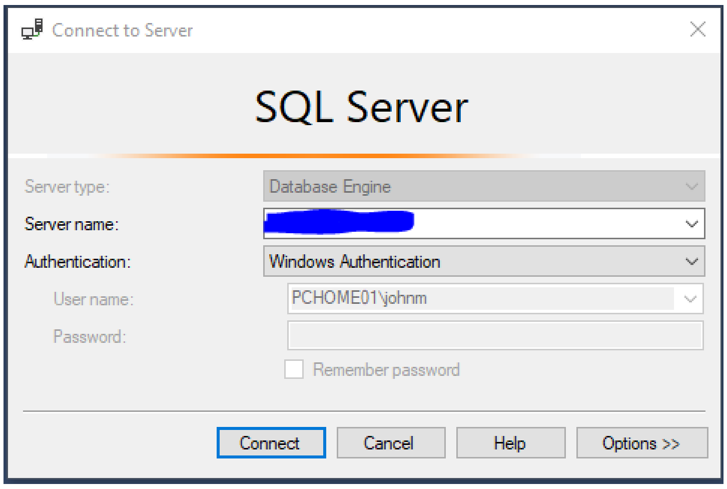
DATABASE PRACTICE.

SECTION A: Create a new database in SSMS

1) If you are working with an online SQL Server and not the SSMS, then drop all tables created from previous lessons so you can start with a clean database. Then proceed to step 2d. Depending on the online version you are using, the pictures displayed below may look differently.

2) If you are working with SSMS then follow these steps:
   a) Start SSMS studio and sign in to your SQL Server using Windows Authentication.



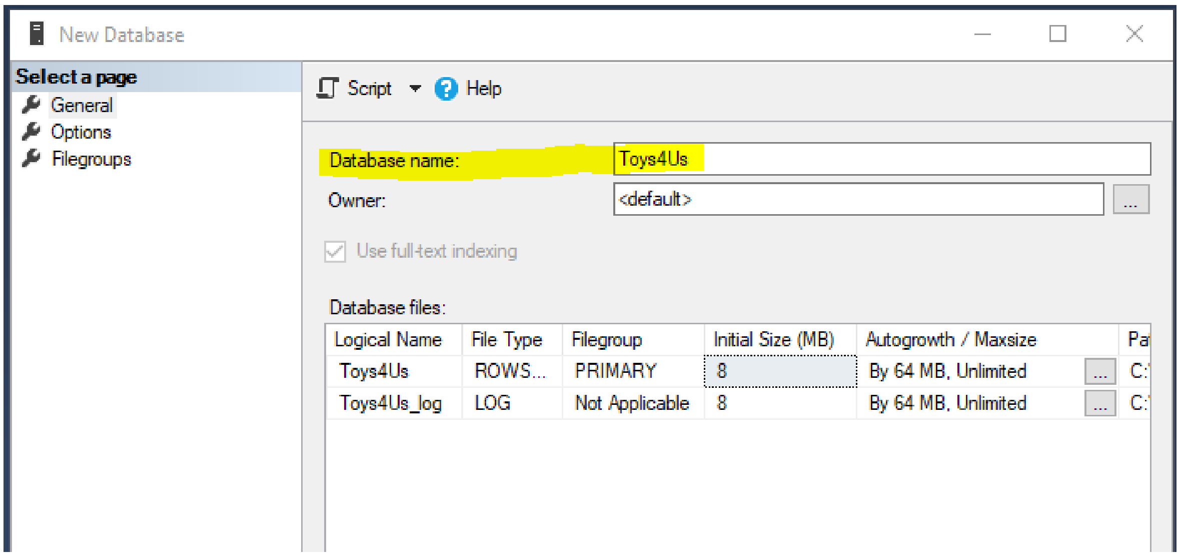
Connect to Server

SQL Server

Server type:	Database Engine
Server name:	[redacted]
Authentication:	Windows Authentication
User name:	PCHOME01\johnm
Password:	

Remember password

   b) **Right Click** on Databases and create a new database with the name **Toys4Us** and press **OK**



New Database

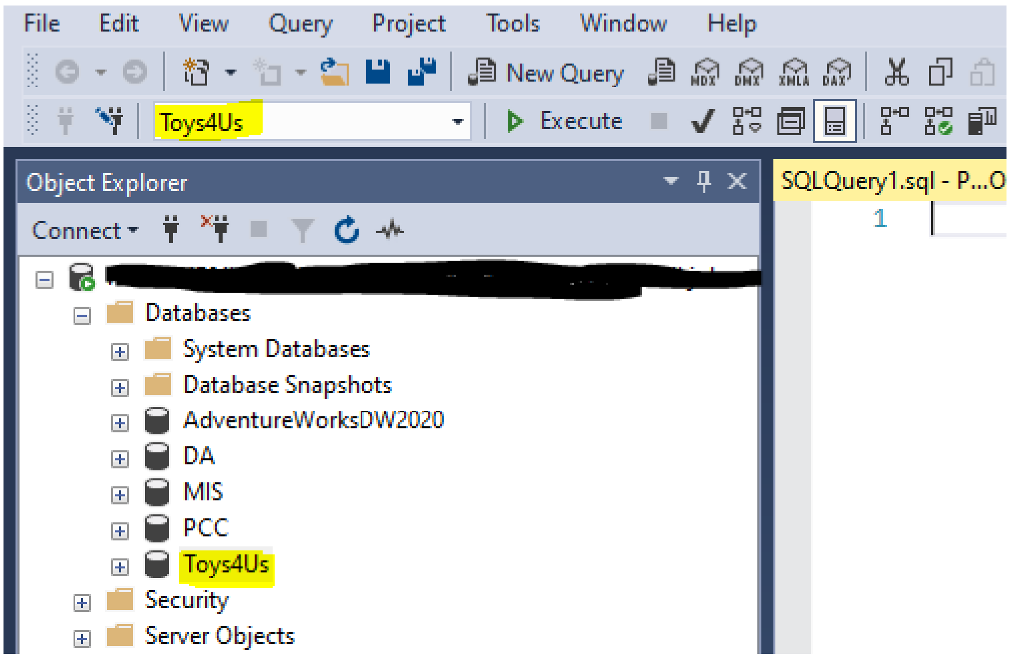
Database name:	Toys4Us
Owner:	<default>

Use full-text indexing

Database files:

Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth / Maxsize	Pa
Toys4Us	ROWS...	PRIMARY	8	By 64 MB, Unlimited	C:
Toys4Us_log	LOG	Not Applicable	8	By 64 MB, Unlimited	C:

   c) On the Toys4Us new database **right click** and press **New Query**. Make sure that Toys4Us is selected in the tool bar. Make absolute certain that every time you create a new query, you ALWAYS check that the Toys4Us is the selected database, otherwise, all entities will be created in a different database. This will not produce the results required.

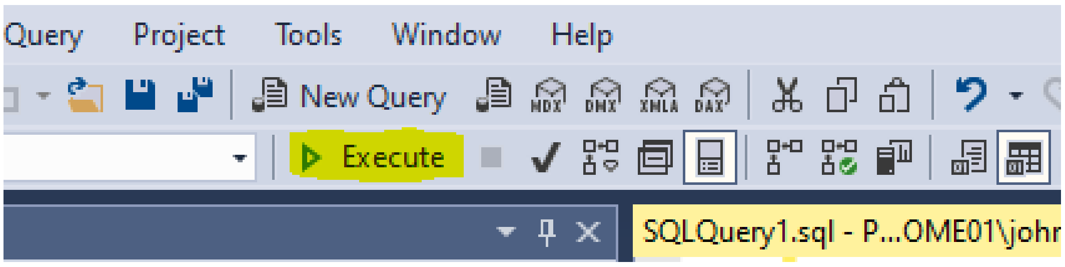


d) Visit my GitHub (https://github.com/gitioannis) and select my **SQL repository**

e) Click on the **CreateTable_tCustomer_Import.sql**

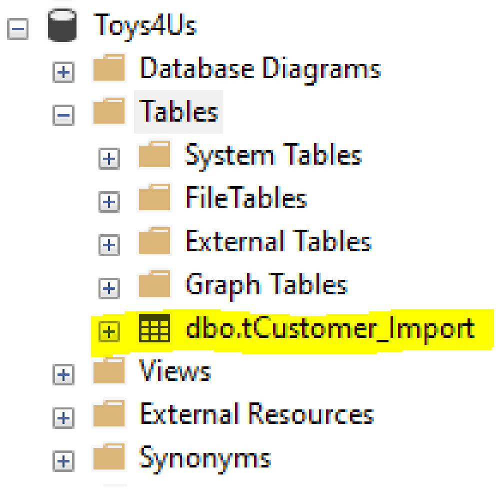
f) Select the code from the repository and paste it in the SSMS query window.

g) To run the code press on the **Execute** button.

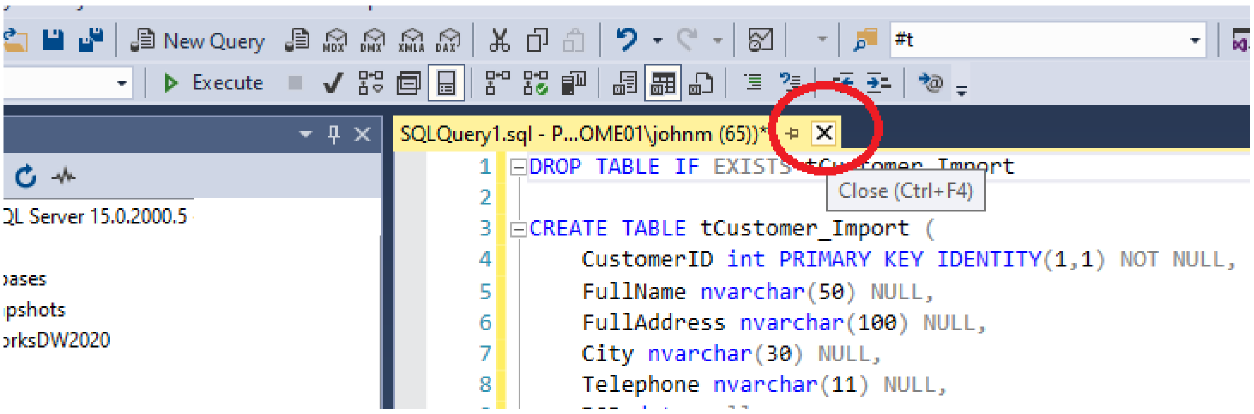


h) Expand the Toys4Us database. Right click on Tables and select **Refresh.**

i) Expand on the Tables section (if not already expanded) and you should be able to see the new table created:



j) Close the open query by pressing on the **X** button as per below. If prompted to save then select No.

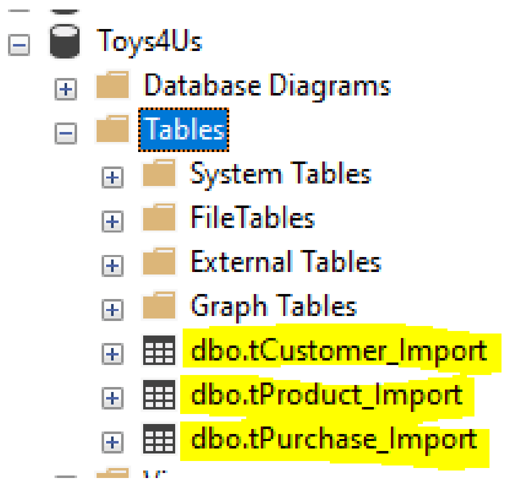


```
DROP TABLE IF EXISTS tCustomer_Import

CREATE TABLE tCustomer_Import (
    CustomerID int PRIMARY KEY IDENTITY(1,1) NOT NULL,
    FullName nvarchar(50) NULL,
    FullAddress nvarchar(100) NULL,
    City nvarchar(30) NULL,
    Telephone nvarchar(11) NULL,
```

k) Select the Toys4Us database. Right click and select **New Query**. Make sure that the Toys4Us database is selected on the tool bar as explained on the previous step 2c.

l) Repeat the same process to create the next two tables from my GitHub:
   i) CreateTable_tProduct_Import
   ii) CreateTable_tPurchase_Import

m) When all are finally refreshed, in the tables section in SSMS then you should have the below structure:



```
Toys4Us
    Database Diagrams
    Tables
        System Tables
        FileTables
        External Tables
        Graph Tables
        dbo.tCustomer_Import
        dbo.tProduct_Import
        dbo.tPurchase_Import
```

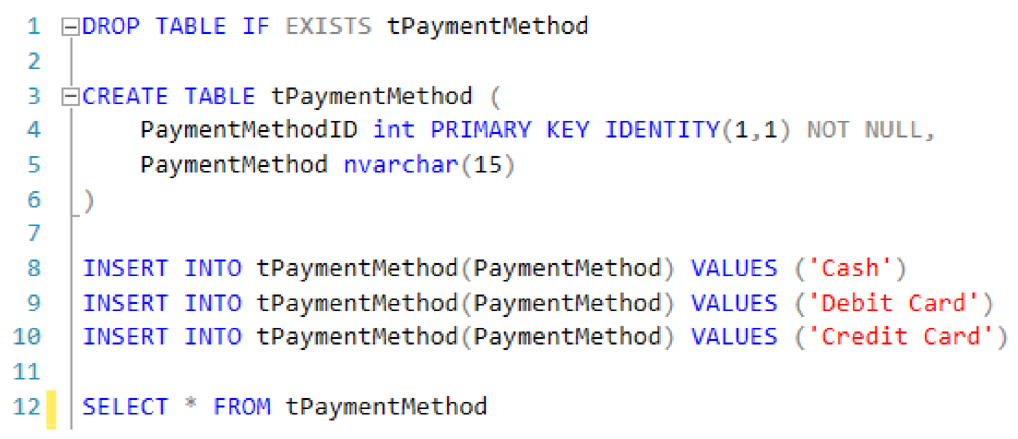
n) For the next section you will need to create a folder in your computer to save your scripts. Call the folder **Toys4UsResources.**

SECTION B: Create table tPaymentMethod.

PaymentMethod	
PaymentMethodID	Paymentmethod
int	nvarchar(15)
1	Cash
2	Debit Card
3	Credit Card

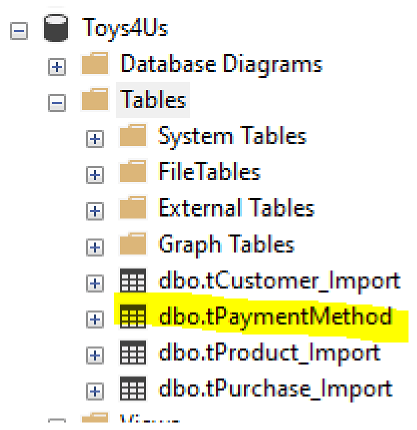
- Table specifications:
  - PaymentMethodID field:
    - Primary Key
    - Will be int type.
    - Will not accept null values.
    - Will auto-increment by 1 starting with 1.
  - PaymentMethod field:
    - Will accept nvarchar type.
    - Max character length 15.
- Actions:

1. Create a new query. Make sure that Toys4Us database is selected in the toolbar as previously demonstrated.
2. Write a script that will drop the table if the table exists (Line 1)
3. Create the table with the above specifications. (Lines 3-6)
4. Write the script to insert the values as shown above. (Lines 8-10)
5. Write a script that will list all the table values. (Line 12)



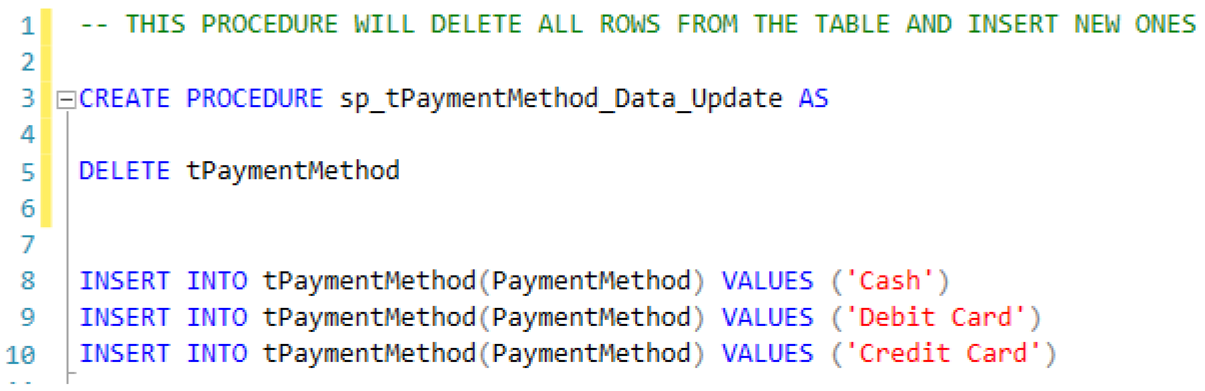
```
1  DROP TABLE IF EXISTS tPaymentMethod
2
3  CREATE TABLE tPaymentMethod (
4      PaymentMethodID int PRIMARY KEY IDENTITY(1,1) NOT NULL,
5      PaymentMethod nvarchar(15)
6  )
7
8  INSERT INTO tPaymentMethod(PaymentMethod) VALUES ('Cash')
9  INSERT INTO tPaymentMethod(PaymentMethod) VALUES ('Debit Card')
10 INSERT INTO tPaymentMethod(PaymentMethod) VALUES ('Credit Card')
11
12 SELECT * FROM tPaymentMethod
```

6. Execute the above statement.
7. Refresh the tables section. A new tPaymentMethod table is now created.



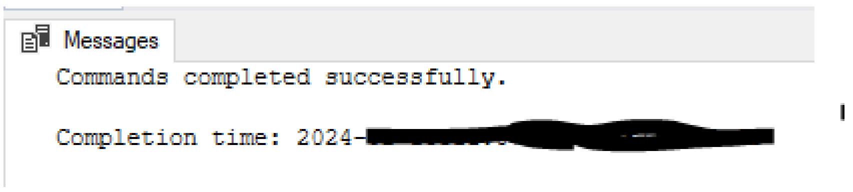
```
⊟ Toys4Us
    ⊞ Database Diagrams
    ⊟ Tables
        ⊞ System Tables
        ⊞ FileTables
        ⊞ External Tables
        ⊞ Graph Tables
        ⊞ dbo.tCustomer_Import
        ⊞ dbo.tPaymentMethod
        ⊞ dbo.tProduct_Import
        ⊞ dbo.tPurchase_Import
```

8. Save all the above actions into a single SQL script file on your local hard drive Toys4UsResources folder with the name *CreateTable_tPaymentMethod.sql*
9. You will now create a stored procedure with a name *sp_tPaymentMethod_Data_Update*
10. This procedure will delete the data from the tPaymentMethod table and re-insert the data again.
11. In the query window keep lines 8-10 and delete everything else and type the code below:



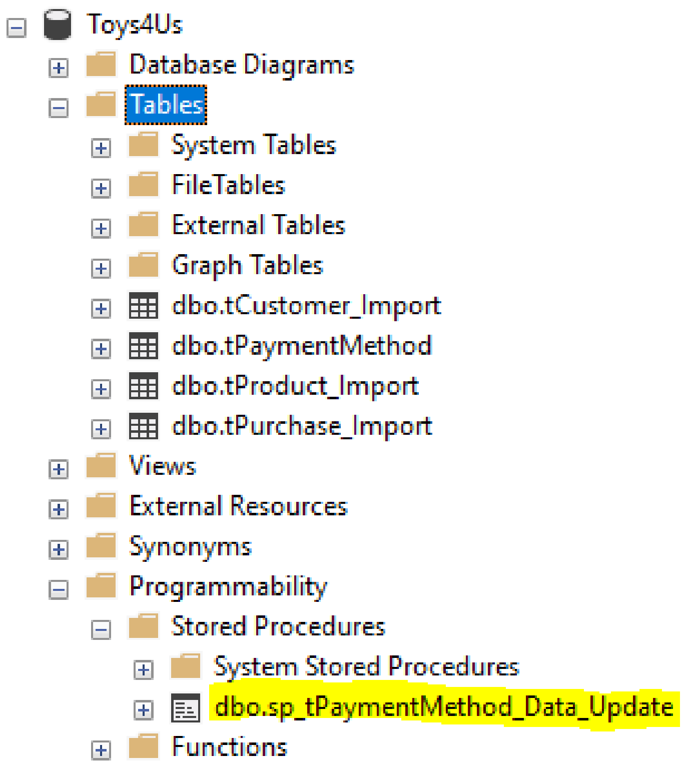
```sql
1   -- THIS PROCEDURE WILL DELETE ALL ROWS FROM THE TABLE AND INSERT NEW ONES
2
3   CREATE PROCEDURE sp_tPaymentMethod_Data_Update AS
4
5   DELETE tPaymentMethod
6
7
8   INSERT INTO tPaymentMethod(PaymentMethod) VALUES ('Cash')
9   INSERT INTO tPaymentMethod(PaymentMethod) VALUES ('Debit Card')
10  INSERT INTO tPaymentMethod(PaymentMethod) VALUES ('Credit Card')
```

12. Execute the code. In the messages section you will get the response:



```
Messages
Commands completed successfully.

Completion time: 2024-[redacted]
```

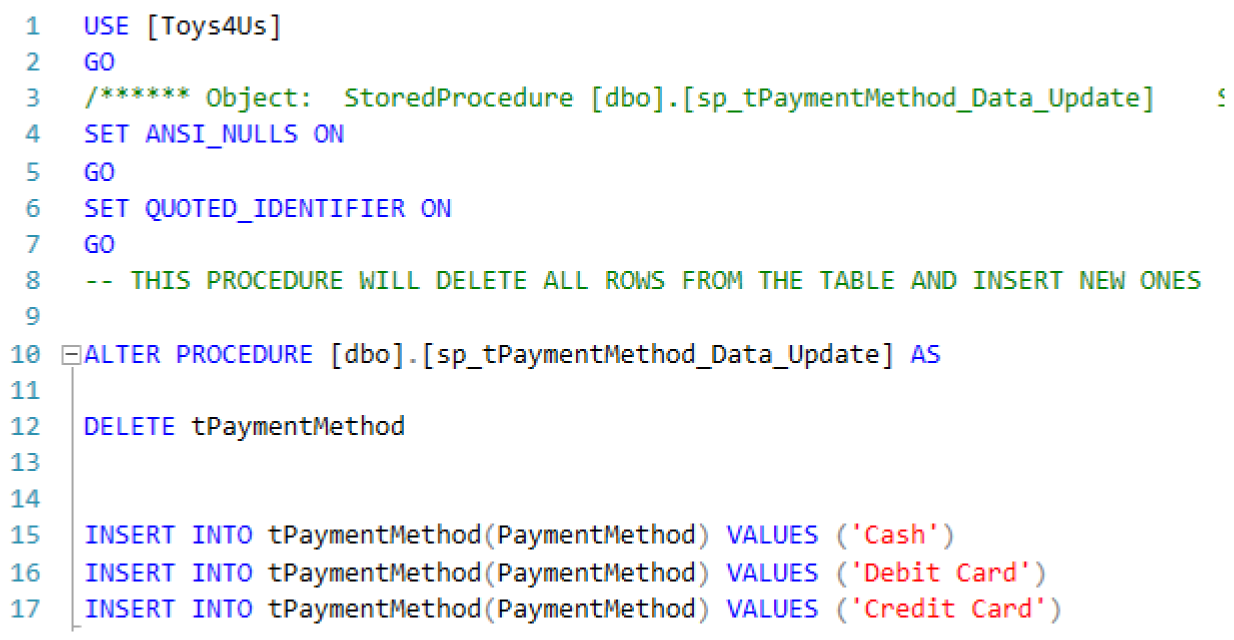
13. Expand the Programmability section and then the Stored Procedures section. Right click on Stored Procedures section and Refresh. You should now have a new stored procedure as follows:



```
⊟ Toys4Us
    ⊞ Database Diagrams
    ⊟ Tables
        ⊞ System Tables
        ⊞ FileTables
        ⊞ External Tables
        ⊞ Graph Tables
        ⊞ dbo.tCustomer_Import
        ⊞ dbo.tPaymentMethod
        ⊞ dbo.tProduct_Import
        ⊞ dbo.tPurchase_Import
    ⊞ Views
    ⊞ External Resources
    ⊞ Synonyms
    ⊟ Programmability
        ⊟ Stored Procedures
            ⊞ System Stored Procedures
            ⊞ dbo.sp_tPaymentMethod_Data_Update
        ⊞ Functions
```

14. To modify the contents of the stored procedure, right click on sp_tPaymentMethod_Data_Update and select **Modify**

15. SSMS studio will produce the below query:



```
1   USE [Toys4Us]
2   GO
3   /****** Object:  StoredProcedure [dbo].[sp_tPaymentMethod_Data_Update]    S
4   SET ANSI_NULLS ON
5   GO
6   SET QUOTED_IDENTIFIER ON
7   GO
8   -- THIS PROCEDURE WILL DELETE ALL ROWS FROM THE TABLE AND INSERT NEW ONES
9
10  ALTER PROCEDURE [dbo].[sp_tPaymentMethod_Data_Update] AS
11
12  DELETE tPaymentMethod
13
14
15  INSERT INTO tPaymentMethod(PaymentMethod) VALUES ('Cash')
16  INSERT INTO tPaymentMethod(PaymentMethod) VALUES ('Debit Card')
17  INSERT INTO tPaymentMethod(PaymentMethod) VALUES ('Credit Card')
```

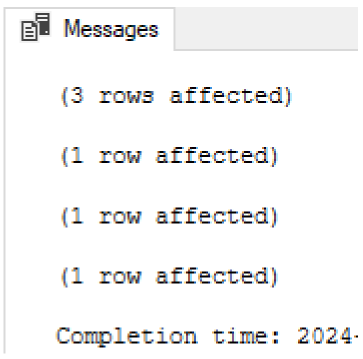
16. All the code from line 1 to line 7 has been automatically added by the system. Above the ALTER PROCEDURE statement enter the below comment:

    -- Stored Procedure created by: <*Your name*>

17. Note the lines 1 and 2 tell SSMS to select the correct database to make the modifications. This is a good practice to adapt in your coding.
18. Line 10 has now been changed to ALTER from CREATE. This is because you are not creating a new procedure. You just modify some of the code.
19. **Execute** the code to save the changes.
20. To run the code, create a new query window under the Toys4Us database and type the statement:

    EXECUTE sp_tPaymentMethod_Data_Update

21. You should get the confirmation in the Messages section:



```
(3 rows affected)

(1 row affected)

(1 row affected)

(1 row affected)

Completion time: 2024
```

(3 rows affected) is the message to confirm that 3 rows have been deleted

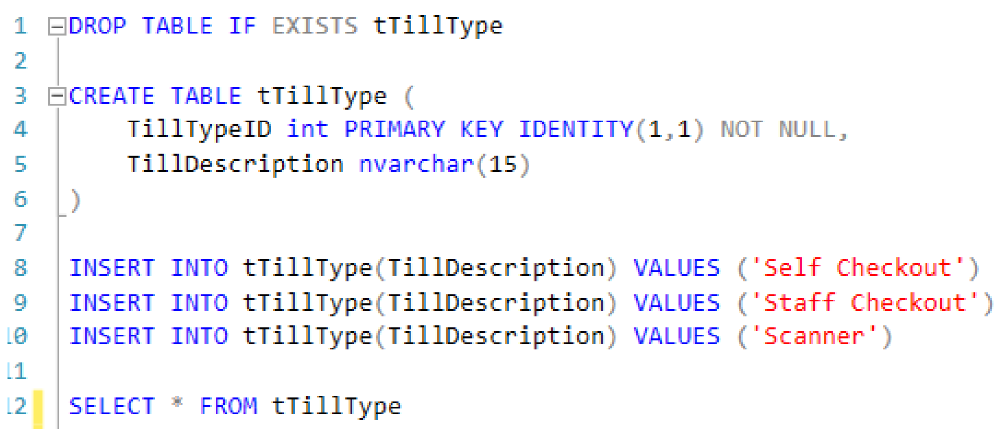
(1 row affected) is the message to confirm that the insert statement has added a new row.

You will now repeat all the above statements to create the two other tables in section C and D.

SECTION C: Create table tTillType:

TillType	
TillTypeID	TillDescription
int	nvarchar(15)
1	Self Checkout
2	Staff Checkout
3	Scanner

- Table specifications:
    - TillTypeID field:
        - Primary Key
        - Will be int type.
        - Will not accept null values.
        - Will auto-increment by 1 starting with 1.
    - TillDescription field:
        - Will accept nvarchar type.
        - Max character length 15.
- Actions:
    1. Write a script that will drop the table if the table exists
    2. Create the table with the above specifications.
    3. Write the script to insert the values as shown above.
    4. Write a script that will list all the table values.



```sql
DROP TABLE IF EXISTS tTillType

CREATE TABLE tTillType (
    TillTypeID int PRIMARY KEY IDENTITY(1,1) NOT NULL,
    TillDescription nvarchar(15)
)

INSERT INTO tTillType(TillDescription) VALUES ('Self Checkout')
INSERT INTO tTillType(TillDescription) VALUES ('Staff Checkout')
INSERT INTO tTillType(TillDescription) VALUES ('Scanner')

SELECT * FROM tTillType
```

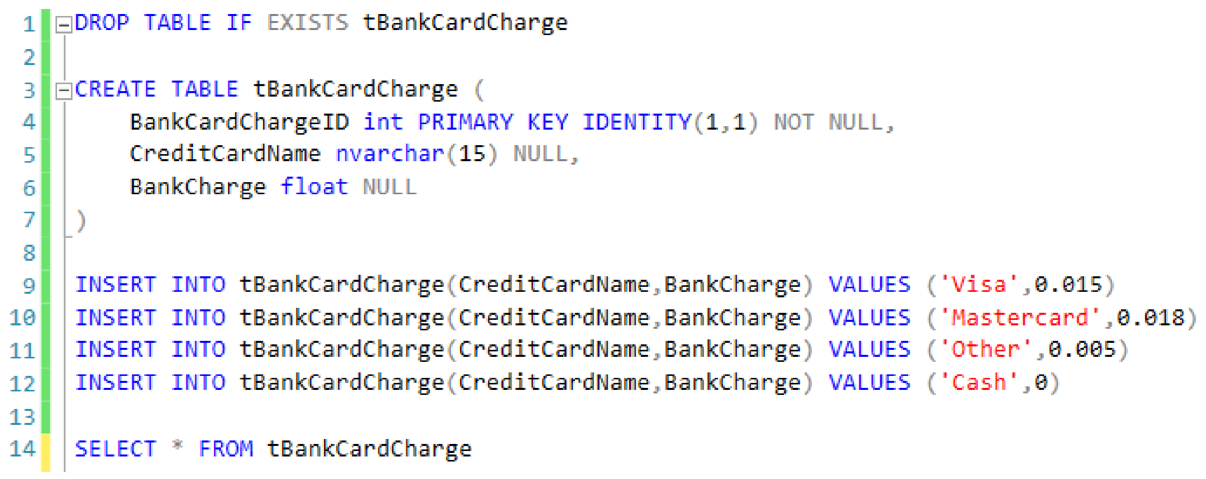
---

5. Save all the above actions into a single SQL script file on your local hard drive folder (as above) with the name *CreateTable_tTillType.sql*
6. Create a stored procedure with a name sp_*tTillType_Data_Update*
7. The stored procedure will delete the previous data and re-insert the values. Think of this procedure as a reset process for STAGE 1.

SECTION D: Create table tBankCardCharge:

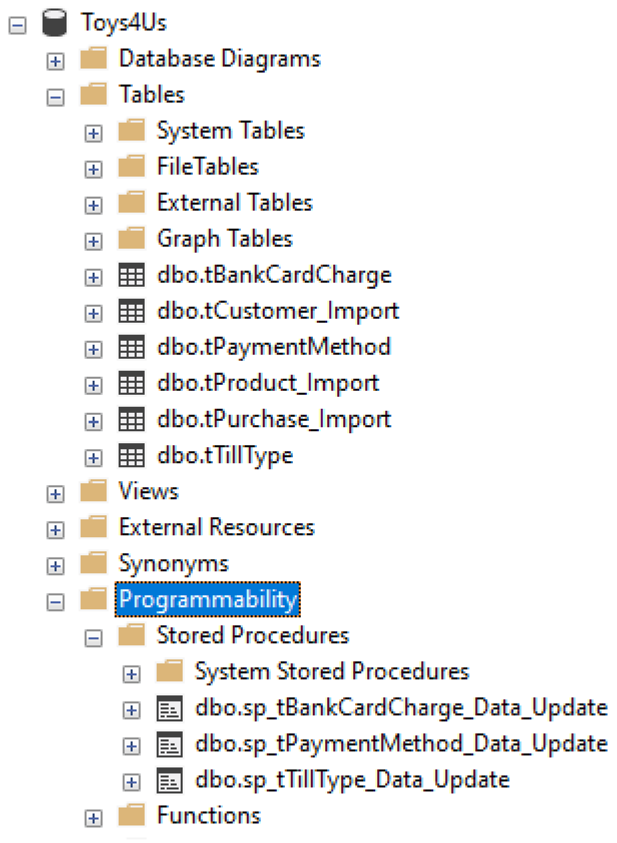
BankCardCharge		
BankCardChargeID	CreditCardName	BankCharge
int	nvarchar(15)	float
1	Visa	0.015
2	Mastercard	0.018
3	Other	0.005
4	Cash	0

- Table specifications:
  - BankCardChargeID field:
    - Primary Key
    - Will be int type.
    - Will not accept null values.
    - Will auto-increment by 1 starting with 1.
  - CreditCardName field:
    - Will accept nvarchar type.
    - Max character length 15.
  - BankCharge
    - Will accept float values.
- Actions:
  - Write a script that will drop the table if the table exists
  - Create the table with the above specifications.
  - Write the script to insert the values as shown above.
  - Write a script that will list all the table values.
  - Save all the above actions into a single SQL script file on your local hard drive with the name *CreateTable_tBankCardCharge.sql*
  - Create a stored procedure with a name sp_*tBankCardCharge_Data_Update*



```sql
DROP TABLE IF EXISTS tBankCardCharge

CREATE TABLE tBankCardCharge (
    BankCardChargeID int PRIMARY KEY IDENTITY(1,1) NOT NULL,
    CreditCardName nvarchar(15) NULL,
    BankCharge float NULL
)

INSERT INTO tBankCardCharge(CreditCardName,BankCharge) VALUES ('Visa',0.015)
INSERT INTO tBankCardCharge(CreditCardName,BankCharge) VALUES ('Mastercard',0.018)
INSERT INTO tBankCardCharge(CreditCardName,BankCharge) VALUES ('Other',0.005)
INSERT INTO tBankCardCharge(CreditCardName,BankCharge) VALUES ('Cash',0)

SELECT * FROM tBankCardCharge
```

At this stage when all resources are created, and the database is refreshed you should have a structure as per below:



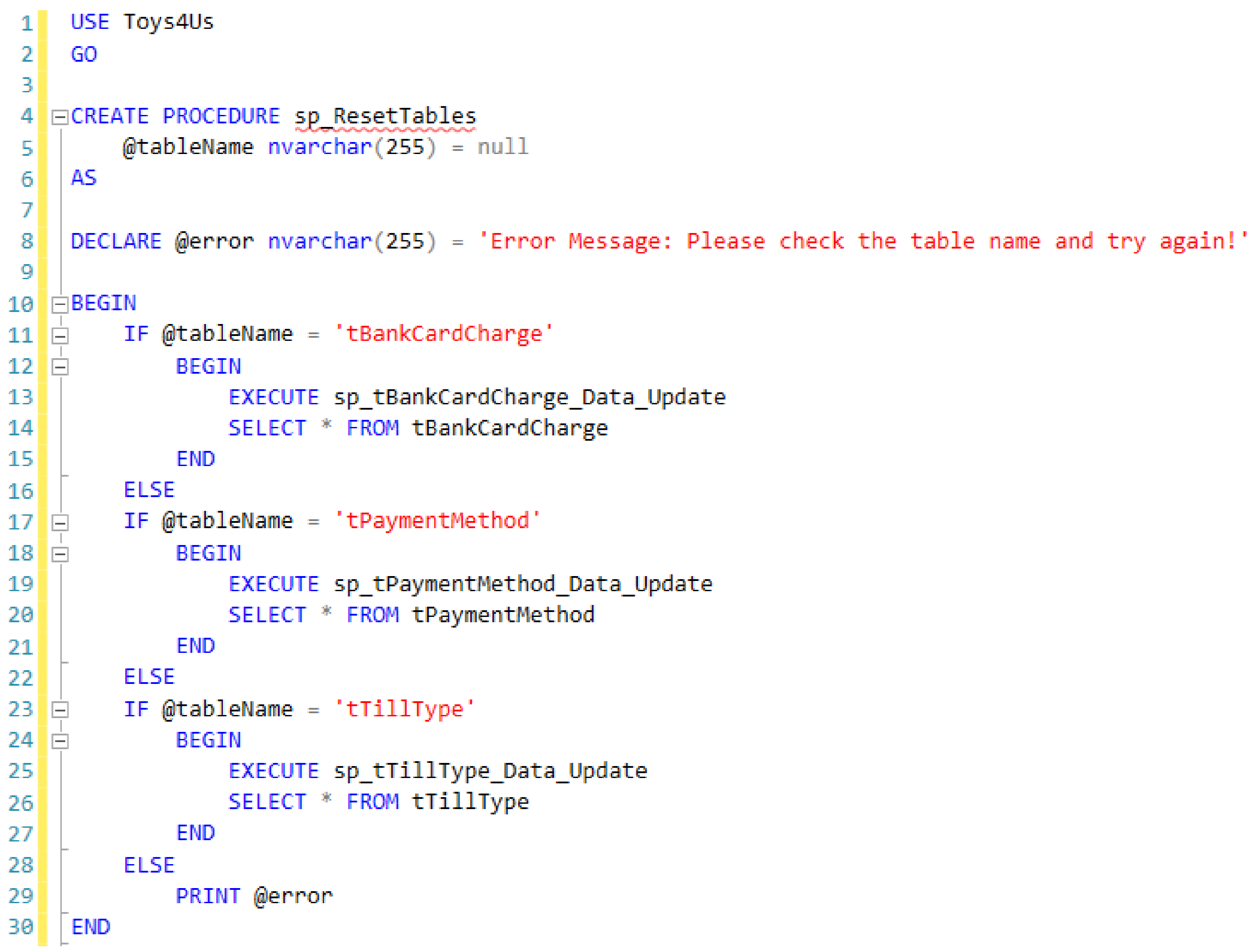
```
Toys4Us
    Database Diagrams
    Tables
        System Tables
        FileTables
        External Tables
        Graph Tables
        dbo.tBankCardCharge
        dbo.tCustomer_Import
        dbo.tPaymentMethod
        dbo.tProduct_Import
        dbo.tPurchase_Import
        dbo.tTillType
    Views
    External Resources
    Synonyms
    Programmability
        Stored Procedures
            System Stored Procedures
            dbo.sp_tBankCardCharge_Data_Update
            dbo.sp_tPaymentMethod_Data_Update
            dbo.sp_tTillType_Data_Update
        Functions
```

In the following section you will create some additional stored procedures to automate your processes.

SECTION E1: Automations

You will now create a new stored procedure that will take as parameters which table to update. If the user enters the wrong table name, then the store procedure will give the user an error message.

1) Create a stored procedure to update selected tables as per below code.
2) Execute the stored procedure.

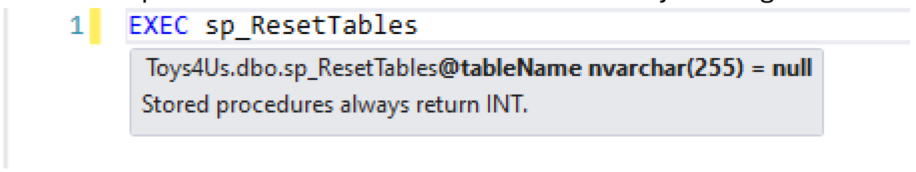


```sql
USE Toys4Us
GO

CREATE PROCEDURE sp_ResetTables
    @tableName nvarchar(255) = null
AS

DECLARE @error nvarchar(255) = 'Error Message: Please check the table name and try again!'

BEGIN
    IF @tableName = 'tBankCardCharge'
        BEGIN
            EXECUTE sp_tBankCardCharge_Data_Update
            SELECT * FROM tBankCardCharge
        END
    ELSE
    IF @tableName = 'tPaymentMethod'
        BEGIN
            EXECUTE sp_tPaymentMethod_Data_Update
            SELECT * FROM tPaymentMethod
        END
    ELSE
    IF @tableName = 'tTillType'
        BEGIN
            EXECUTE sp_tTillType_Data_Update
            SELECT * FROM tTillType
        END
    ELSE
        PRINT @error
END
```

3) In a new query type (EXEC and EXECUTE is the same command) :

   EXEC sp_ResetTables

4) Press the space bar at the end of the statement and you will get the below message:



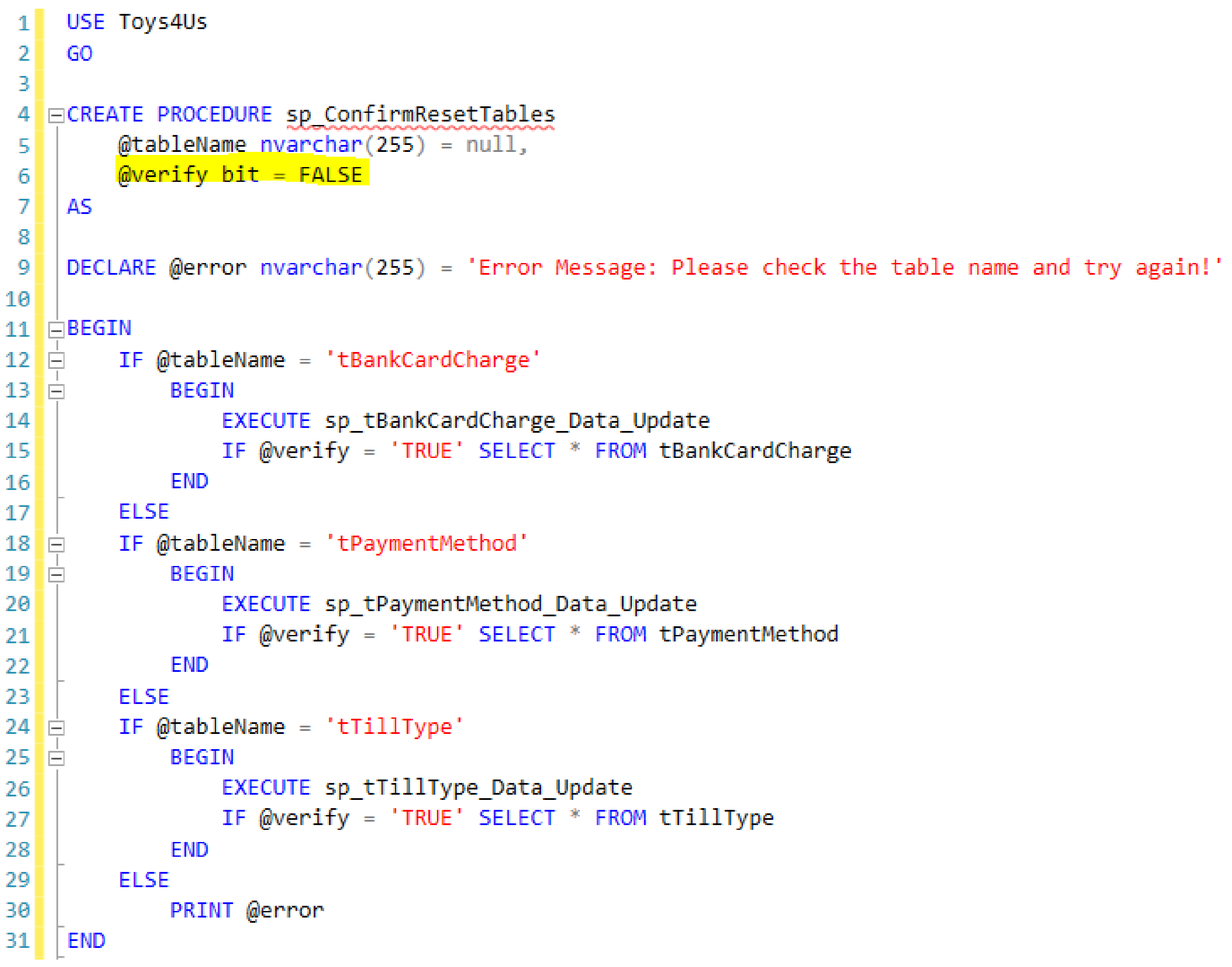
```
EXEC sp_ResetTables
```

Toys4Us.dbo.sp_ResetTables**@tableName nvarchar(255) = null**
Stored procedures always return INT.

5) The system is requesting from you to type the name of the table as a parameter. Type tBankCardCharge and execute
6) Execute the stored procedure without a parameter or by giving the wrong table to test the error message.

SECTION E2: Automations

You will now create a new stored procedure based on the previous one. In the previous section the stored procedure runs the update procedure for the table requested and then runs a select statement to verify the data in that table. This time you will add a new parameter to control if you would like to verify the data at the end of the update.

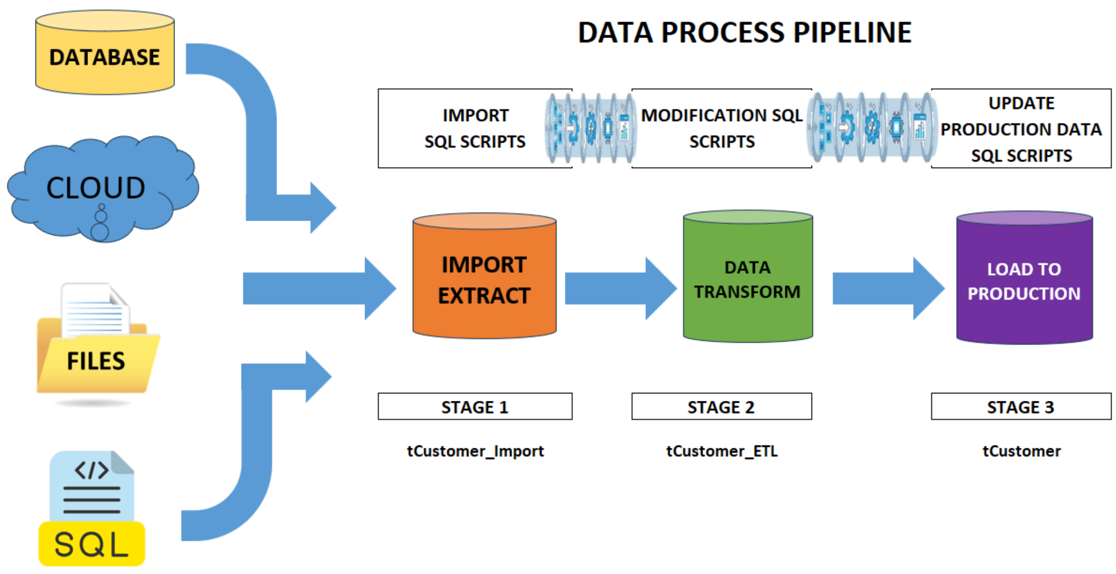


```
USE Toys4Us
GO

CREATE PROCEDURE sp_ConfirmResetTables
    @tableName nvarchar(255) = null,
    @verify bit = FALSE
AS

DECLARE @error nvarchar(255) = 'Error Message: Please check the table name and try again!'

BEGIN
    IF @tableName = 'tBankCardCharge'
        BEGIN
            EXECUTE sp_tBankCardCharge_Data_Update
            IF @verify = 'TRUE' SELECT * FROM tBankCardCharge
        END
    ELSE
    IF @tableName = 'tPaymentMethod'
        BEGIN
            EXECUTE sp_tPaymentMethod_Data_Update
            IF @verify = 'TRUE' SELECT * FROM tPaymentMethod
        END
    ELSE
    IF @tableName = 'tTillType'
        BEGIN
            EXECUTE sp_tTillType_Data_Update
            IF @verify = 'TRUE' SELECT * FROM tTillType
        END
    ELSE
        PRINT @error
END
```

1) In a new query window enter the command: EXEC sp_ConfirmResetTables tBankCardCharge
2) Now delete this command and type: EXEC sp_ConfirmResetTables tBankCardCharge, true

Can you tell the difference?

SECTION E3: Customer transformations and data wrangling.

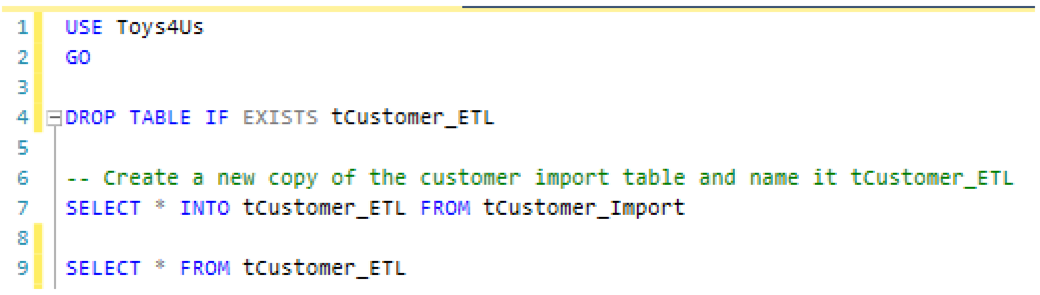


We have discussed about the three stages of the ETL pipeline process in the lesson. In all previous sections you have managed to import and update the data for STAGE 1 using scripts which either created the tables and inserted the data or updated the existing tables with new data. In this section we will create stored procedures to copy the data from the import tables to the ETL tables (when required) and automate the transformation using stored procedures.

The code below (on page 13) performs all the necessary transformations to modify the table structure and perform some data wrangling.

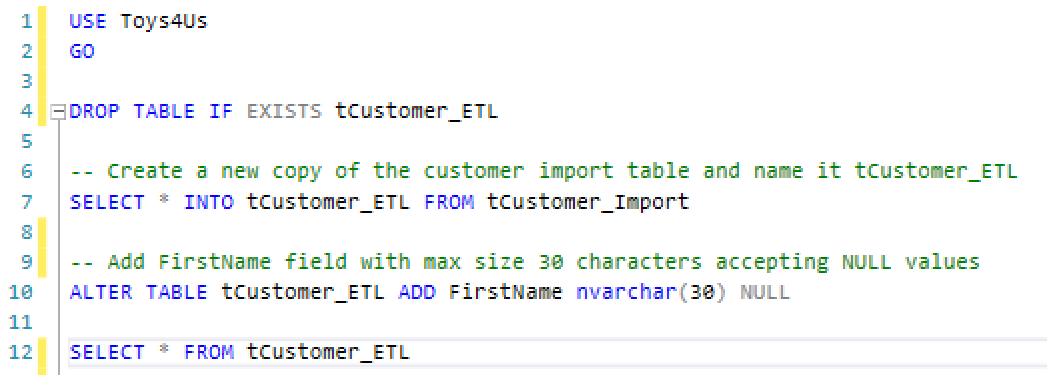
Create a stored procedure named sp_CustomerETL to automate the below script.

For better practice before creating a stored procedure, first start a new query including lines 1 to 7 and then type the select statement from line 55. Run the query. Observe the data in the results section.



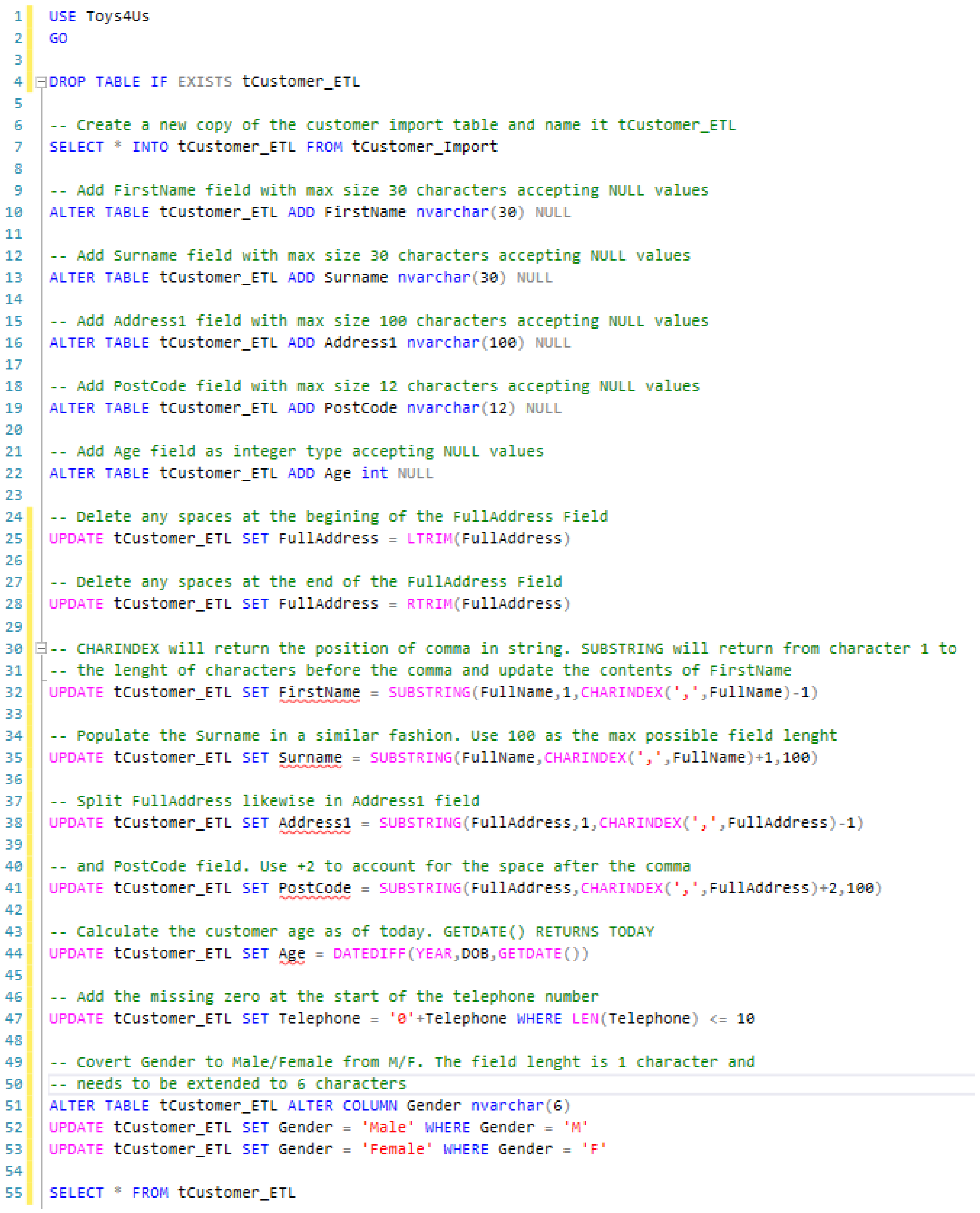
```
1   USE Toys4Us
2   GO
3
4   DROP TABLE IF EXISTS tCustomer_ETL
5
6   -- Create a new copy of the customer import table and name it tCustomer_ETL
7   SELECT * INTO tCustomer_ETL FROM tCustomer_Import
8
9   SELECT * FROM tCustomer_ETL
```

Then introduce lines 9 and 10 from the final code into your code and keep the select statement always as your last line to run. This will allow you to monitor changes as they happen. You always drop the ETL table in line 4 and then making a copy from your import table in line 7. This will allow you to run the code as many times as you wish without errors.



```
1   USE Toys4Us
2   GO
3
4   DROP TABLE IF EXISTS tCustomer_ETL
5
6   -- Create a new copy of the customer import table and name it tCustomer_ETL
7   SELECT * INTO tCustomer_ETL FROM tCustomer_Import
8
9   -- Add FirstName field with max size 30 characters accepting NULL values
10  ALTER TABLE tCustomer_ETL ADD FirstName nvarchar(30) NULL
11
12  SELECT * FROM tCustomer_ETL
```

Once done with the entire code, then add the create stored procedure statement to finalise the process and test.



```sql
USE Toys4Us
GO

DROP TABLE IF EXISTS tCustomer_ETL

-- Create a new copy of the customer import table and name it tCustomer_ETL
SELECT * INTO tCustomer_ETL FROM tCustomer_Import

-- Add FirstName field with max size 30 characters accepting NULL values
ALTER TABLE tCustomer_ETL ADD FirstName nvarchar(30) NULL

-- Add Surname field with max size 30 characters accepting NULL values
ALTER TABLE tCustomer_ETL ADD Surname nvarchar(30) NULL

-- Add Address1 field with max size 100 characters accepting NULL values
ALTER TABLE tCustomer_ETL ADD Address1 nvarchar(100) NULL

-- Add PostCode field with max size 12 characters accepting NULL values
ALTER TABLE tCustomer_ETL ADD PostCode nvarchar(12) NULL

-- Add Age field as integer type accepting NULL values
ALTER TABLE tCustomer_ETL ADD Age int NULL

-- Delete any spaces at the begining of the FullAddress Field
UPDATE tCustomer_ETL SET FullAddress = LTRIM(FullAddress)

-- Delete any spaces at the end of the FullAddress Field
UPDATE tCustomer_ETL SET FullAddress = RTRIM(FullAddress)

-- CHARINDEX will return the position of comma in string. SUBSTRING will return from character 1 to
-- the lenght of characters before the comma and update the contents of FirstName
UPDATE tCustomer_ETL SET FirstName = SUBSTRING(FullName,1,CHARINDEX(',',FullName)-1)

-- Populate the Surname in a similar fashion. Use 100 as the max possible field lenght
UPDATE tCustomer_ETL SET Surname = SUBSTRING(FullName,CHARINDEX(',',FullName)+1,100)

-- Split FullAddress likewise in Address1 field
UPDATE tCustomer_ETL SET Address1 = SUBSTRING(FullAddress,1,CHARINDEX(',',FullAddress)-1)

-- and PostCode field. Use +2 to account for the space after the comma
UPDATE tCustomer_ETL SET PostCode = SUBSTRING(FullAddress,CHARINDEX(',',FullAddress)+2,100)

-- Calculate the customer age as of today. GETDATE() RETURNS TODAY
UPDATE tCustomer_ETL SET Age = DATEDIFF(YEAR,DOB,GETDATE())

-- Add the missing zero at the start of the telephone number
UPDATE tCustomer_ETL SET Telephone = '0'+Telephone WHERE LEN(Telephone) <= 10

-- Covert Gender to Male/Female from M/F. The field lenght is 1 character and
-- needs to be extended to 6 characters
ALTER TABLE tCustomer_ETL ALTER COLUMN Gender nvarchar(6)
UPDATE tCustomer_ETL SET Gender = 'Male' WHERE Gender = 'M'
UPDATE tCustomer_ETL SET Gender = 'Female' WHERE Gender = 'F'

SELECT * FROM tCustomer_ETL
```

SECTION F: Creating stage 3.

Using the above examples of code and standard practices, create a stored procedure that will create the Stage 3 tables. We were able to identify issues only in the customer table. All the rest of the tables do not need any transformations in STAGE 2. So, the final stage should have these tables:

- tCustomer
- tProduct
- tPurchase
- tBankCardCharge
- tPaymentMethod
- tTillType