Day-9

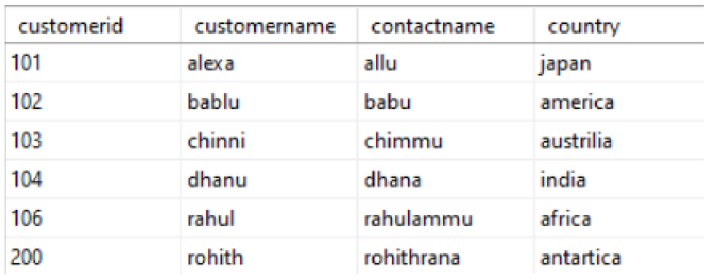
Date: - 17-08-2024

SQL Language

DQL-DATA QUERY LANGUAGE:

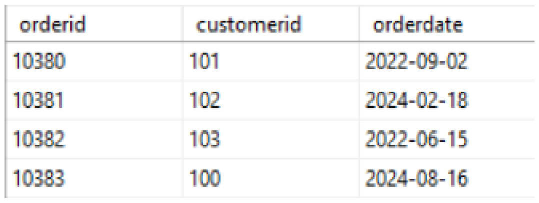
DQL statements are used for performing queries on the data within schema objects.

Assignment 1: Craft a query using an INNER JOIN to combine 'orders' and 'customers' tables for customers in a specified region, and a LEFT JOIN to display all customers including those without orders.



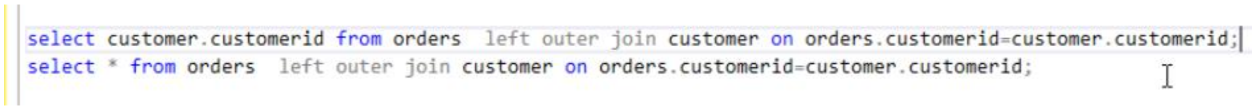
customerid	customername	contactname	country
101	alexa	allu	japan
102	bablu	babu	america
103	chinni	chimmu	austrilia
104	dhanu	dhana	india
106	rahul	rahulammu	africa
200	rohith	rohithrana	antartica

Fig-1: Customer table

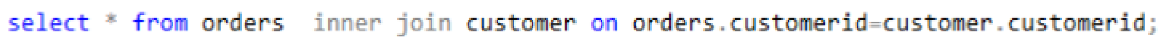


orderid	customerid	orderdate
10380	101	2022-09-02
10381	102	2024-02-18
10382	103	2022-06-15
10383	100	2024-08-16

Fig-2: Orders table

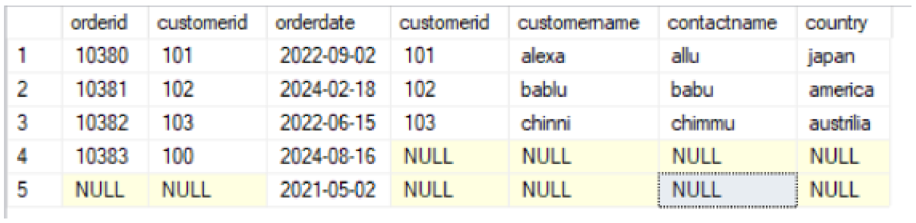


```
select customer.customerid from orders  left outer join customer on orders.customerid=customer.customerid;
select * from orders  left outer join customer on orders.customerid=customer.customerid;
```

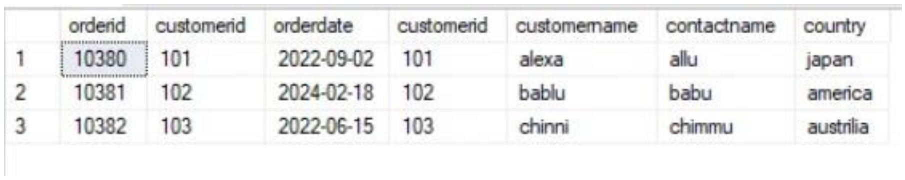


```
select * from orders  inner join customer on orders.customerid=customer.customerid;
```

Fig-3: SQL Query



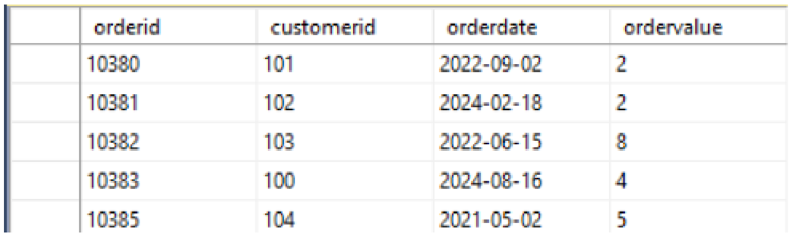
	orderid	customerid	orderdate	customerid	customername	contactname	country
1	10380	101	2022-09-02	101	alexa	allu	japan
2	10381	102	2024-02-18	102	bablu	babu	america
3	10382	103	2022-06-15	103	chinni	chimmu	austrilia
4	10383	100	2024-08-16	NULL	NULL	NULL	NULL
5	NULL	NULL	2021-05-02	NULL	NULL	NULL	NULL



	orderid	customerid	orderdate	customerid	customername	contactname	country
1	10380	101	2022-09-02	101	alexa	allu	japan
2	10381	102	2024-02-18	102	bablu	babu	america
3	10382	103	2022-06-15	103	chinni	chimmu	austrilia

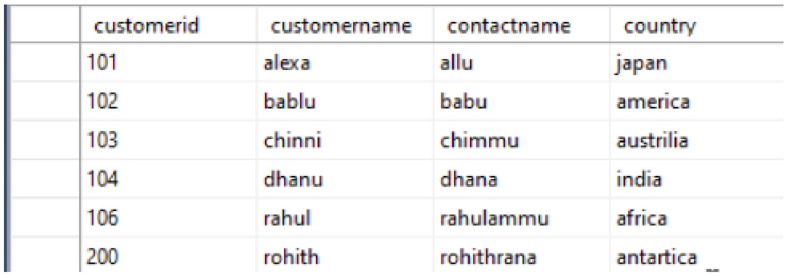
Fig-4: output

Assignment 2: Utilize a subquery to find customers who have placed orders above the average order value, and write a UNION query to combine two SELECT statements with the same number of columns.



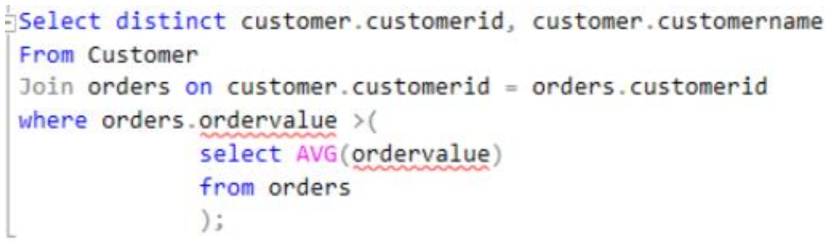
	orderid	customerid	orderdate	ordervalue
	10380	101	2022-09-02	2
	10381	102	2024-02-18	2
	10382	103	2022-06-15	8
	10383	100	2024-08-16	4
	10385	104	2021-05-02	5

Fig-1: Orders Table



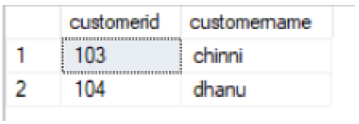
	customerid	customername	contactname	country
	101	alexa	allu	japan
	102	bablu	babu	america
	103	chinni	chimmu	austrilia
	104	dhanu	dhana	india
	106	rahul	rahulammu	africa
	200	rohith	rohithrana	antartica

Fig-2: Customer Table



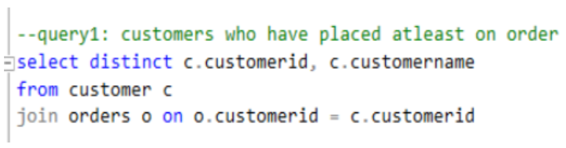
```
Select distinct customer.customerid, customer.customername
From Customer
Join orders on customer.customerid = orders.customerid
where orders.ordervalue >(
             select AVG(ordervalue)
             from orders
             );
```

Fig-3: SQL query for order value above the avg order value

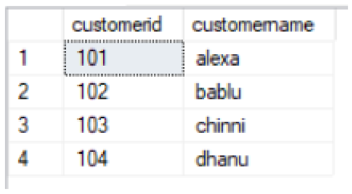


	customerid	customername
1	103	chinni
2	104	dhanu

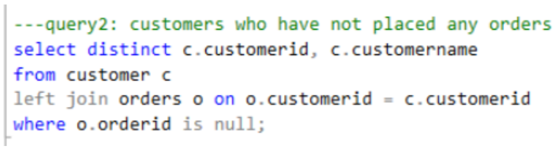
Fig-4: output for fig3



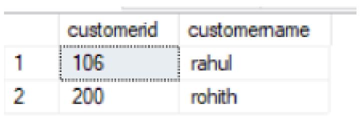
```
--query1: customers who have placed atleast on order
select distinct c.customerid, c.customername
from customer c
join orders o on o.customerid = c.customerid
```



	customerid	customername
1	101	alexa
2	102	bablu
3	103	chinni
4	104	dhanu

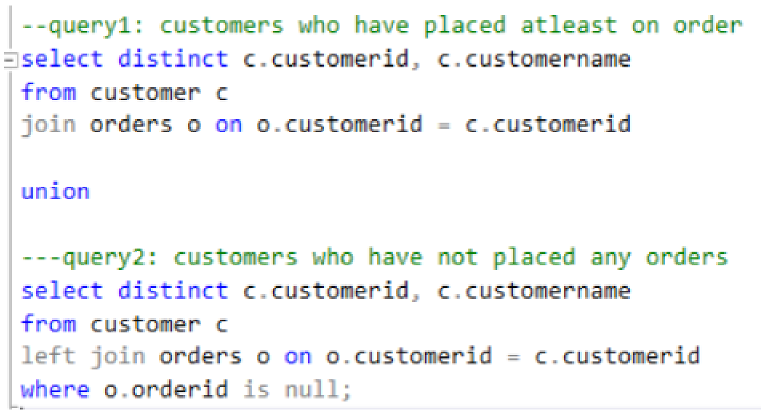


```
---query2: customers who have not placed any orders
select distinct c.customerid, c.customername
from customer c
left join orders o on o.customerid = c.customerid
where o.orderid is null;
```

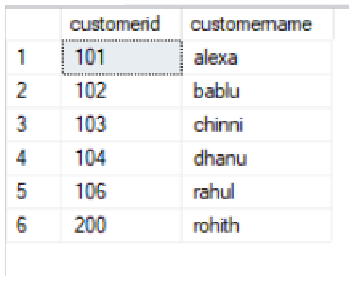


	customerid	customername
1	106	rahul
2	200	rohith

UNION



```
--query1: customers who have placed atleast on order
select distinct c.customerid, c.customername
from customer c
join orders o on o.customerid = c.customerid

union

---query2: customers who have not placed any orders
select distinct c.customerid, c.customername
from customer c
left join orders o on o.customerid = c.customerid
where o.orderid is null;
```



	customerid	customername
1	101	alexa
2	102	bablu
3	103	chinni
4	104	dhanu
5	106	rahul
6	200	rohith

Fig-5: Using Union

Basic Data Retrieval statements:

1. Select:

In SQL Server, the SELECT statement is used to retrieve data from a database.

To retrieve all columns from a table: **SELECT * FROM TableName;**

To retrieve only certain columns: **SELECT Column1, Column2 FROM TableName;**

2. Where Clauses and Filtering Data:

In SQL Server, the WHERE clause is used to filter records and return only those that meet specified criteria.

The general syntax for the WHERE clause is: **SELECT column1, column2, ... FROM table_name**

WHERE condition;

To filter rows based on exact matches: **SELECT * FROM Employees WHERE Department = 'Sales';**

3. Order by, Group by and Having Clauses:

Order by: The ORDER BY clause is used to sort the result set of a query based on one or more columns. You can specify the sort direction as ascending (ASC) or descending (DESC).

Basic Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1 [ASC|DESC], column2 [ASC|DESC], ...;
```

Example-1: Sort by a Single Column

```
SELECT * FROM Employees
ORDER BY LastName ASC;
```

Group By: The GROUP BY clause is used to group rows that have the same values in specified columns into summary rows. Often, it is used with aggregate functions like COUNT, SUM, AVG, MIN, and MAX.

Basic Syntax

```
SELECT column1, aggregate_function(column2)
FROM table_name
GROUP BY column1;
```

Example-1: Group and Aggregate

```
SELECT Department, COUNT(*) AS NumberOfEmployees
FROM Employees
GROUP BY Department;
```

Having Clause:

The HAVING clause is used to filter groups created by the GROUP BY clause. It is similar to the WHERE clause but is applied after grouping.

Basic Syntax

```
SELECT column1, aggregate_function(column2)
FROM table_name
GROUP BY column1
HAVING aggregate_function(column2) condition;
```

Example-1: Filter Groups

```
SELECT Department, COUNT(*) AS NumberOfEmployees
FROM Employees
GROUP BY Department
```

HAVING COUNT(*) > 10;

Combining ORDER BY, GROUP BY, and HAVING:

Example:

SELECT Department, AVG(Salary) AS AverageSalary, COUNT(*) AS NumberOfEmployees

FROM Employees

GROUP BY Department

HAVING COUNT(*) > 10 ORDER BY AverageSalary DESC;

Advanced Data Retrieval statements:

Joins: Joins combine rows from two or more tables based on related columns.

In SQL Server, JOIN operations are used to retrieve data from multiple tables based on related columns.

Types of Joins:

- **Inner Join:** An INNER JOIN returns only the rows where there is a match in both tables. If there is no match, the row is not included in the result set.
  Syntax: **SELECT columns**
  ```
  SELECT columns
      FROM table1
      INNER JOIN table2
      ON table1.column = table2.column;
  ```
- **Left Join:** A LEFT JOIN (or LEFT OUTER JOIN) returns all rows from the left table and the matched rows from the right table. If there is no match, the result is NULL for columns from the right table.
  Syntax:
  ```
  SELECT columns
      FROM table1
      LEFT JOIN table2
      ON table1.column = table2.column;
  ```
- **Right Join:** A RIGHT JOIN (or RIGHT OUTER JOIN) returns all rows from the right table and the matched rows from the left table. If there is no match, the result is NULL for columns from the left table.
  Syntax:
  ```
  SELECT columns
      FROM table1
      RIGHT JOIN table2
      ON table1.column = table2.column;
  ```
- **Full Join:** A FULL JOIN (or FULL OUTER JOIN) returns all rows when there is a match in either the left or right table. If there is no match, the result is NULL for columns from the table without a match.

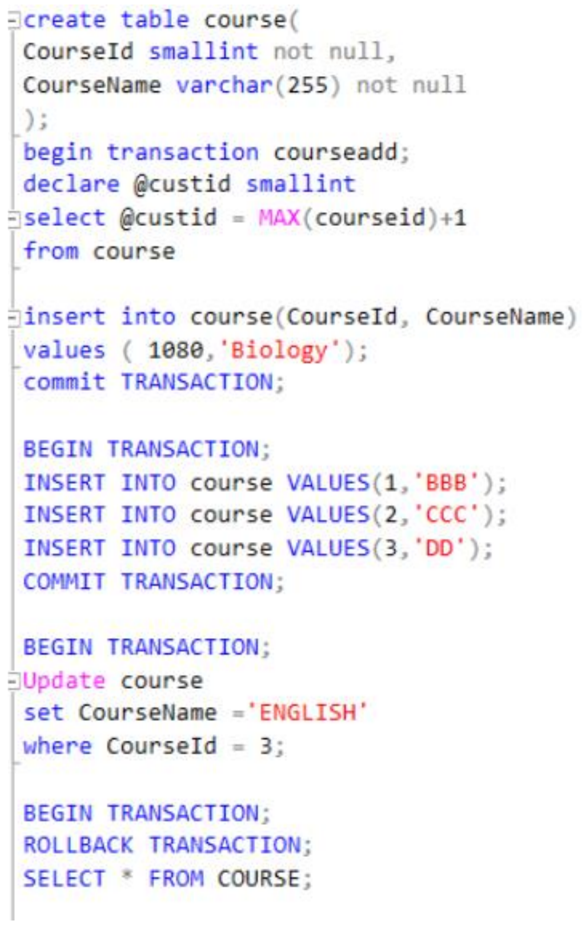
Syntax: **SELECT columns**
FROM table1
FULL JOIN table2
ON table1.column = table2.column;

- **Cross Join:** A CROSS JOIN returns the Cartesian product of both tables. It combines each row from the first table with every row from the second table. It's useful for generating all possible combinations of rows.
  Syntax: **SELECT columns**
  FROM table1
  CROSS JOIN table2;

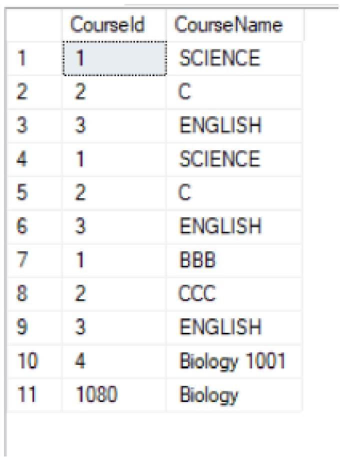
Rollback SQL: ROLLBACK is used to undo changes made during a transaction.

Assignment 3:

Compose SQL statements to BEGIN a transaction, INSERT a new record into the 'Courseadd' table, COMMIT the transaction, then UPDATE the 'Course' table, and ROLLBACK the transaction. in SQL server

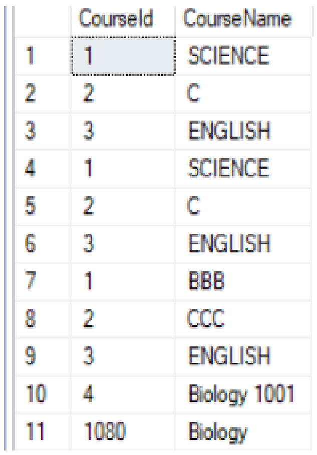


```
create table course(
CourseId smallint not null,
CourseName varchar(255) not null
);
begin transaction courseadd;
declare @custid smallint
select @custid = MAX(courseid)+1
from course

insert into course(CourseId, CourseName)
values ( 1080,'Biology');
commit TRANSACTION;

BEGIN TRANSACTION;
INSERT INTO course VALUES(1,'BBB');
INSERT INTO course VALUES(2,'CCC');
INSERT INTO course VALUES(3,'DD');
COMMIT TRANSACTION;

BEGIN TRANSACTION;
Update course
set CourseName ='ENGLISH'
where CourseId = 3;

BEGIN TRANSACTION;
ROLLBACK TRANSACTION;
SELECT * FROM COURSE;
```



	CourseId	CourseName
1	1	SCIENCE
2	2	C
3	3	ENGLISH
4	1	SCIENCE
5	2	C
6	3	ENGLISH
7	1	BBB
8	2	CCC
9	3	ENGLISH
10	4	Biology 1001
11	1080	Biology

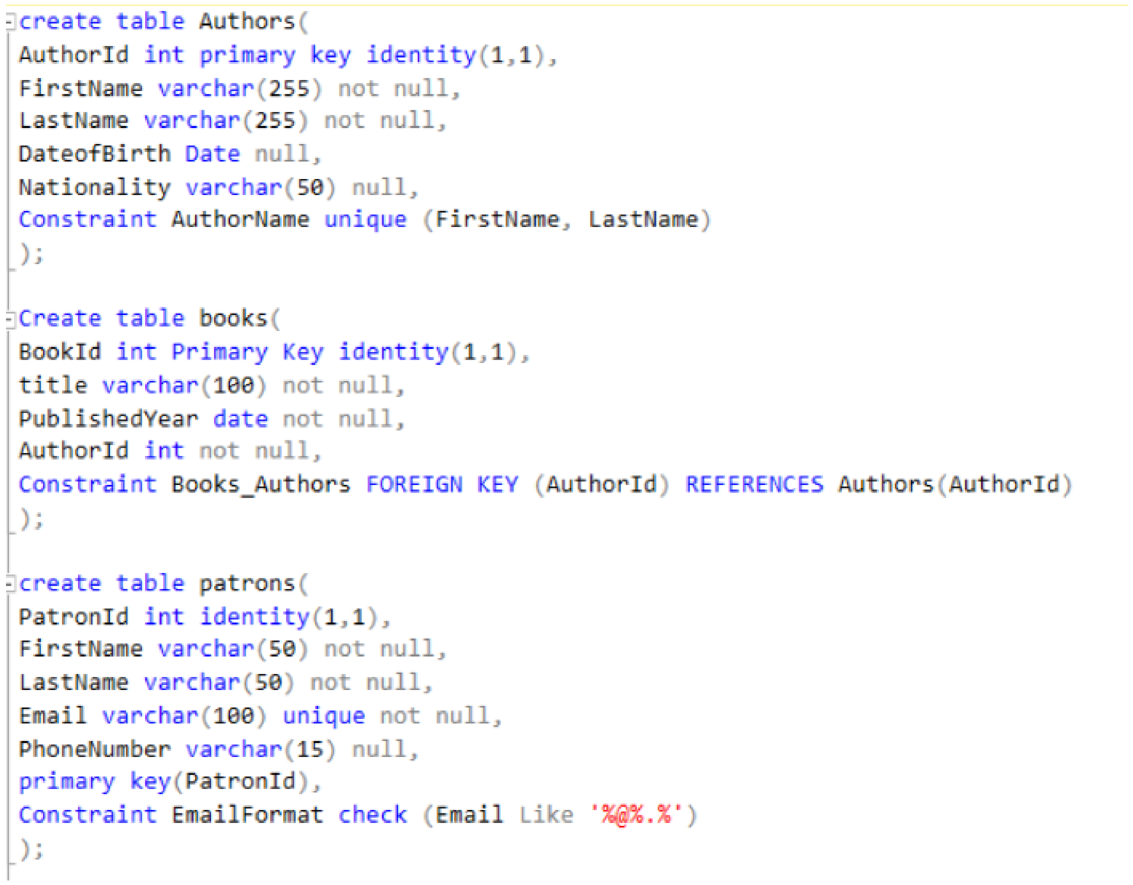
Fig-1: Update the Course table



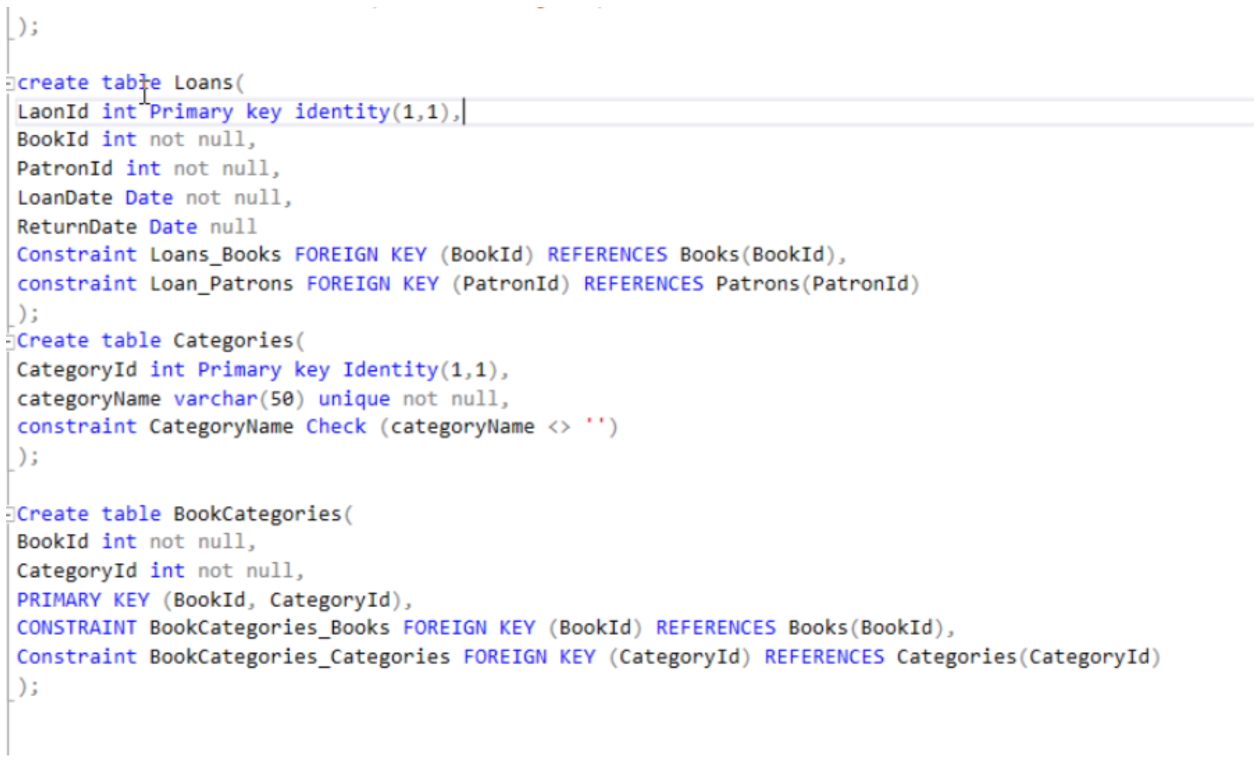
	CourseId	CourseName
1	1	SCIENCE
2	2	C
3	3	ENGLISH
4	1	SCIENCE
5	2	C
6	3	ENGLISH
7	1	BBB
8	2	CCC
9	3	ENGLISH
10	4	Biology 1001
11	1080	Biology

Fg-2: After RollBack Transaction

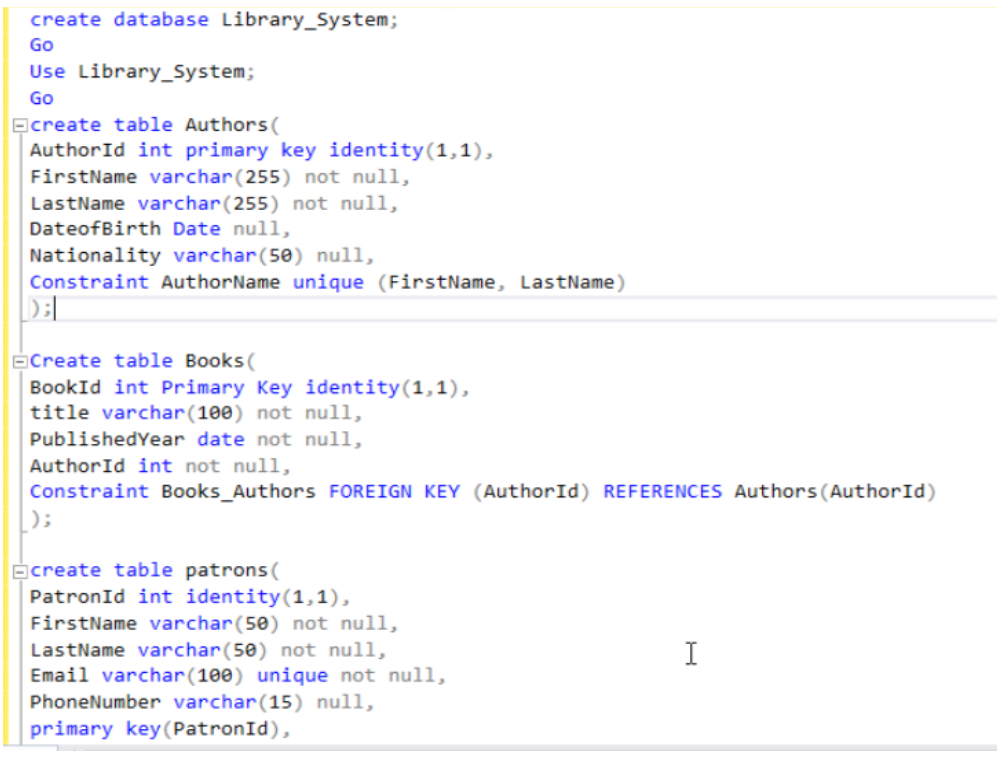
Assignment 4: Design a database schema for a library system, including tables, fields, and constraints like NOT NULL, UNIQUE, and CHECK. Include primary and foreign keys to establish relationships between tables.

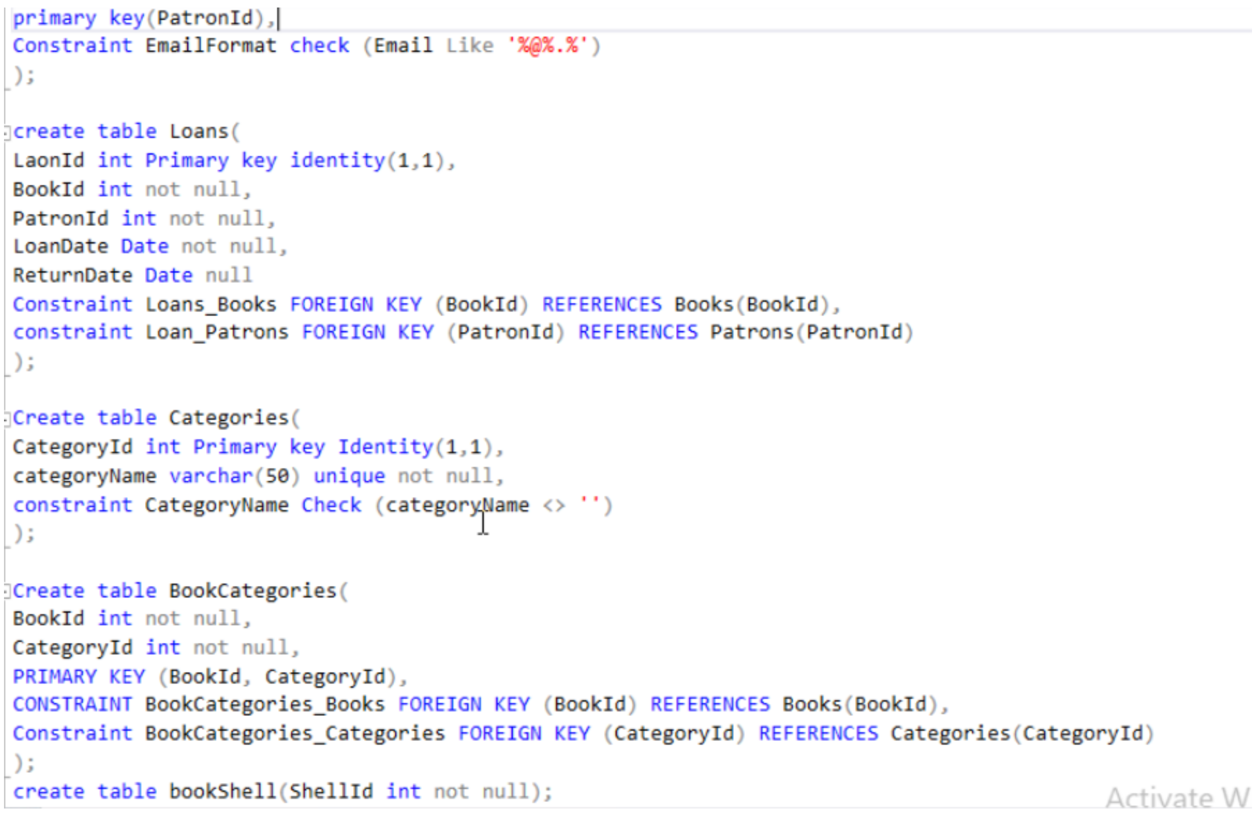


```
create table Authors(
AuthorId int primary key identity(1,1),
FirstName varchar(255) not null,
LastName varchar(255) not null,
DateofBirth Date null,
Nationality varchar(50) null,
Constraint AuthorName unique (FirstName, LastName)
);

Create table books(
BookId int Primary Key identity(1,1),
title varchar(100) not null,
PublishedYear date not null,
AuthorId int not null,
Constraint Books_Authors FOREIGN KEY (AuthorId) REFERENCES Authors(AuthorId)
);

create table patrons(
PatronId int identity(1,1),
FirstName varchar(50) not null,
LastName varchar(50) not null,
Email varchar(100) unique not null,
PhoneNumber varchar(15) null,
primary key(PatronId),
Constraint EmailFormat check (Email Like '%@%.%')
);
```


```
);

create table Loans(
LaonId int Primary key identity(1,1),
BookId int not null,
PatronId int not null,
LoanDate Date not null,
ReturnDate Date null
Constraint Loans_Books FOREIGN KEY (BookId) REFERENCES Books(BookId),
constraint Loan_Patrons FOREIGN KEY (PatronId) REFERENCES Patrons(PatronId)
);
Create table Categories(
CategoryId int Primary key Identity(1,1),
categoryName varchar(50) unique not null,
constraint CategoryName Check (categoryName <> '')
);

Create table BookCategories(
BookId int not null,
CategoryId int not null,
PRIMARY KEY (BookId, CategoryId),
CONSTRAINT BookCategories_Books FOREIGN KEY (BookId) REFERENCES Books(BookId),
Constraint BookCategories_Categories FOREIGN KEY (CategoryId) REFERENCES Categories(CategoryId)
);
```

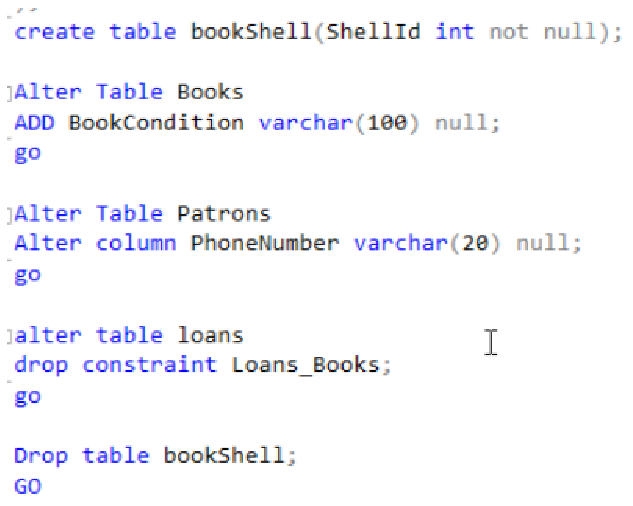
Assignment 5: Write SQL statements to CREATE a new database and tables that reflect the library schema you designed earlier. Use ALTER statements to modify the table structures and DROP statements to remove a redundant table.


```
create database Library_System;
Go
Use Library_System;
Go
create table Authors(
AuthorId int primary key identity(1,1),
FirstName varchar(255) not null,
LastName varchar(255) not null,
DateofBirth Date null,
Nationality varchar(50) null,
Constraint AuthorName unique (FirstName, LastName)
);

Create table Books(
BookId int Primary Key identity(1,1),
title varchar(100) not null,
PublishedYear date not null,
AuthorId int not null,
Constraint Books_Authors FOREIGN KEY (AuthorId) REFERENCES Authors(AuthorId)
);

create table patrons(
PatronId int identity(1,1),
FirstName varchar(50) not null,
LastName varchar(50) not null,
Email varchar(100) unique not null,
PhoneNumber varchar(15) null,
primary key(PatronId),
```

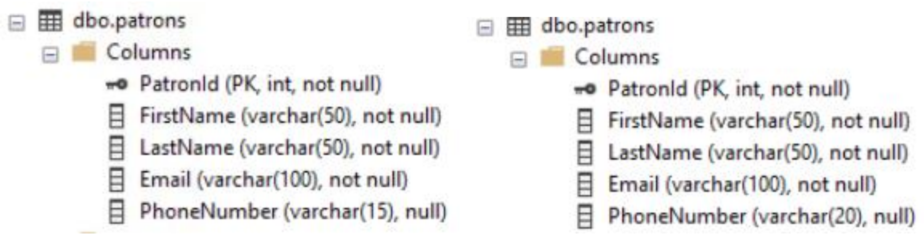


```
primary key(PatronId),
Constraint EmailFormat check (Email Like '%@%.%')
);

create table Loans(
LaonId int Primary key identity(1,1),
BookId int not null,
PatronId int not null,
LoanDate Date not null,
ReturnDate Date null
Constraint Loans_Books FOREIGN KEY (BookId) REFERENCES Books(BookId),
constraint Loan_Patrons FOREIGN KEY (PatronId) REFERENCES Patrons(PatronId)
);

Create table Categories(
CategoryId int Primary key Identity(1,1),
categoryName varchar(50) unique not null,
constraint CategoryName Check (categoryName <> '')
);

Create table BookCategories(
BookId int not null,
CategoryId int not null,
PRIMARY KEY (BookId, CategoryId),
CONSTRAINT BookCategories_Books FOREIGN KEY (BookId) REFERENCES Books(BookId),
Constraint BookCategories_Categories FOREIGN KEY (CategoryId) REFERENCES Categories(CategoryId)
);
create table bookShell(ShellId int not null);
```

Activate W



```
create table bookShell(ShellId int not null);

Alter Table Books
ADD BookCondition varchar(100) null;
go

Alter Table Patrons
Alter column PhoneNumber varchar(20) null;
go

alter table loans
drop constraint Loans_Books;
go

Drop table bookShell;
GO
```



```
dbo.patrons
  Columns
    PatronId (PK, int, not null)
    FirstName (varchar(50), not null)
    LastName (varchar(50), not null)
    Email (varchar(100), not null)
    PhoneNumber (varchar(15), null)
```

```
dbo.patrons
  Columns
    PatronId (PK, int, not null)
    FirstName (varchar(50), not null)
    LastName (varchar(50), not null)
    Email (varchar(100), not null)
    PhoneNumber (varchar(20), null)
```

Fig-1: After alter patron table column phone number datatype from Varchar(15) to Varchar(20)

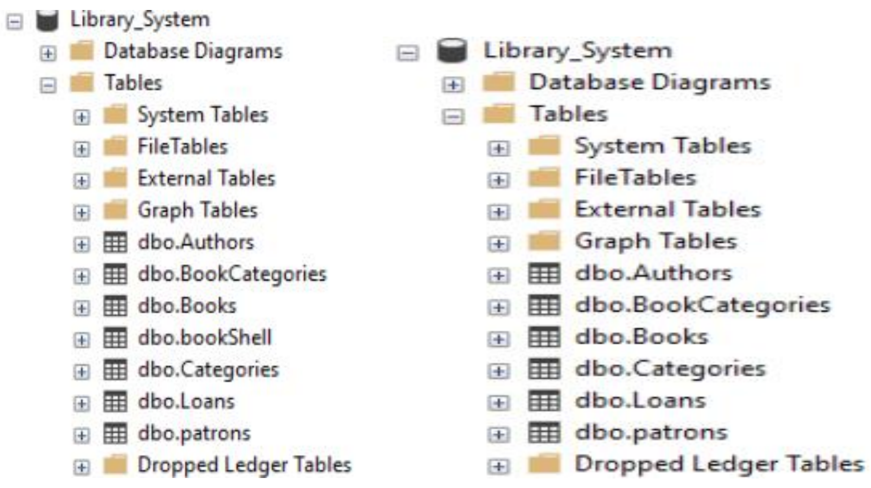
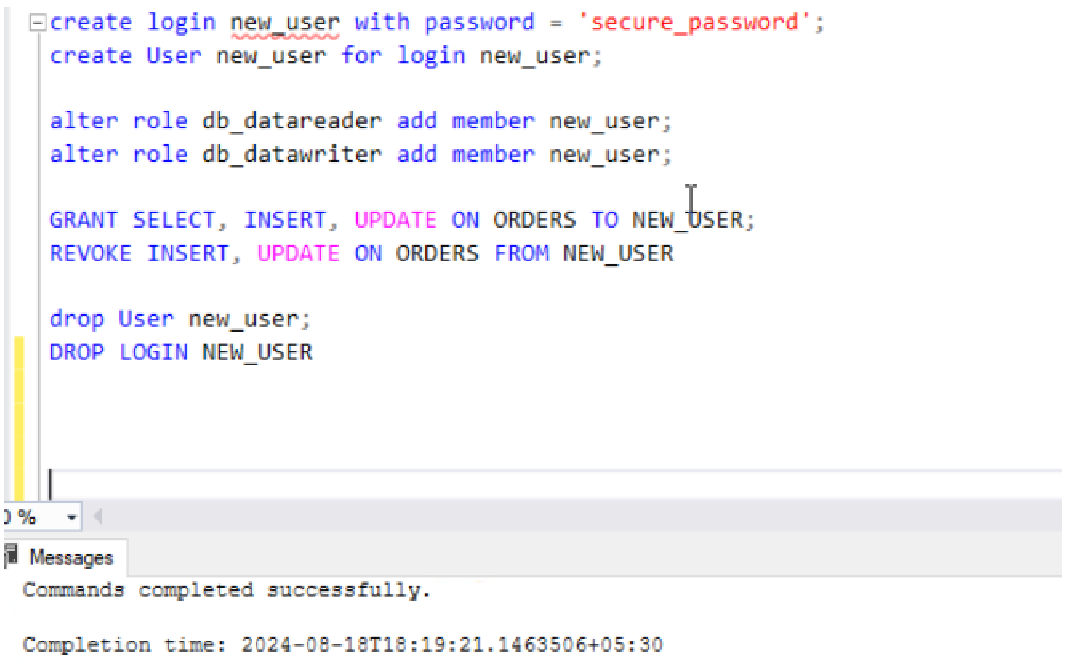


Fig-2: After Drop the Table of Bookshell

Assignment 6: Create a new database user with specific privileges using the CREATE USER and GRANT commands. Then, write a script to REVOKE certain privileges and DROP the user.



```
create login new_user with password = 'secure_password';
create User new_user for login new_user;

alter role db_datareader add member new_user;
alter role db_datawriter add member new_user;

GRANT SELECT, INSERT, UPDATE ON ORDERS TO NEW_USER;
REVOKE INSERT, UPDATE ON ORDERS FROM NEW_USER

drop User new_user;
DROP LOGIN NEW_USER
```

```
Commands completed successfully.

Completion time: 2024-08-18T18:19:21.1463506+05:30
```