Activity No. 14.1 - Transactions	
Name: Efa, Christian Guevarra, Hans Angelo Mendoza, John Renzo Nicolas, Sean Julian Vinluan, Armando	**Date:** 02/12/2022
Section: CPE21S3	**Instructor:** Dr. Jonathan Vidal Taylar

Objectives:

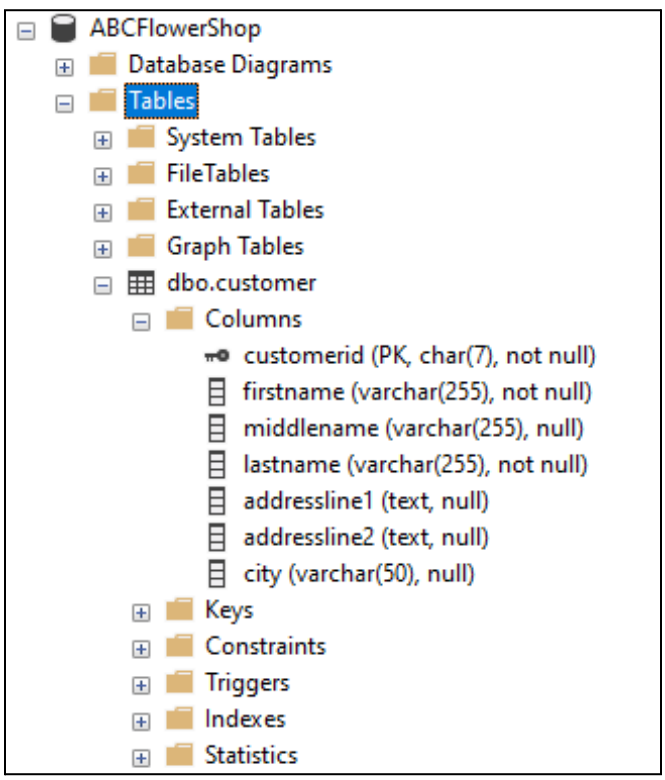
This activity aims to create and manage transactions in databases

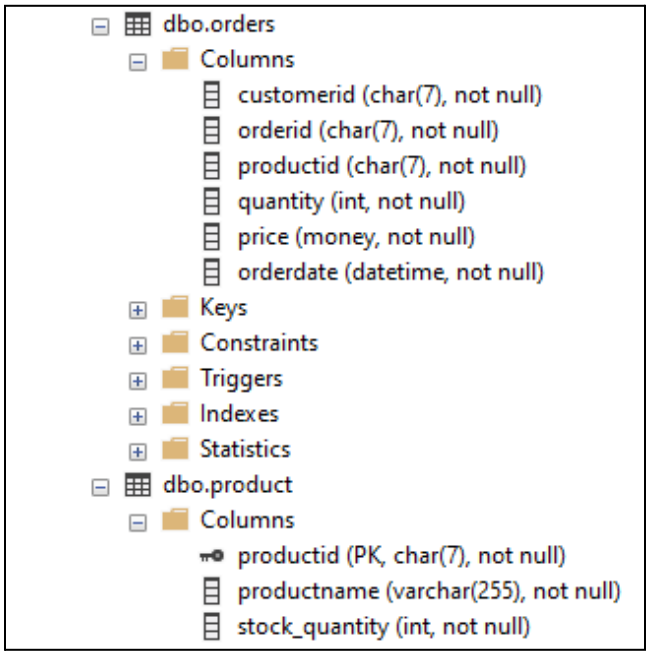
Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Create and apply transactions in databases
2.2 Troubleshoot error(s) encountered in developing transactions.

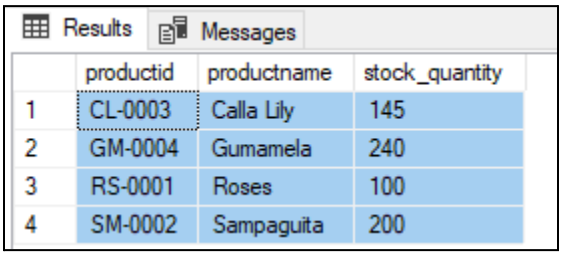
Output

1. Create the ABCFlowerShop database and the tables in ABCFlowerShop database





2. Insert the following data into the product table.



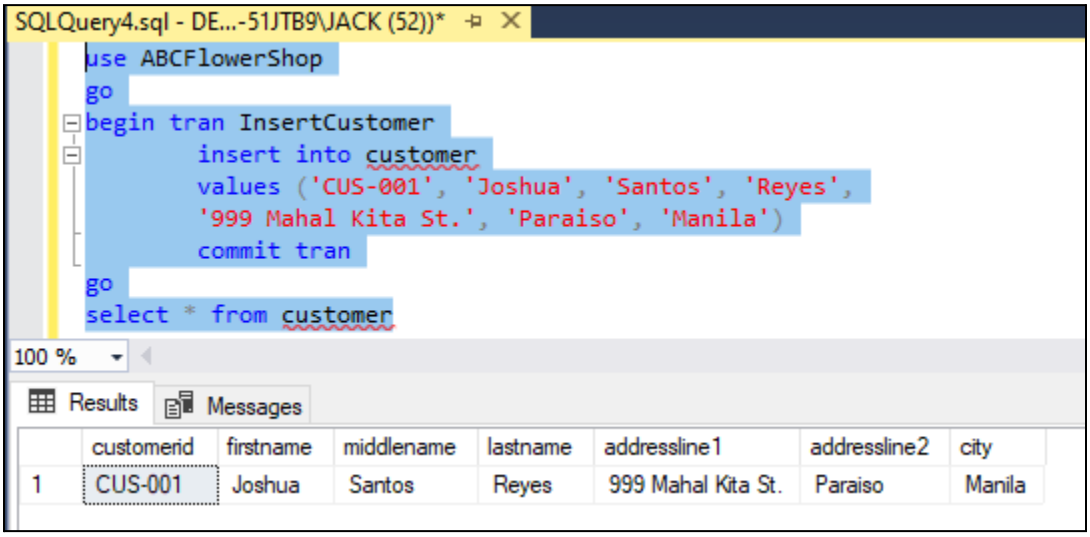
Results | Messages

	productid	productname	stock_quantity
1	CL-0003	Calla Lily	145
2	GM-0004	Gumamela	240
3	RS-0001	Roses	100
4	SM-0002	Sampaguita	200

COMMIT TRANSACTION

A. Using T-SQL

Step 1: In Object Explorer, connect to an instance of Database Engine and then expand that instan
Step 2: Create a new query. Type the given T-SQL. Execute the given query.



SQLQuery4.sql - DE...-51JTB9\JACK (52))*

```
use ABCFlowerShop
go
begin tran InsertCustomer
        insert into customer
        values ('CUS-001', 'Joshua', 'Santos', 'Reyes',
        '999 Mahal Kita St.', 'Paraiso', 'Manila')
        commit tran
go
select * from customer
```

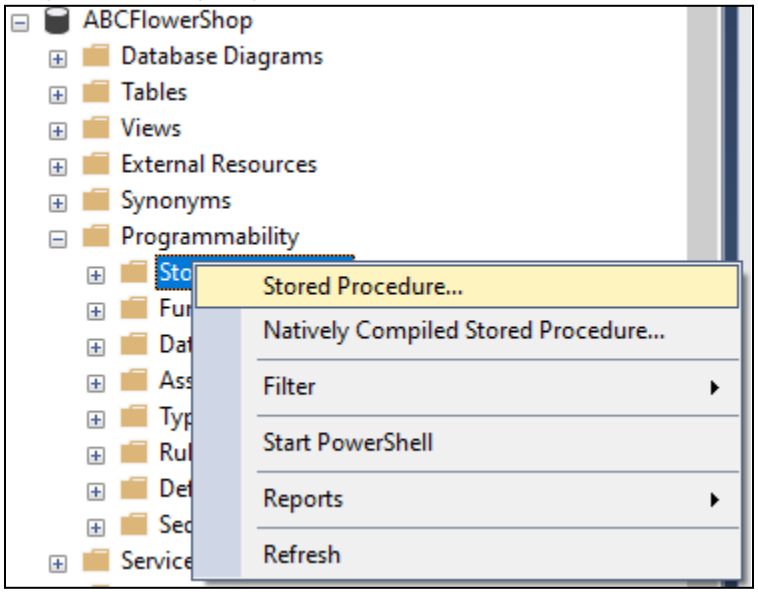
100 %

Results | Messages

	customerid	firstname	middlename	lastname	addressline1	addressline2	city
1	CUS-001	Joshua	Santos	Reyes	999 Mahal Kita St.	Paraiso	Manila

B. Using Stored Procedure

Step 1: In Object Explorer, connect to an instance of Database Engine and then expand that instance.
Step 2: Expand Databases, expand the ABCFlowerShop database, and then expand Programmability. Right-click Stored Procedures. Choose Stored Procedure



Step 3: Create the uspInsertOrder stored procedure. Type your name as author and the current date in Create data field. Click Execute or Press F5 to save the stored procedure.

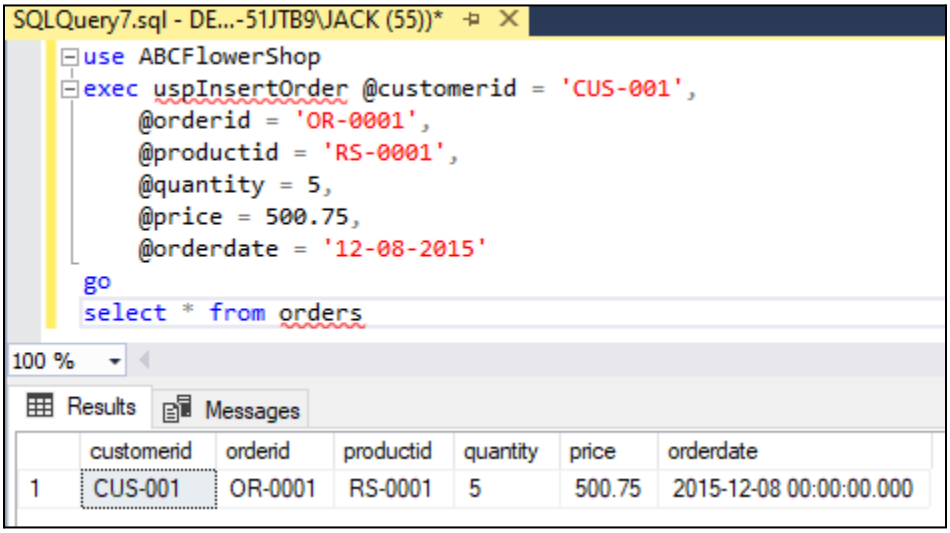
```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- ============================================
-- Author:      <Project 11,,Name>
-- Create date: <November 28,,>
-- Description: <Description,,>
-- ============================================
CREATE PROCEDURE uspInsertOrder
    -- Add the parameters for the stored procedure here
    @customerid char(7), @orderid char(7), @productid char(7),
    @quantity int, @price money, @orderdate datetime
AS
BEGIN TRAN InsertOrder
    BEGIN TRY
        INSERT orders (customerid, orderid, productid,
        quantity, price, orderdate)
        VALUES(@customerid, @orderid, @productid,
        @quantity, @price, @orderdate)
        COMMIT TRAN InsertOrder
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN InsertOrder
    END CATCH
```

100 %

Messages

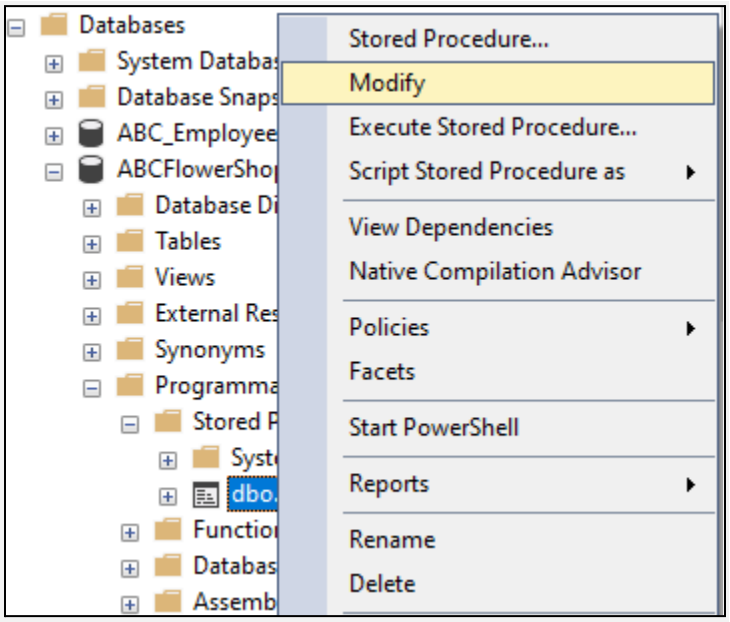
```
Commands completed successfully.

Completion time: 2022-11-28T15:01:26.6299176+08:00
```

Step 4: Create a new query to test the uspInsertOrder stored procedure. Click Execute or Press F5



ROLLBACK TRANSACTION

Step 1: In Object Explorer, connect to an instance of Database Engine and then expand that instance.
Step 2: Expand Databases, expand the ABCFlowerShop database, and then expand Programmability.
Expand Stored Procedures and Right-click uspInsertOrder. Choose Modify.



Step 3:Modify the uspInsertOrder stored procedure using the given figure to rollback the transaction if the ordered product is out of stock.

SQLQuery9.sql - DE...-51JTB9\JACK (59))*

```sql
USE [ABCFlowerShop]
GO
/****** Object:  StoredProcedure [dbo].[uspInsertOrder]    Scri
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      <Project 11,,Name>
-- Create date: <November 28,,>
-- Description: <Description,,>
-- =============================================
ALTER PROCEDURE [dbo].[uspInsertOrder]
    -- Add the parameters for the stored procedure here
    @customerid char(7), @orderid char(7), @productid char(7),
    @quantity int, @price money, @orderdate datetime
AS
DECLARE @stock_quantity int
BEGIN
SET NOCOUNT ON
SET XACT_ABORT ON
    BEGIN TRY
        BEGIN TRAN InsertOrder
        SELECT @stock_quantity = stock_quantity from product
        where productid = @productid
        INSERT orders (customerid, orderid, productid,
        quantity, price, orderdate)
        VALUES(@customerid, @orderid, @productid,
        @quantity, @price, @orderdate)
        if @quantity > @stock_quantity
            BEGIN
            ROLLBACK TRAN InsertOrder
            PRINT 'Out of stock'
            END
        else

            COMMIT TRAN InsertOrder

    END TRY
    BEGIN CATCH
        SELECT ERROR_MESSAGE()
        IF @@TRANCOUNT > 0
            ROLLBACK TRAN InsertOrder
    END CATCH

END
```

100 %

Messages

```
Commands completed successfully.

Completion time: 2022-11-28T15:14:24.8501409+08:00
```

Step 4: Create a new query to test the rollback transaction of uspInsertOrder stored procedure. Click Execute or Press F5

SQLQuery15.sql - D...-51JTB9\JACK (52))*

```
USE ABCFlowerShop
GO
EXEC uspInsertOrder @customerid = 'CUS-001',
                    @orderid = 'OR-00001',
                    @productid = 'CL-0003',
                    @quantity = 200,
                    @price = 500.75,
                    @orderdate = '12-08-2015'
Select * from orders
```

100 %

Results | Messages

customerid	orderid	productid	quantity	price	orderdate

SQLQuery15.sql - D...-51JTB9\JACK (52))*

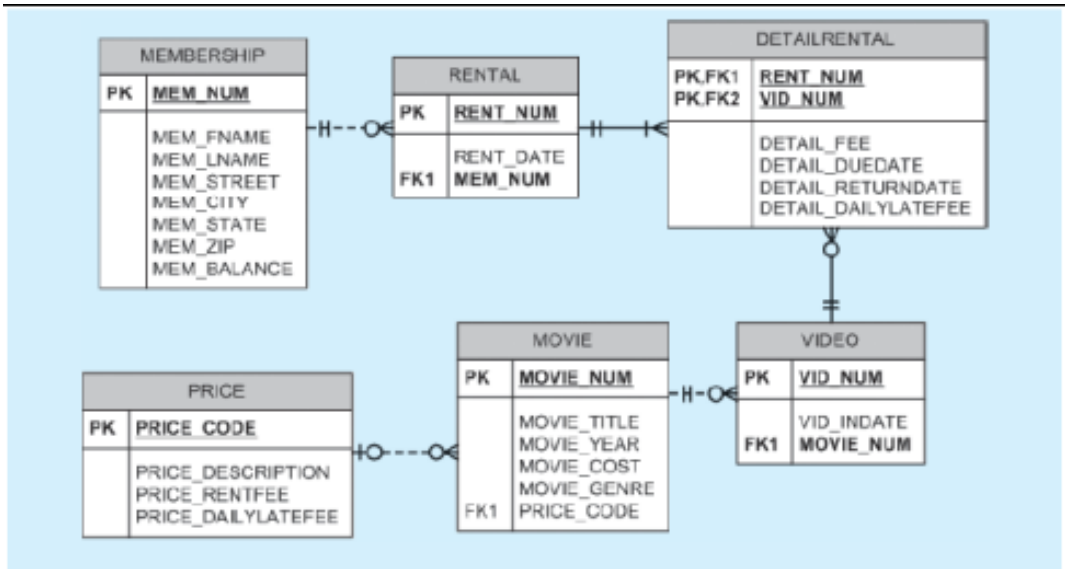
```
USE ABCFlowerShop
GO
EXEC uspInsertOrder @customerid = 'CUS-001',
                    @orderid = 'OR-00001',
                    @productid = 'CL-0003',
                    @quantity = 200,
                    @price = 500.75,
                    @orderdate = '12-08-2015'
Select * from orders
```

100 %

Results | Messages

```
Out of stock

(0 rows affected)

Completion time: 2022-11-28T15:30:53.7726621+08:00
```

Supplementary Activity



1. Create a database TinyVideo.
2. Using the given ERD, create the following tables and relationships. Assign the appropriate data types

Supplementary 1 a...ryan Mendoza (63))* | SQLQuery10.sql - D...ryan Mendoza (60))*

```sql
CREATE DATABASE TinyVideo
GO

USE TinyVideo
GO

CREATE TABLE MEMBERSHIP(
    MEM_NUM INT PRIMARY KEY,
    MEM_FNAME VARCHAR(50),
    MEM_LNAME VARCHAR(50),
    MEM_STREET VARCHAR(100),
    MEM_CITY VARCHAR(100),
    MEM_STATE VARCHAR(100),
    MEM_ZIP VARCHAR(100),
    MEM_BALANCE VARCHAR(100)
    )
GO

CREATE TABLE RENTAL(
    RENT_NUM INT PRIMARY KEY,
    RENT_DATE DATETIME,
    MEM_NUM INT NOT NULL
    FOREIGN KEY (MEM_NUM) REFERENCES MEMBERSHIP(MEM_NUM)
    )
GO

CREATE TABLE PRICE(
    PRICE_CODE CHAR(7) PRIMARY KEY,
    PRICE_DESCRIPTION TEXT,
    PRICE_RENTALFEE MONEY,
    PRICE_DAILYLATEFEE MONEY
    )
GO

CREATE TABLE MOVIE(
    MOVIE_NUM INT PRIMARY KEY,
    MOVIE_TITLE VARCHAR(100),
    MOVIE_YEAR DATE,
    MOVIE_COST MONEY,
    MOVIE_GENRE VARCHAR(100),
    PRICE_CODE CHAR(7)
    FOREIGN KEY (PRICE_CODE) REFERENCES PRICE(PRICE_CODE)
    )
GO
```

90 %

Messages

```
Commands completed successfully.

Completion time: 2022-11-30T17:33:13.3187521+08:00
```

Continuation

```sql
CREATE TABLE VIDEO(
    VID_NUM INT PRIMARY KEY,
    VID_INDATE DATETIME,
    MOVIE_NUM INT NOT NULL
    FOREIGN KEY (MOVIE_NUM) REFERENCES MOVIE(MOVIE_NUM)
    )
GO

CREATE TABLE DETAILRENTAL(
    RENT_NUM INT,
    VID_NUM INT,
    DETAIL_FEE MONEY,
    DETAIL_DUEDATE DATETIME,
    DETAIL_RETURNDATE DATETIME,
    DETAIL_DAILYLATEFEE MONEY
    FOREIGN KEY (RENT_NUM) REFERENCES RENTAL(RENT_NUM),
    FOREIGN KEY (VID_NUM) REFERENCES VIDEO(VID_NUM)
    )
GO
```

90 %

Messages

```
Commands completed successfully.

Completion time: 2022-11-30T17:33:13.3187521+08:00
```

Observation:

In this Preliminary Procedure, we have created the database using a new query based on the given entity relationship diagram.

3. Create a transaction to insert a new movie and display an appropriate error message if the user inserts a duplicate movie.

Creation of Transaction in a stored procedure

SQLQuery12.sql - D...ryan Mendoza (57))* | SQLQuery10.sql - D...ryan Mendoza (60))* | Supplementary 3.sq...ryan

```sql
GO
-- ============================================
-- Author:      <Group 11 CPE21S3>
-- Create date: <November 30, 2022>
-- Description: <Supplementary Step 3>
-- ============================================
CREATE PROCEDURE uspInsertMovie
    -- Add the parameters for the stored procedure here
    @MOVIE_NUM INT, @MOVIE_TITLE VARCHAR(100), @MOVIE_YEAR DATE,
    @MOVIE_COST MONEY, @MOVIE_GENRE VARCHAR(100), @PRICE_CODE CHAR(7)
AS
DECLARE @MOVIE VARCHAR(100)
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    BEGIN TRY
        BEGIN TRAN InsertMovie
        SELECT @MOVIE = MOVIE_TITLE FROM MOVIE
        WHERE MOVIE_TITLE = @MOVIE_TITLE

        INSERT INTO MOVIE (MOVIE_NUM, MOVIE_TITLE, MOVIE_YEAR, MOVIE_COST, MOVIE_GENRE, PRICE_CODE)
        VALUES
        (@MOVIE_NUM, @MOVIE_TITLE, @MOVIE_YEAR, @MOVIE_COST, @MOVIE_GENRE, @PRICE_CODE)

        IF @MOVIE_TITLE = @MOVIE
            BEGIN
            ROLLBACK TRAN InsertMovie
            PRINT 'Movie Title already on List'
            END
        ELSE
            COMMIT TRAN InsertMovie
    END TRY
    BEGIN CATCH
        SELECT ERROR_MESSAGE()
        IF @@TRANCOUNT > 0
            ROLLBACK TRAN InsertMovie
    END CATCH
END
GO
```

90 %

Messages

```
Commands completed successfully.

Completion time: 2022-11-30T17:13:55.2562879+08:00
```

Preliminary preparation for Price table

SQLQuery10.sql - D...ryan Mendoza (60))* | Supplementary 3.sq...ryan Mendoza (58)) | Supplem

```sql
USE TinyVideo
GO

INSERT INTO PRICE (PRICE_CODE)
    VALUES
    ('PRC-001')
GO

SELECT * FROM PRICE
GO
```

90 %

Results | Messages

	PRICE_CODE	PRICE_DESCRIPTION	PRICE_RENTALFEE	PRICE_DAILYLATEFEE
1	PRC-001	NULL	NULL	NULL

When the insertion of the movie is successful

SQLQuery12.sql - D...ryan Mendoza (57))* | SQLQuery10.sql - D...ryan Mendoza (60))* | Supplementary

```sql
USE TinyVideo
GO

EXEC uspInsertMovie
    @MOVIE_NUM = 1001,
    @MOVIE_TITLE = 'Avengers Infinity War',
    @MOVIE_YEAR = '2018-04-25',
    @MOVIE_COST = 280.00,
    @MOVIE_GENRE = 'Sci-Fi',
    @PRICE_CODE = 'PRC-001'
GO

SELECT * FROM MOVIE
GO
```

90 %

Results | Messages

	MOVIE_NUM	MOVIE_TITLE	MOVIE_YEAR	MOVIE_COST	MOVIE_GENRE	PRICE_CODE
1	1001	Avengers Infinity War	2018-04-25	280.00	Sci-Fi	PRC-001

When the insertion of the movie is unsuccessful (duplicate movie titles)

SQLQuery12.sql - D...ryan Mendoza (57))* | SQLQuery10.sql - D...ryan Mendoza (60))* | Supplementary

```sql
USE TinyVideo
GO

EXEC uspInsertMovie
    @MOVIE_NUM = 1002,
    @MOVIE_TITLE = 'Avengers Infinity War',
    @MOVIE_YEAR = '2018-04-25',
    @MOVIE_COST = 280.00,
    @MOVIE_GENRE = 'Sci-Fi',
    @PRICE_CODE = 'PRC-001'
GO

SELECT * FROM MOVIE
GO
```

90 %

Results | Messages

	MOVIE_NUM	MOVIE_TITLE	MOVIE_YEAR	MOVIE_COST	MOVIE_GENRE	PRICE_CODE
1	1001	Avengers Infinity War	2018-04-25	280.00	Sci-Fi	PRC-001

SQLQuery12.sql - D...ryan Mendoza (57))* | SQLQuery10.sql - D...ryan Mendoza (60))* | Supplementary 3.sq...

```sql
USE TinyVideo
GO

EXEC uspInsertMovie
    @MOVIE_NUM = 1002,
    @MOVIE_TITLE = 'Avengers Infinity War',
    @MOVIE_YEAR = '2018-04-25',
    @MOVIE_COST = 280.00,
    @MOVIE_GENRE = 'Sci-Fi',
    @PRICE_CODE = 'PRC-001'
GO

SELECT * FROM MOVIE
GO
```

90 %

Results | Messages

```
Movie Title already on List

(1 row affected)

Completion time: 2022-11-30T17:15:17.3117245+08:00
```

Observation:

We created a transaction as a stored procedure wherein it will deny entries if the entered movie already exists on the Movies table. The transaction will use the movie title as the basis whether to add or deny the entry since repeating primary keys will implicitly deny multiple copies of it. As we can observe on our screenshots, the 1st insertion of movie 'Avengers Infinity War' was successful, but on the 2nd insertion of the same movie title, it was denied since it already exists on the Movies table.

4. Create a transaction to insert a rental transaction and perform the following conditions:

Supplementary 4.sq...ryan Mendoza (57)) | SQLQuery1.sql - DE...ryan Mendoza (53))* | Supplementa

```sql
GO
-- ============================================
-- Author:      <Group 11 CPE21S3>
-- Create date: <November 30, 2022>
-- Description: <Supplementary Step 4>
-- ============================================
CREATE PROCEDURE uspInsertRental
    -- Add the parameters for the stored procedure here
    @RENT_NUM INT, @RENT_DATE DATETIME, @MEM_NUM INT
AS
DECLARE @Membership INT, @Balance MONEY
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    BEGIN TRY
        BEGIN TRAN InsertRental
        SELECT @Membership = COUNT(*) FROM MEMBERSHIP
        WHERE MEM_NUM = @MEM_NUM

        IF @Membership = 0
            BEGIN
                ROLLBACK TRAN InsertRental
                PRINT 'Unverified Membership. Please get a memberhsip first before renting'
            END
        ELSE
            BEGIN
                SELECT @Balance = MEM_BALANCE FROM MEMBERSHIP
                WHERE MEM_NUM = @MEM_NUM

                PRINT 'Your Previous Balance: ' + CONVERT(VARCHAR(20), @Balance)

                IF @Balance > 500
                    BEGIN
                        INSERT INTO RENTAL (RENT_NUM, RENT_DATE, MEM_NUM)
                        VALUES
                        (@RENT_NUM, @RENT_DATE, @MEM_NUM)
                        PRINT 'Rental Transaction Successful'
                        PRINT 'Your New Balance: ' + CONVERT(VARCHAR(20), @Balance - 500)
                    END
                ELSE
                    PRINT 'Insufficient Membership Balance, Rental Transaction Unsuccessful'
                COMMIT TRAN InsertRental
            END
    END TRY
    BEGIN CATCH
        SELECT ERROR_MESSAGE()
        IF @@TRANCOUNT > 0
            ROLLBACK TRAN InsertRental
    END CATCH
END
GO
```

82 %

Messages

```
Commands completed successfully.

Completion time: 2022-12-01T20:08:49.6127568+08:00
```

a. Verify that the membership number exists in the MEMBERSHIP table. If it does not exist, then a message should be displayed stating that the membership does not exist and no data should be written to the database.

When Membership does not exist

Supplementary 4.sq...ryan Mendoza (57)) | SQLQuery1.sql - DE...rya

```sql
USE TinyVideo
GO

EXEC uspInsertRental
    @RENT_NUM = 1,
    @RENT_DATE = '2022-11-30',
    @MEM_NUM = 1001
GO

SELECT * FROM RENTAL
GO
```

90 %

Results | Messages

RENT_NUM	RENT_DATE	MEM_NUM

Supplementary 4.sq...ryan Mendoza (57)) | SQLQuery1.sql - DE...ryan Mend

```sql
USE TinyVideo
GO

EXEC uspInsertRental
    @RENT_NUM = 1,
    @RENT_DATE = '2022-11-30',
    @MEM_NUM = 1001
GO

SELECT * FROM RENTAL
GO
```

90 %

Results | Messages

```
Unverified Membership. Please get a memberhsip first before renting

(0 rows affected)

Completion time: 2022-12-01T19:57:06.3459548+08:00
```

When Membership do exist

Supplementary 4.sq...ryan Mendoza (57)) | SQLQuery1.sql - DE...ryan Mendoza (53))* | Supplementary 1 a...

```sql
USE TinyVideo
GO

INSERT INTO MEMBERSHIP(MEM_NUM, MEM_FNAME, MEM_LNAME, MEM_BALANCE)
    VALUES
    (1001, 'Leonardo', 'Da Vinci', 600),
    (1002, 'Pablo', 'Picasso', 300)
GO

SELECT * FROM MEMBERSHIP
GO
```

90 %

Results | Messages

	MEM_NUM	MEM_FNAME	MEM_LNAME	MEM_STREET	MEM_CITY	MEM_STATE	MEM_ZIP	MEM_BALA
1	1001	Leonardo	Da Vinci	NULL	NULL	NULL	NULL	600
2	1002	Pablo	Picasso	NULL	NULL	NULL	NULL	300

Supplementary 4.sq...ryan Mendoza (57)) | SQLQuery1.sql - DE...r

```sql
USE TinyVideo
GO

EXEC uspInsertRental
    @RENT_NUM = 1,
    @RENT_DATE = '2022-11-30',
    @MEM_NUM = 1001
GO

SELECT * FROM RENTAL
GO
```

82 %

Results | Messages

	RENT_NUM	RENT_DATE	MEM_NUM
1	1	2022-11-30 00:00:00.000	1001

Supplementary 4.sq...ryan Mendoza (57)) | SQLQuery1.sql - DE...ryan

```sql
USE TinyVideo
GO

EXEC uspInsertRental
    @RENT_NUM = 1,
    @RENT_DATE = '2022-11-30',
    @MEM_NUM = 1001
GO

SELECT * FROM RENTAL
GO
```

82 %

Results | Messages

```
Your Previous Balance: 600.00
Rental Transaction Successful
Your New Balance: 100.00

(1 row affected)

Completion time: 2022-12-01T20:10:35.5508942+08:00
```

Observation for part 4a:

In this procedure, we are tasked to make a transaction to allow or deny entries if a user is making a rental based on his/her membership. For the first two screenshots provided above, the RENTAL was denied since the RENTAL made has no existing or matching MEM_NUM on the MEMBERSHIPS table. On contrary to our latter screenshots, the RENTAL was allowed since there exists a matching membership number on the MEMBERSHIPS table.

b. If the membership does exist, then retrieve the membership balance and display a message stating the balance amount as the previous balance. If the membership balance is greater than 500.00, allow the rental transaction. Otherwise, no data should be written to the database.

When a membership have enough funds

Supplementary 4.sq...ryan Mendoza (57)) | SQLQuery1.sql - DE...ryan

```sql
USE TinyVideo
GO

EXEC uspInsertRental
    @RENT_NUM = 1,
    @RENT_DATE = '2022-11-30',
    @MEM_NUM = 1001
GO

SELECT * FROM RENTAL
GO
```

82 %

Results | Messages

```
Your Previous Balance: 600.00
Rental Transaction Successful
Your New Balance: 100.00

(1 row affected)

Completion time: 2022-12-01T20:10:35.5508942+08:00
```

When a membership does not have enough funds

Supplementary 4.sq...ryan Mendoza (57)) | SQLQuery1.sql - DE

```sql
USE TinyVideo
GO

EXEC uspInsertRental
    @RENT_NUM = 2,
    @RENT_DATE = '2022-11-30',
    @MEM_NUM = 1002
GO

SELECT * FROM RENTAL
GO
```

82 %

Results | Messages

	RENT_NUM	RENT_DATE	MEM_NUM
1	1	2022-11-30 00:00:00.000	1001

Supplementary 4.sq...ryan Mendoza (57)) | SQLQuery1.sql - DE...ryan M

```sql
USE TinyVideo
GO

EXEC uspInsertRental
    @RENT_NUM = 2,
    @RENT_DATE = '2022-11-30',
    @MEM_NUM = 1002
GO

SELECT * FROM RENTAL
GO
```

82 %

Results | Messages

```
Your Previous Balance: 300.00
Insufficient Membership Balance, Rental Transaction Unsuccessful

(1 row affected)

Completion time: 2022-12-01T20:11:50.3926084+08:00
```

Observation part 4b:

On part b, we will further classify if the rental to be made will be successful by determining if the membership has enough balance to pay the rental needed for the request, which is defined as 500 on this procedure. Going back to the insertion of the members, we have MEM_NUM 1001 having a balance of 600, while MEM_NUM 1002 has a balance of 300. By testing both of the Members to perform a Rental, MEM_NUM 1001, successfully made a RENTAL since it initially had a balance of 600 which is enough to pay for the rent. On the other hand, MEM_NUM 1002, failed to make the RENTAL since it does not have enough balance for the rent

5. Create a transaction to insert a video transaction and perform the following conditions:

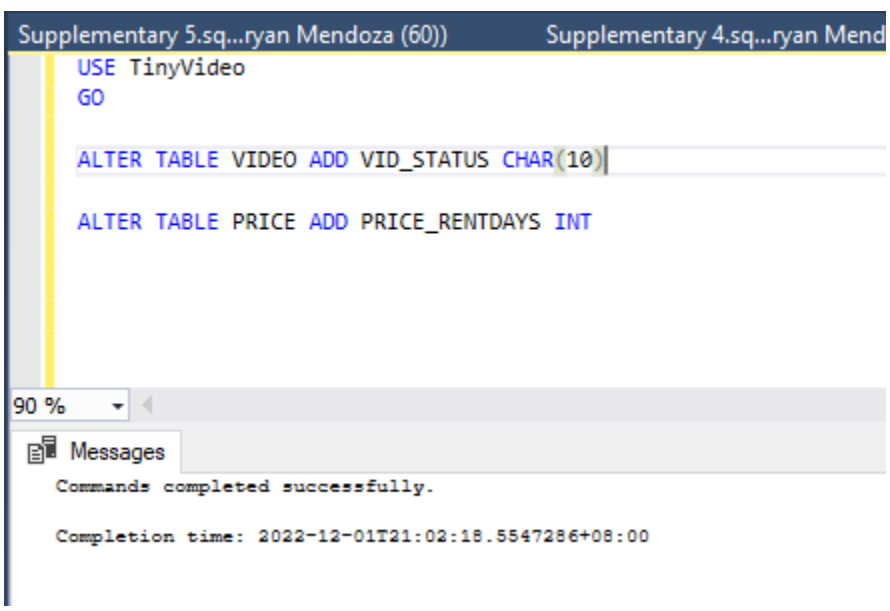
a. Verify that the video number exists in the VIDEO table. If it does not exist, then display a message that the video does not exist, and do not write any data to the database.

b. If the video number does exist, then verify that the VID_STATUS for that video is “IN”.

i. If the status is not “IN”, then display a message that the return of the video must be entered before it can be rented again, and do not write any data to the database.

ii. If the status is “IN”, then retrieve the values of PRICE_RENTFEE, PRICE_DAILYLATEFEE, and PRICE_RENTDAYS associated with the video from the PRICE table. Calculate the due date for the video rental by adding the number of days found in PRICE_RENTDAYS above to 11:59:59PM (hours:minutes:seconds) on the current system date.

Preliminary Task to do the procedure: Adding new fields for VIDEO table and PRICE table



Creation of the new Transaction as a Stored Procedure

Supplementary 5.sq...ryan Mendoza (60))* | Supplementary 4.sq...ryan Mendoza (57)) | SQLQuery1

```sql
-- ============================================
-- Author:      <Group 11 CPE21S3>
-- Create date: <December 01, 2022>
-- Description: <Supplementary Step 5>
-- ============================================
CREATE PROCEDURE uspInsertDetailRental
    -- Add the parameters for the stored procedure here
    @RENT_NUM INT, @VID_NUM INT
AS
DECLARE @Video INT, @Status CHAR(10), @LateFee MONEY, @RentDays INT, @RentFee MONEY, @Time DATETIME
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    BEGIN TRY
        BEGIN TRAN InsertDetailRental
        SELECT @Video = COUNT(*) FROM VIDEO
        WHERE VID_NUM = @VID_NUM

        IF @Video = 0
            BEGIN
                ROLLBACK TRAN InsertDetailRental
                PRINT 'The Video does not Exist.'
            END
        ELSE
            BEGIN
                SELECT @Status = VID_STATUS FROM VIDEO
                WHERE VID_NUM = @VID_NUM

                IF @Status = 'IN'
                    BEGIN
                        SELECT @RentFee = a.PRICE_RENTALFEE FROM PRICE a
                        INNER JOIN MOVIE b ON a.PRICE_CODE = b.PRICE_CODE
                        INNER JOIN VIDEO c ON b.MOVIE_NUM = c.MOVIE_NUM
                        WHERE c.VID_NUM = @VID_NUM

                        SELECT @RentDays = a.PRICE_RENTDAYS FROM PRICE a
                        INNER JOIN MOVIE b ON a.PRICE_CODE = b.PRICE_CODE
                        INNER JOIN VIDEO c ON b.MOVIE_NUM = c.MOVIE_NUM
                        WHERE c.VID_NUM = @VID_NUM

                        SELECT @LateFee = a.PRICE_DAILYLATEFEE FROM PRICE a
                        INNER JOIN MOVIE b ON a.PRICE_CODE = b.PRICE_CODE
                        INNER JOIN VIDEO c ON b.MOVIE_NUM = c.MOVIE_NUM
                        WHERE c.VID_NUM = @VID_NUM

                        SELECT @Time = GETDATE() + @RentDays

                        INSERT INTO DETAILRENTAL (RENT_NUM, VID_NUM, DETAIL_FEE, DETAIL_DUEDATE, DETAIL_DAILYLATEFEE)
                            VALUES
                            (@RENT_NUM, @VID_NUM, @RentFee, @Time, @LateFee)
                        PRINT 'The Video Renting is successful. Thank you.'
                    END
                ELSE
                    PRINT 'The Video needs to be returned before renting again.'

                COMMIT TRAN InsertDetailRental
            END
    END TRY
    BEGIN CATCH
        SELECT ERROR_MESSAGE()
        IF @@TRANCOUNT > 0
            ROLLBACK TRAN InsertRental
    END CATCH
END
GO
```

68 %

Messages

```
Commands completed successfully.

Completion time: 2022-12-01T21:05:32.8993354+05:00
```

Preliminary Insertion of Values for Testing

Supplementary 5.sq...ryan Mendoza (60)) Supplementary 4.sq...ryan Mendoza (57))

```sql
USE TinyVideo
GO

INSERT INTO MEMBERSHIP(MEM_NUM, MEM_FNAME, MEM_LNAME, MEM_BALANCE)
    VALUES
    (1003,'Michaelangelo', 'Buonarroti', 1000)

INSERT INTO RENTAL
    VALUES
    (2, '2022-12-02', 1003)

INSERT INTO PRICE(PRICE_CODE, PRICE_RENTALFEE, PRICE_DAILYLATEFEE, PRICE_RENTDAYS)
    VALUES
    ('PRC-002', 300.00, 10.00, 14)

INSERT INTO MOVIE
    VALUES
    (2, 'Spiderman No Way Home', '2021-12-17', 300.00, 'Sci-Fi', 'PRC-002')

INSERT INTO VIDEO(VID_NUM, MOVIE_NUM, VID_STATUS)
    VALUES
    (1, 2, 'IN')
```

82 %

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

Completion time: 2022-12-01T21:18:16.1892178+08:00
```

Transaction Execution: When the Video exists and is ‘IN’

Supplementary 5.sq...ryan Mendoza (60)) | Supplementary 4.sq...ryan Mendoza (57)) | SQLQuery1.sql - DE...ryan Me

```sql
USE TinyVideo
GO

SELECT * FROM MEMBERSHIP
SELECT * FROM RENTAL
SELECT * FROM PRICE
SELECT * FROM MOVIE
SELECT * FROM VIDEO

EXEC uspInsertDetailRental
    @RENT_NUM = 2,
    @VID_NUM = 1
GO

SELECT * FROM DETAILRENTAL
```

82 %

Results | Messages

	MEM_NUM	MEM_FNAME	MEM_LNAME	MEM_STREET	MEM_CITY	MEM_STATE	MEM_ZIP	MEM_BALANCE
1	1001	Leonardo	Da Vinci	NULL	NULL	NULL	NULL	600
2	1002	Pablo	Picasso	NULL	NULL	NULL	NULL	300
3	1003	Michaelangelo	Buonarroti	NULL	NULL	NULL	NULL	1000

	RENT_NUM	RENT_DATE	MEM_NUM
1	1	2022-11-30 00:00:00.000	1001
2	2	2022-12-02 00:00:00.000	1003

	PRICE_CODE	PRICE_DESCRIPTION	PRICE_RENTALFEE	PRICE_DAILYLATEFEE	PRICE_RENTDAYS
1	PRC-001	NULL	NULL	NULL	NULL
2	PRC-002	NULL	300.00	10.00	14

	MOVIE_NUM	MOVIE_TITLE	MOVIE_YEAR	MOVIE_COST	MOVIE_GENRE	PRICE_CODE
1	1	Avengers Infinity War	2018-04-25	280.00	Sci-Fi	PRC-001
2	2	Spiderman No Way Home	2021-12-17	300.00	Sci-Fi	PRC-002

	VID_NUM	VID_INDATE	MOVIE_NUM	VID_STATUS
1	1	NULL	2	IN

	RENT_NUM	VID_NUM	DETAIL_FEE	DETAIL_DUEDATE	DETAIL_RETURNDATE	DETAIL_DAILYLATEFEE
1	2	1	300.00	2022-12-15 21:21:43.380	NULL	10.00

```
Supplementary 5.sq...ryan Mendoza (60))        Supplementary 4.sq...ry

USE TinyVideo
GO

SELECT * FROM MEMBERSHIP
SELECT * FROM RENTAL
SELECT * FROM PRICE
SELECT * FROM MOVIE
SELECT * FROM VIDEO

EXEC uspInsertDetailRental
    @RENT_NUM = 2,
    @VID_NUM = 1
GO

SELECT * FROM DETAILRENTAL
```

82 %

Results | Messages

```
(3 rows affected)

(2 rows affected)

(2 rows affected)

(2 rows affected)

(1 row affected)
The Video Renting is successful. Thank you.

(1 row affected)

Completion time: 2022-12-01T21:21:43.5621099+08:00
```

Observation:

On this part, the query was successful since the video requested exists and the video status is 'IN' the transaction was successful and the DETAIL_DUEDATE assigned to the RENTALDETAIL table was incremented by 14 days (PRICE_RENTDAYS) from the system date.

Transaction Execution: When Video exists but Not ‘IN’

Supplementary 5.sq...ryan Mendoza (60)) | Supplementary 4.sq...ryan Mendoza (57)) | SQLQuery1.sql - DE...ryan Me

```
USE TinyVideo
GO

SELECT * FROM MEMBERSHIP
SELECT * FROM RENTAL
SELECT * FROM PRICE
SELECT * FROM MOVIE
SELECT * FROM VIDEO

EXEC uspInsertDetailRental
    @RENT_NUM = 2,
    @VID_NUM = 1
GO

SELECT * FROM DETAILRENTAL
```

82 %

Results | Messages

	MEM_NUM	MEM_FNAME	MEM_LNAME	MEM_STREET	MEM_CITY	MEM_STATE	MEM_ZIP	MEM_BALANCE
1	1001	Leonardo	Da Vinci	NULL	NULL	NULL	NULL	600
2	1002	Pablo	Picasso	NULL	NULL	NULL	NULL	300
3	1003	Michaelangelo	Buonarroti	NULL	NULL	NULL	NULL	1000

	RENT_NUM	RENT_DATE	MEM_NUM
1	1	2022-11-30 00:00:00.000	1001
2	2	2022-12-02 00:00:00.000	1003

	PRICE_CODE	PRICE_DESCRIPTION	PRICE_RENTALFEE	PRICE_DAILYLATEFEE	PRICE_RENTDAYS
1	PRC-001	NULL	NULL	NULL	NULL
2	PRC-002	NULL	300.00	10.00	14

	MOVIE_NUM	MOVIE_TITLE	MOVIE_YEAR	MOVIE_COST	MOVIE_GENRE	PRICE_CODE
1	1	Avengers Infinity War	2018-04-25	280.00	Sci-Fi	PRC-001
2	2	Spiderman No Way Home	2021-12-17	300.00	Sci-Fi	PRC-002

	VID_NUM	VID_INDATE	MOVIE_NUM	VID_STATUS
1	1	NULL	2	OUT

	RENT_NUM	VID_NUM	DETAIL_FEE	DETAIL_DUEDATE	DETAIL_RETURNDATE	DETAIL_DAILYLATEFEE
1	2	1	300.00	2022-12-15 21:21:43.380	NULL	10.00

Supplementary 5.sq...ryan Mendoza (60)) Supplementary 4.sq...

```sql
USE TinyVideo
GO

SELECT * FROM MEMBERSHIP
SELECT * FROM RENTAL
SELECT * FROM PRICE
SELECT * FROM MOVIE
SELECT * FROM VIDEO

EXEC uspInsertDetailRental
    @RENT_NUM = 2,
    @VID_NUM = 1
GO

SELECT * FROM DETAILRENTAL
```

82 %

Results | Messages

```
(3 rows affected)

(2 rows affected)

(2 rows affected)

(2 rows affected)

(1 row affected)
The Video needs to be returned before renting again.

(1 row affected)

Completion time: 2022-12-01T21:24:50.0733295+08:00
```

Observation:

On this part, the query provided an existing video on the VIDEO table but the video status is current 'OUT' therefore, the rental was not successful since it is currently rented by others.

Transaction Execution: When The Video does not Exist

Supplementary 5.sq...ryan Mendoza (60)) Supplementary 4.sq...ryan Mendoza (57))

```sql
USE TinyVideo
GO

SELECT * FROM MEMBERSHIP
SELECT * FROM RENTAL
SELECT * FROM PRICE
SELECT * FROM MOVIE
SELECT * FROM VIDEO

EXEC uspInsertDetailRental
    @RENT_NUM = 2,
    @VID_NUM = 2
GO

SELECT * FROM DETAILRENTAL
```

82 %

Results Messages

```
(3 rows affected)

(2 rows affected)

(2 rows affected)

(2 rows affected)

(1 row affected)
The Video does not Exist.

(1 row affected)

Completion time: 2022-12-01T21:41:59.9909717+08:00
```

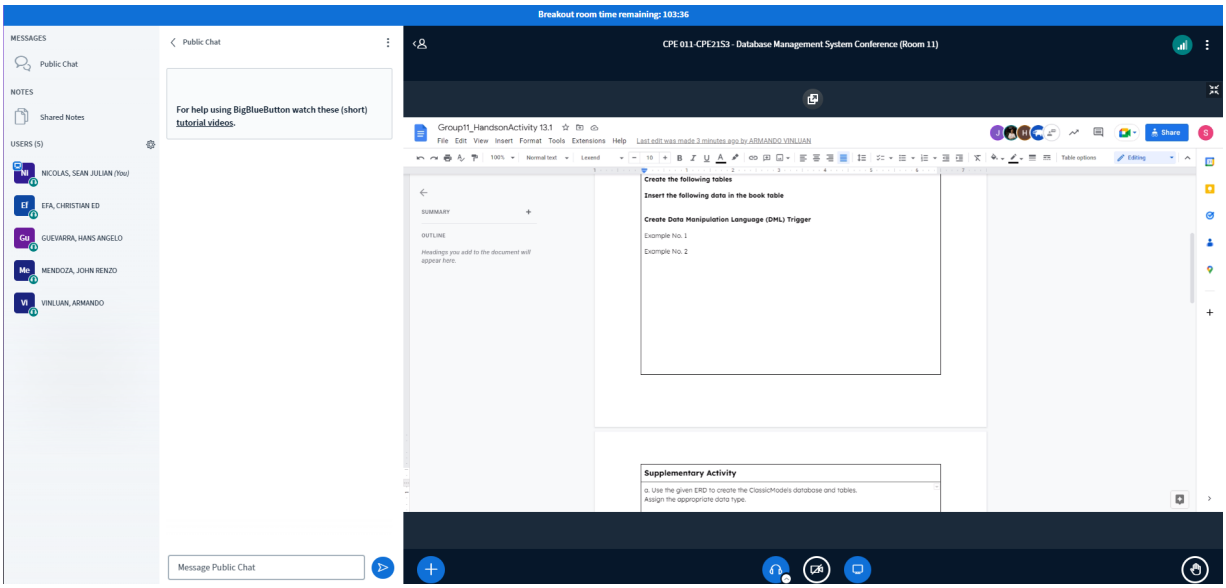
Observation:

On this part, the video on the query does not exist on the VIDEO table and therefore not allowing the rental to progress. In addition, it resulted in an error.

Conclusion

Transactions are critical for ensuring database integrity. When various related activities are carried out simultaneously or when numerous users are interacting with a database at once, they are utilized to protect integrity. In order to maintain a consistent data set, many database uses necessitate storing data to numerous tables or multiple rows to the same table. Through the use of transactions, it is ensured that other connections to the same database either view all the updates or none at all.

Proof of Collaboration



Honor Pledge

"I accept responsibility for my role in ensuring the integrity of the work submitted by the group in which I participated."