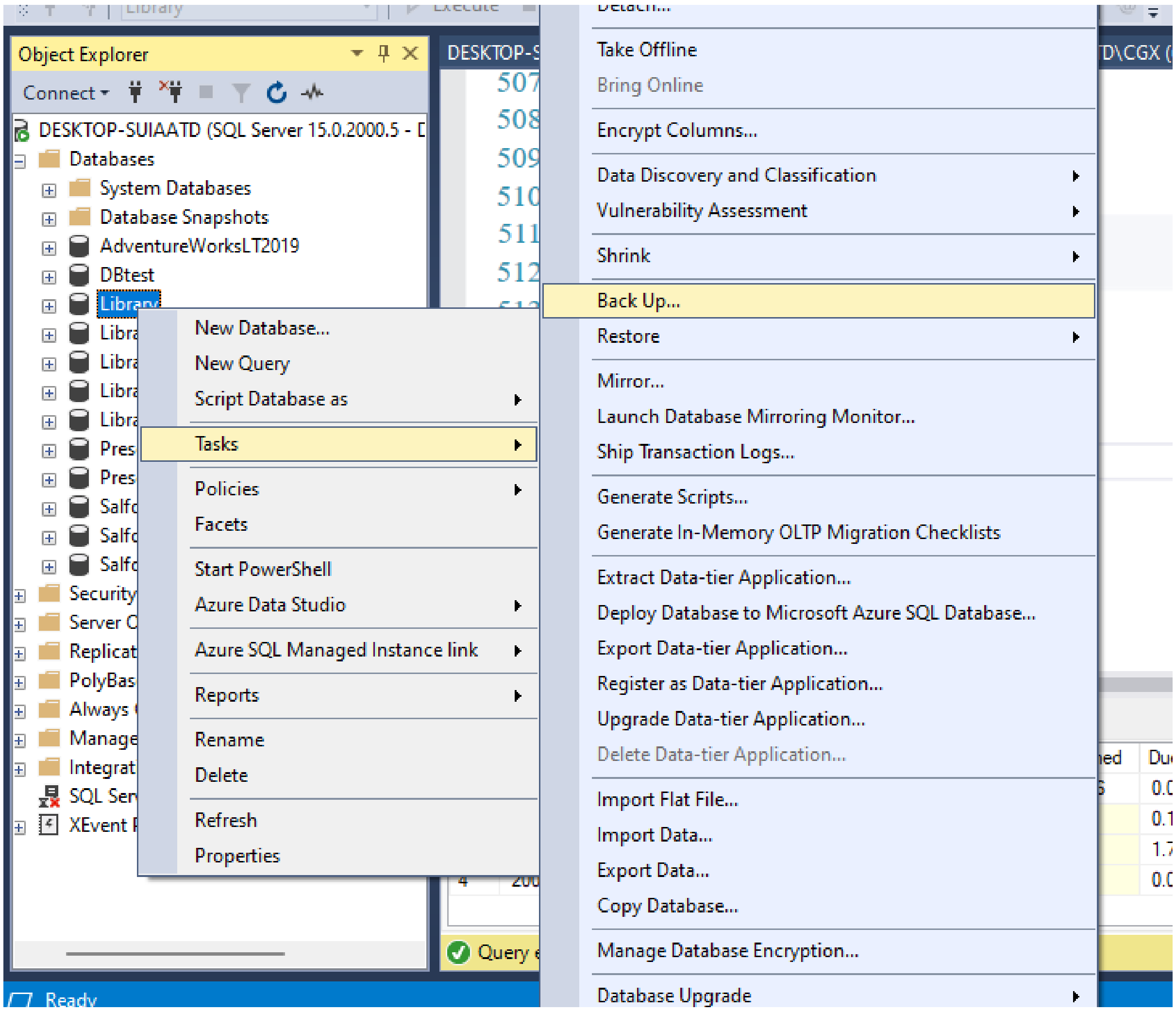
- Every entity in the database has a Primary Key that uniquely identifies it, this ensures that the data is not recorded multiple times.
- In the PhoneNumber column of the Member Table, I added a constraint that checks that every phone number is 11 digit long.
- In the EmailAddress column of the Member Table, added a CHECK constraint that helps to enforce data integrity, by ensuring that email addresses adhere to the basic form required for a valid email address.
- Making sure I assigned the right datatype to each column.
- Accurately assigning foreign keys forming relationships to each column.

Data Concurrency:- I made sure the database procedures and functions has the ACID properties. For instance in the Library.Inactive_Member_Trigger a concurrent transaction occurs. I make sure the trigger commits from the begin of the transaction to the end of the transaction. If not the transaction would not run.

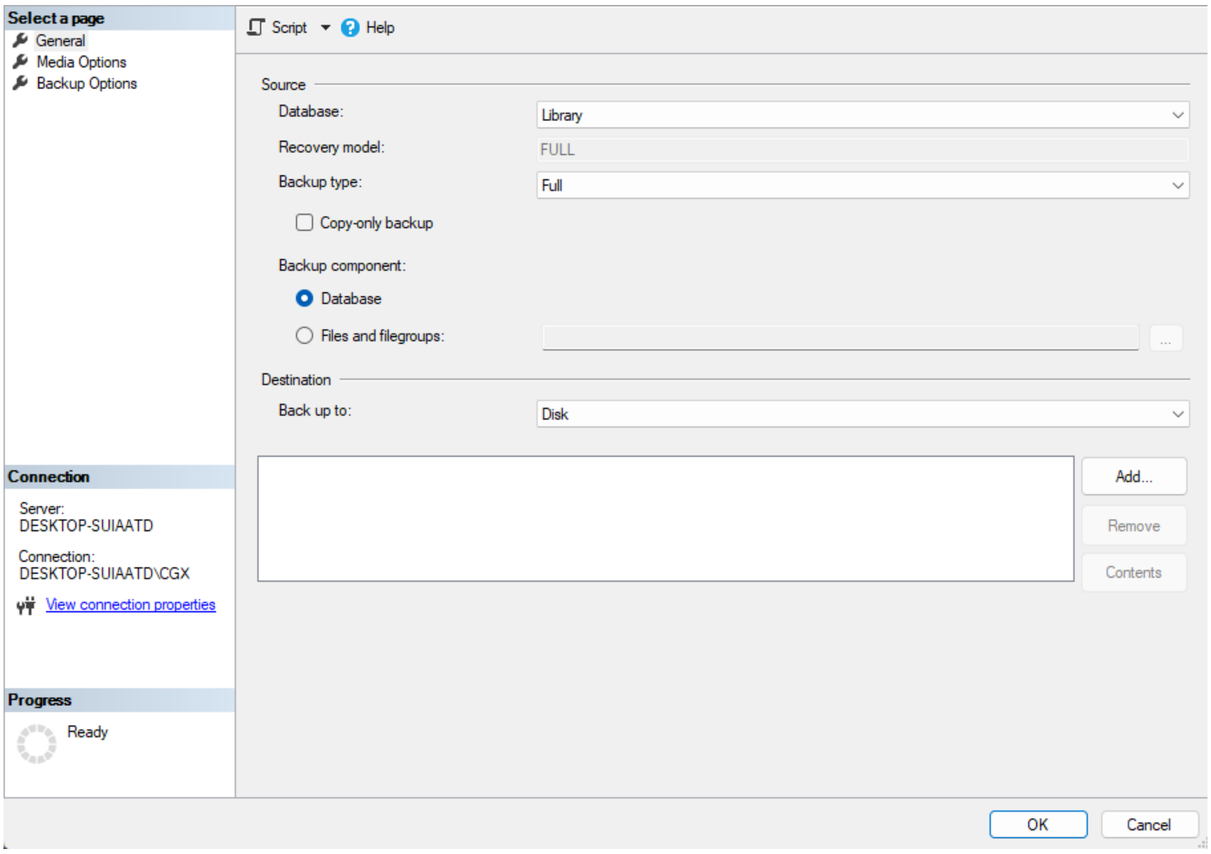
Data Security:- In a case of data leakage, password hacking can be used to identify what the orignal password. I use password salted hash to make it difficult.

Backup and Recovery:

I created a new folder in my C: - and named it Library back up. I right clicked on the Library database, selected Tasks and then Back Up.



In the BackUp database, I clicked on Add to select the folder that I created in my C drive and my back up type to full


Select a page
- General
- Media Options
- Backup Options

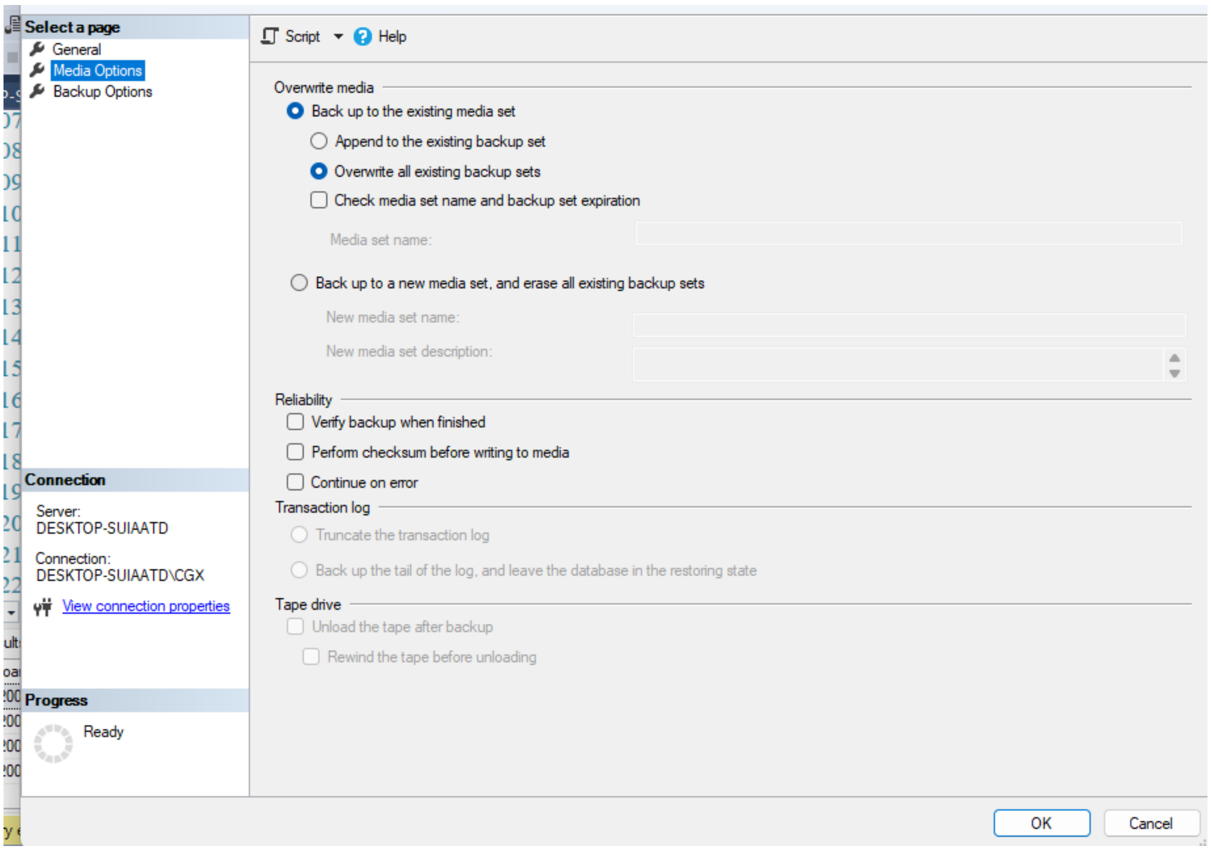
Source
Database: Library
Recovery model: FULL
Backup type: Full
☐ Copy-only backup
Backup component:
● Database
○ Files and filegroups:

Destination
Back up to: Disk

Connection
Server:
DESKTOP-SUIAATD
Connection:
DESKTOP-SUIAATD\CGX
View connection properties

Progress
Ready

Clicked on Media option and then in the Overwrite media, I picked **Overwrite all existing backup sets,** to empty the back up sets


Select a page
- General
- Media Options
- Backup Options

Overwrite media
● Back up to the existing media set
○ Append to the existing backup set
● Overwrite all existing backup sets
☐ Check media set name and backup set expiration
Media set name:
○ Back up to a new media set, and erase all existing backup sets
New media set name:
New media set description:

Reliability
☐ Verify backup when finished
☐ Perform checksum before writing to media
☐ Continue on error

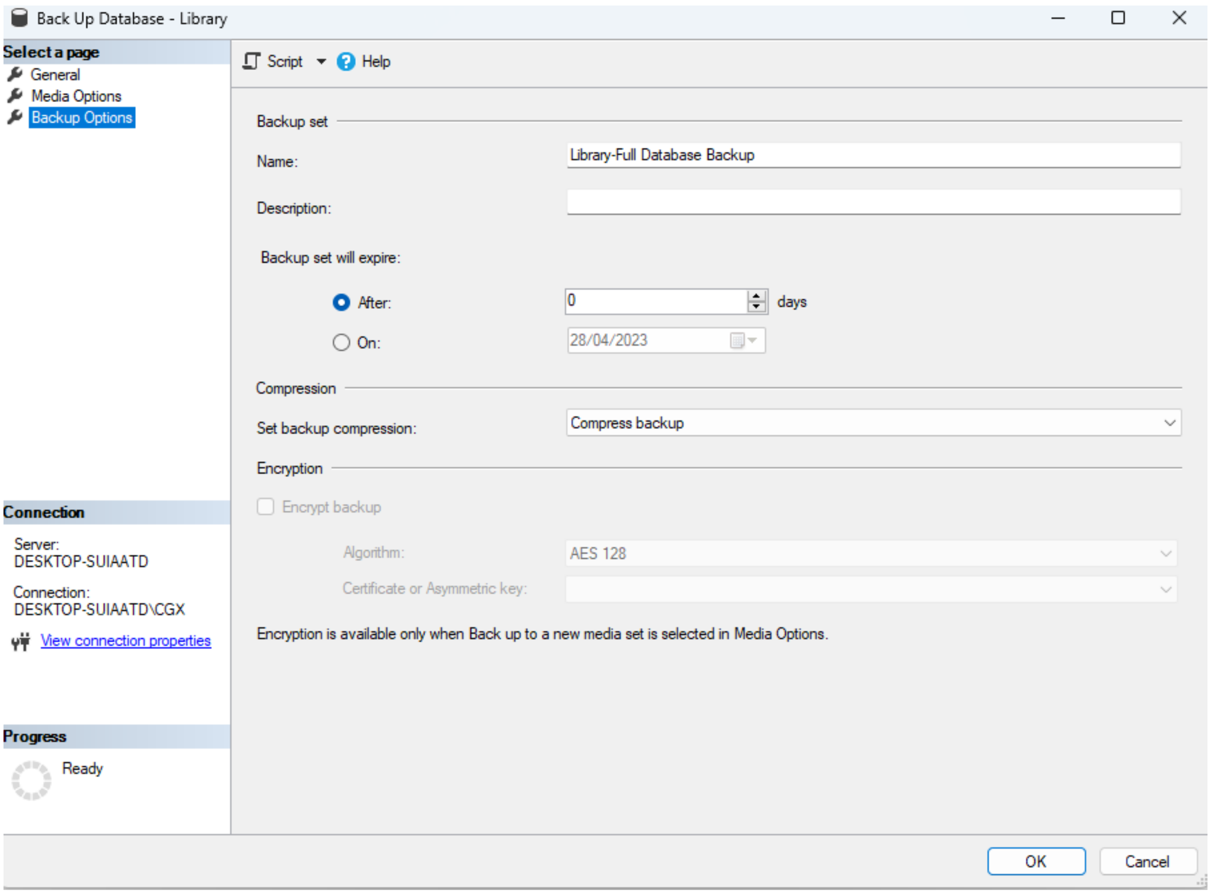
Transaction log
○ Truncate the transaction log
○ Back up the tail of the log, and leave the database in the restoring state

Tape drive
☐ Unload the tape after backup
☐ Rewind the tape before unloading

Connection
Server:
DESKTOP-SUIAATD
Connection:
DESKTOP-SUIAATD\CGX
View connection properties

Progress
Ready

In the back up option, I chose to Compress Backup.


Back Up Database - Library

Select a page
- General
- Media Options
- Backup Options

Script ▾ Help

Backup set

Name: Library-Full Database Backup

Description:

Backup set will expire:

● After: 0 days

○ On: 28/04/2023

Compression

Set backup compression: Compress backup

Encryption

☐ Encrypt backup

Algorithm: AES 128

Certificate or Asymmetric key:

Encryption is available only when Back up to a new media set is selected in Media Options.

Connection

Server:
DESKTOP-SUIAATD

Connection:
DESKTOP-SUIAATD\CGX

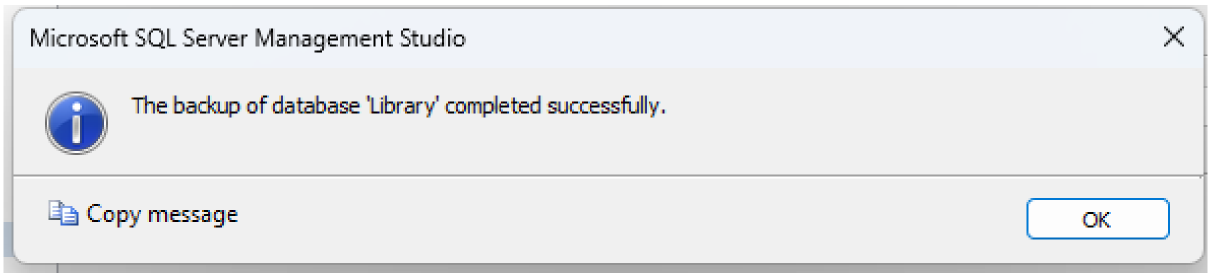
View connection properties

Progress

Ready

OK Cancel

Clicked on ok and my backup was successful.


Microsoft SQL Server Management Studio

The backup of database 'Library' completed successfully.

Copy message OK

CONCLUSION

In conclusion, I was able to design a database that is flexible, efficient, secured. Triggers, stored procedures, views and functions are efficient because the all ACID tested.