**Step 4**

**Name: Vaishnavi Pravin Apsingkar**

**NN: 142**

**#CONSTRAINT:**

**Constraint Syntax:**

[ CONSTRAINT constraint\_name ]

{

{ PRIMARY KEY | UNIQUE }

[ CLUSTERED | NONCLUSTERED ]

(column [ ASC | DESC ] [ ,...n ] )

[ WITH FILLFACTOR = fillfactor

[ WITH ( <index\_option>[ , ...n ] ) ]

[ ON { partition\_scheme\_name ( partition\_column\_name ... )

| filegroup | "default" } ]

| FOREIGN KEY

( column [ ,...n ] )

REFERENCES referenced\_table\_name [ ( ref\_column [ ,...n ] ) ]

[ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]

[ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]

[ NOT FOR REPLICATION ]

| CONNECTION

( { node\_table TO node\_table }

[ , {node\_table TO node\_table }]

[ , ...n ]

)

[ ON DELETE { NO ACTION | CASCADE } ]

| DEFAULT constant\_expression FOR column [ WITH VALUES ]

| CHECK [ NOT FOR REPLICATION ] ( logical\_expression )

}

**Working of constraints:**

Constraints in SQL are like rules for the data in a table. They make sure the data to be inserted/updated in certain conditions and help keep the data accurate and organized. When data is inserted or updated in a table, the DBMS checks if it satisfies the constraints defined on that table using check condition from constraints.

**Constraint Query:**

ALTER TABLE Expense

ADD CONSTRAINT CK\_ExpenseType142

CHECK (ExpenseType IN ('L', 'R')

OR

ExpenseType IS NULL);

**Result:**

Commands completed successfully.

**Tests for constraint:**

**1.Test 1**

**Query:**

Use KOT142;

INSERT INTO Expense (TripID, ExpenseNumber, ExpenseType, Date)

VALUES ('T142', 1 , 'A' , '2024-03-07' );

**Result:**

Msg 547, Level 16, State 0, Line 3

The INSERT statement conflicted with the CHECK constraint "CK\_ExpenseType142". The conflict occurred in database "KOT142", table "dbo.Expense", column 'ExpenseType'.

The statement has been terminated.

=>The test terminated because the condition does not satisfy.

**2. Test 2**

**Query:**

Use KOT142;

INSERT INTO Expense (TripID, ExpenseNumber, ExpenseType, Date)

VALUES ('T001', 2 , 'L' , '2024-03-07' );

**Result:**

(1 row affected)

=>The data inserted because the condition was satisfied.

(removed the added row from db)

**#TRIGGER**

**Trigger Syntax:**

CREATE [ OR ALTER ] TRIGGER [ schema\_name . ]trigger\_name

ON { table | view }

[ WITH <dml\_trigger\_option> [ ,...n ] ]

{ FOR | AFTER | INSTEAD OF }

{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }

[ WITH APPEND ]

[ NOT FOR REPLICATION ]

AS {sql\_statement [ ; ] [ ,...n ] | EXTERNAL NAME <method specifier [ ; ] > }

**Working of triggers:**

Triggers are executed when events like insert, update or delete operation is carried out on table.

**Trigger Query:**

CREATE TRIGGER FuelMax142

ON RoadExpenses

FOR INSERT

AS

BEGIN

DECLARE @Cost Money;

DECLARE @errorMessage CHAR(70);

SELECT @Cost = ins.FuelCost

FROM inserted ins;

IF @Cost > 200

BEGIN

SET @errorMessage = 'Failure on insert. FuelCost cannot exceed 200. NN:142';

BEGIN TRY

BEGIN TRANSACTION;

RAISERROR(@errorMessage, 16, 1);

ROLLBACK TRANSACTION;

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

BEGIN

ROLLBACK TRANSACTION;

END

PRINT @errorMessage;

END CATCH

END

END;

**Results:**

Commands completed successfully.

**Tests for trigger:**

(removed last entry for T012, from RoadExpenses Table, for testing purpose)

**1. Test 1**

**Query:**

Use KOT142;

INSERT INTO RoadExpenses (TripID, ExpenseNumber, FuelGallons, FuelCost)

VALUES ('T012', 2, 12, 280);

**Result:**

Failure on insert. FuelCost cannot exceed 200. NN:142

Msg 3609, Level 16, State 1, Line 3

The transaction ended in the trigger. The batch has been aborted.

=>This gave error, as Cost>200.

**2. Test 2**

**Query:**

Use KOT142;

INSERT INTO RoadExpenses (TripID, ExpenseNumber, FuelGallons, FuelCost)

VALUES ('T012', 2, 10, 38.5000);

**Result:**

(1 row affected)

=>This successfully updated the table by inserting the given data, as Cost<200.

**#Stored Procedure**

**Syntax:**

CREATE [ OR ALTER ] { PROC | PROCEDURE }

[schema\_name.] procedure\_name [ ; number ]

[ { @parameter\_name [ type\_schema\_name. ] data\_type }

[ VARYING ] [ NULL ] [ = default ] [ OUT | OUTPUT | [READONLY]

] [ ,...n ]

[ WITH <procedure\_option> [ ,...n ] ]

[ FOR REPLICATION ]

AS { [ BEGIN ] sql\_statement [;] [ ...n ] [ END ] }

[;]

<procedure\_option> ::=

[ ENCRYPTION ]

[ RECOMPILE ]

[ EXECUTE AS Clause ]

**Working of stored procedure:**

Stored procedure has one or more SQL statements which can be reused.

**Query:**

CREATE PROCEDURE usp\_Truck142

@TruckID CHAR(5)

AS

BEGIN

SELECT MAX(IncomingMiles) AS HighestIncMileage

FROM Trip

WHERE TruckID = @TruckID;

END;

**Result:**

Commands completed successfully.

**Tests for stored procedure:**

**1. Test 1**

**Query:**

EXEC usp\_Truck142 @TruckID = 'B-002';

**Result:**

HighestIncMileage

-----------------

NULL

(1 row affected)

=>If we give TruckID not available in the table, it returns NULL value and does not produce error.

**2. Test 2**

**Query:**

EXEC usp\_Truck142 @TruckID = 'B-019';

**Result:**

HighestIncMileage

-----------------

27200

(1 row affected)

**3. Test 3**

**Query:**

EXEC usp\_Truck142 @TruckID = 'B-001';

**Result:**

HighestIncMileage

-----------------

3700

(1 row affected)

=>If multiple entries are present, it returns corresponding highest(maximum) values of incoming miles.

**4. Test 4**

**Query:**

EXEC usp\_Truck142 @TruckID = '';

**Result:**

HighestIncMileage

-----------------

NULL

(1 row affected)

=>If checked for empty values, it returns null and does not produce error.