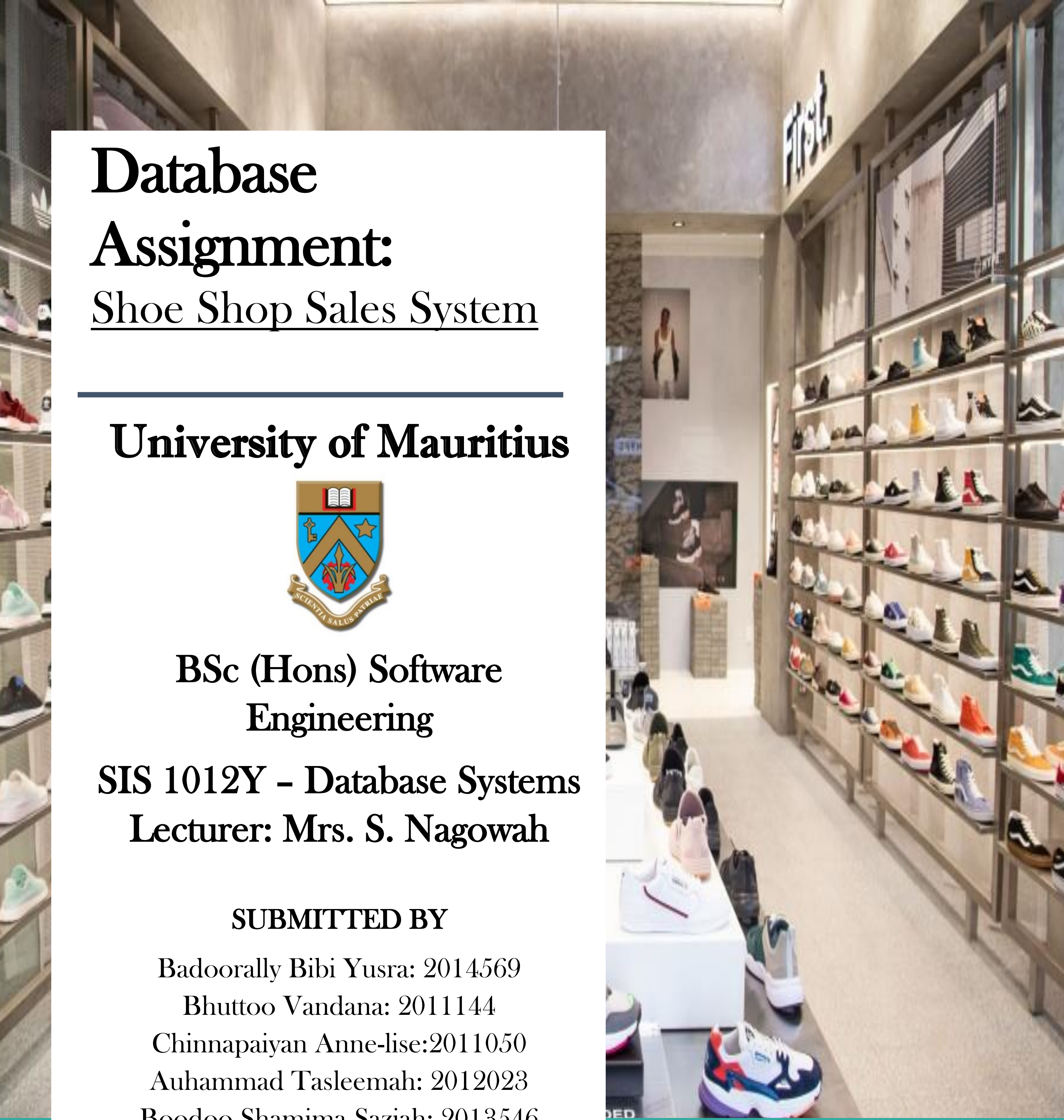


Database Assignment:

Shoe Shop Sales System

University of Mauritius

BSc (Hons) Software Engineering

SIS 1012Y – Database Systems

Lecturer: Mrs. S. Nagowah

SUBMITTED BY

Badoorally Bibi Yusra: 2014569
Bhuttoo Vandana: 2011144
Chinnapaiyan Anne-lise:2011050
Auhammad Tasleemah: 2012023
Boodoo Shamima Saziah: 2013546

31 MAY 2021

Table of Contents

1. INTRODUCTION

1.1 Introduction

In this assignment, the data for a sale in a shoe shop is being recorded in a database. When a customer arrives at the shop and purchases a shoe, the customer details and order details are entered in the database. The data about the product is retrieved from the shop sales system. Then the customer's invoice details are recorded. Moreover, details about the shoes in stock and details on employees are also stored in the database.

1.2 Description of the system

When the customer purchases a shoe, he or she will be given a unique order ID for his/her order and the customer's details will be recorded. The order table will contain information about the date the customer ordered the shoe and order description. The product table contains information about the different types of shoes the customer ordered. Then, the customer will receive an invoice when purchasing a shoe. The invoice contains the total amount of payment he/she has to make. Each product ID is related to shoe ID which contains details about the shoes available. The employee details will also be recorded and each employee will have access to the orders made by customers and will be able to manage them.

1.3 Aims and objectives

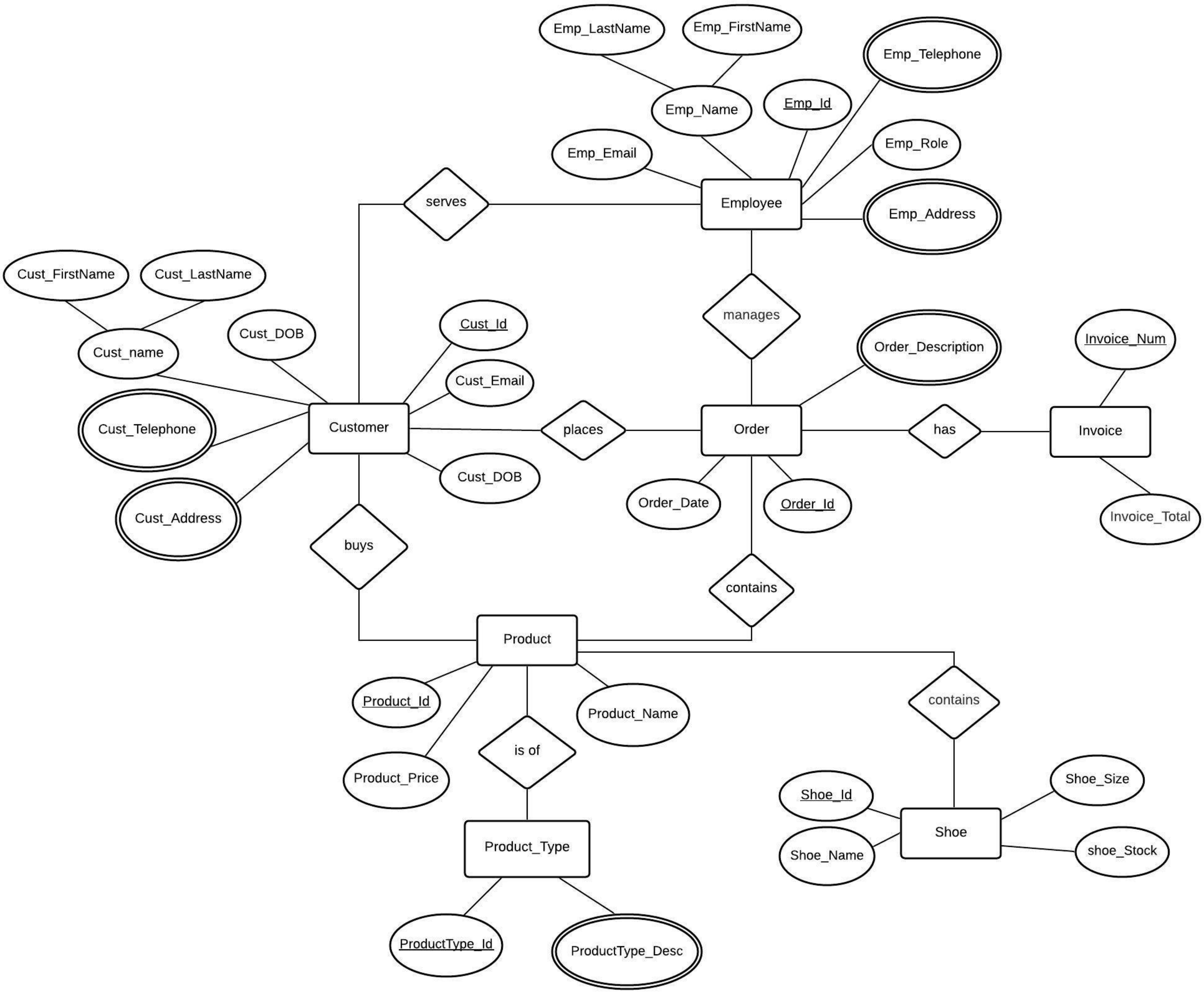
- To arrange all details of a business sales system properly in a database which would help access each information more easily.
- To eliminate redundant data.
- To be able to build a flexible database design.
- To be able to use stored procedures and triggers for inserting, updating and deleting tables in the database, add constraints and check for abnormal data.

1.4 Scope

- The system is targeted to facilitate management of the shoe shop by employees.
- The system should not allow abnormal data to be entered.
- The system should be easy in such a way that employees which are not familiar with SQL Management Studio can easily insert, update or delete values in the database through stored procedures.

2. ERD

ERD for Shoe Shop Sales System



3. Normalization

→ ONF

Order_Table (Order_ID, Order_Date, Order_Description, Invoice_Total, Invoice_Num, Cust_Id, Cust_First_Name, Cust_Last_Name, Cust_DOB, Cust_Email, Cust_Telephone, Cust_Address, Product_Id, Product_Name, Product_Price, ProductType_Id, ProductType_Desc, Shoe_Id, Shoe_Size, Shoe_Name, Shoe_Desc, shoe_Stock, Emp_Id, Emp_FirstName, Emp_LastName, Emp_Telephone, Emp_Address, Emp_Email, Emp_Role)

→ 1NF

Order_Table (<u>Order_ID</u>, Order_Date, Order_Description, Invoice_Num, Invoice_Total, Product_Id, Product_Name, Product_Price, ProductType_Id, ProductType_Desc, Shoe_Id, Shoe_Size, Shoe_Name, Shoe_Desc, shoe_Stock, (<u>Cust_Id</u>, Cust_First_Name, Cust_Last_Name, Cust_DOB, Cust_Email, Cust_Telephone, Cust_Address, Emp_Id, Emp_FirstName, Emp_LastName, Emp_Telephone, Emp_Address, Emp_Email, Emp_Role))

Functional Dependencies

Order_ID → Order_Date, Order_Description, Invoice_Num, Invoice_Total, Product_Id, Product_Name, Product_Price, ProductType_Id, ProductType_Desc, Shoe_Id, Shoe_Size, Shoe_Name, Shoe_Desc, shoe_Stock

Cust_Id → Cust_First_Name, Cust_Last_Name, Cust_DOB, Cust_Email, Cust_Telephone, Cust_Address

Order_ID, Cust_Id → Emp_Id, Emp_FirstName, Emp_LastName, Emp_Telephone, Emp_Address, Emp_Email, Emp_Role

→ 2NF

Order_Table (<u>Order_ID</u>, Order_Date, Order_Description, Invoice_Num, Invoice_Total, Product_Id, Product_Name, Product_Price, ProductType_Id, ProductType_Desc, Shoe_Id, Shoe_Size, Shoe_Name, Shoe_Desc, shoe_Stock)

Customer_Table (<u>Cust_Id</u>, Cust_First_Name, Cust_Last_Name, Cust_DOB, Cust_Email, Cust_Telephone, Cust_Address)

Customer_Order (Order_ID, Cust_Id, Emp_Id, Emp_FirstName, Emp_LastName, Emp_Telephone, Emp_Address, Emp_Email, Emp_Role)

Transitive Dependencies

Invoice_Num → Invoice_Total

Product_Id → Product_Name, Product_Price, ProductType_Id, Shoe_Id

ProductType_Id → ProductType_Desc

Shoe_Id → Shoe_Size, Shoe_Name, Shoe_Desc, shoe_Stock

Emp_Id → Emp_FirstName, Emp_LastName, Emp_Telephone, Emp_Address, Emp_Email,
Emp_Role

→ 3NF

Order_Table (Order_ID, Order_Date, Order_Description, Invoice_Num, Product_Id)

Customer_Table (Cust_Id, Cust_First_Name, Cust_Last_Name, Cust_DOB, Cust_Email, Cust_Telephone, Cust_Address)

Customer_Order (Order_ID, Cust_Id, Emp_Id)

Invoice (Invoice_Num, Invoice_Total)

Product (Product_Id, Product_Name, Product_Price, ProductType_Id, Shoe_Id)

Product_Type_Table (ProductType_Id, ProductType_Desc)

Shoe_Table (Shoe_Id, Shoe_Size, Shoe_Name, Shoe_Desc, shoe_Stock)

Employee_Table (Emp_Id, Emp_FirstName, Emp_LastName, Emp_Telephone, Emp_Address, Emp_Email, Emp_Role)

4. Implementation

4.1 Table description

1. Customer_Table

Field Name	Description	Field Type	Field Size
Cust_Id	Unique Id for customer	INTEGER	5
Cust_Last_Name	Customer last name	VARCHAR	30
Cust_First_Name	Customer first name	VARCHAR	30
Cust_DOB	Customer date of birth	DATE	10
Cust_Email	Customer email address	VARCHAR	20
Cust_Telephone	Customer telephone number	VARCHAR	12
Cust_Address	Customer address	VARCHAR	30

2. Invoice

Field Name	Description	Field Type	Field Size
Invoice_Num	Unique Number for invoice	INTEGER	5
Invoice_Total	Total for invoice	REAL	5

3. Employee_Table

Field Name	Description	Field Type	Field Size
Emp_Id	Unique Id for employee	INTEGER	3
Emp_FirstName	Employee first name	VARCHAR	20
Emp_LastName	Employee last name	VARCHAR	20
Emp_Telephone	Employee telephone number	VARCHAR	12
Emp_Address	Employee address	VARCHAR	40
Emp_Email	Employee email address	VARCHAR	30
Emp_Role	Employee role	VARCHAR	20

4. Product_Type_Table

Field Name	**Description**	**Field Type**	**Field Size**
ProductType_Id	Unique Id product type	INTEGER	4
ProductType_Desc	Description of product type	VARCHAR	40

5. Shoe_Table

Field Name	**Description**	**Field Type**	**Field Size**
Shoe_Id	Unique Id for each shoe	INTEGER	4
Shoe_Size	Size of shoe	INTEGER	2
Shoe_Desc	Description of shoe	VARCHAR	25
Shoe_Name	Name of shoe	VARCHAR	40
Shoe_Stock	Stock level of shoe	INTEGER	3

6. Product

Field Name	**Description**	**Field Type**	**Field Size**
Product_Id	Unique Id for product	INTEGER	3
Product_Name	Name of product	VARCHAR	40
Product_Price	Price of product	INTEGER	5
ProductType_Id	Unique Id given to product type	INTEGER	4
Shoe_Id	Unique Id given to shoe	INTEGER	4

7. Order_Table

Field Name	**Description**	**Field Type**	**Field Size**
Order_Id	Unique Id for order	INTEGER	3
Order_Date	Date of order	DATE	10
Order_Description	Description of order	VARCHAR	40
Invoice_Num	Unique Number given to an invoice	INTEGER	5
Product_Id	Unique Id given to product	INTEGER	3

8. Customer_Order

Field Name	Description	Field Type	Field Size
Order_Id	Unique Id for order	INTEGER	3
Cust_Id	Unique Id for customer	INTEGER	5
Emp_Id	Unique Id for employee	INTEGER	5

4.2 SQL Create Codes

This section shows how the tables were created according to the table descriptions.

- Customer Table

```
CREATE TABLE Customer_Table (
Cust_Id INTEGER,
Cust_Last_Name VARCHAR (30),
Cust_First_Name VARCHAR (30),
Cust_DOB DATE, /*YYYY-MM-DD*/
Cust_Email VARCHAR (20),
Cust_Telephone VARCHAR (12),
Cust_Address VARCHAR (30),
PRIMARY KEY (Cust_Id)
);
```

- Invoice Table

```
CREATE TABLE Invoice (
Invoice_Num INTEGER,
Invoice_Total REAL,
PRIMARY KEY (Invoice_Num)
);
```

- Employee Table

```
CREATE TABLE Employee_Table (
Emp_Id INTEGER,
Emp_FirstName VARCHAR (20),
Emp_LastName VARCHAR (20),
Emp_Telephone VARCHAR (12),
Emp_Address VARCHAR (40),
Emp_Email VARCHAR (30),
Emp_Role VARCHAR (20),
PRIMARY KEY(Emp_Id)
);
```

- Product_Type_Table

```
CREATE TABLE Product_Type_Table (
ProductType_Id INTEGER,
ProductType_Desc VARCHAR (40),
PRIMARY KEY(ProductType_Id)
);
```

- Shoe_Table

```
CREATE TABLE Shoe_Table (
Shoe_Id INTEGER,
Shoe_Size INTEGER,
Shoe_Desc VARCHAR (25),
Shoe_Name VARCHAR (40),
Shoe_Stock INTEGER,
PRIMARY KEY(Shoe_Id),
);
```

- Product Table

```
CREATE TABLE Product (
Product_Id INTEGER,
Product_Name VARCHAR (40),
ProductType_Id INTEGER,
Shoe_Id INTEGER,
Product_Price REAL,
PRIMARY KEY(Product_Id),
FOREIGN KEY(Shoe_Id) REFERENCES Shoe_Table (Shoe_Id),
FOREIGN KEY(ProductType_Id) REFERENCES Product_Type_Table
(ProductType_Id)
);
```

- Order Table

```
CREATE TABLE Order_Table (
Order_ID INTEGER,
Order_Date DATE,
Order_Description VARCHAR (80),
Invoice_Num INTEGER,
Product_Id INTEGER,
PRIMARY KEY (Order_ID),
FOREIGN KEY(Product_Id) REFERENCES Product
FOREIGN KEY(Invoice_Num) REFERENCES Invoice
);
```

- Customer_order Table

```
CREATE TABLE Customer_Order (
Order_Id INTEGER,
Cust_Id INTEGER,
Emp_Id INTEGER,
FOREIGN KEY(Order_Id) REFERENCES Order_Table (Order_Id),
FOREIGN KEY(Cust_Id) REFERENCES Customer_Table (Cust_Id),
FOREIGN KEY(Emp_Id) REFERENCES Employee_Table (Emp_Id),
);
```

4.3 SQL Insert Codes

❖ Customer Table

```
INSERT INTO Customer_Table
VALUES (20115,'Smith', 'John', '1980-05 03', 'JohnSmith@Yahoo.com',
        '59845624', 'Rose Road Flacq');
INSERT INTO Customer_Table
VALUES (20120,'Baker ','Leila','2006-04-14', 'LeilaBakr@Yahoo.com',
        '57965842', 'Coastal Road Trois Boutique');
INSERT INTO Customer_Table
VALUES (20117,'Jones','John', '1990 -06-02', 'JayJones@hotmail.com',
        '59874585', 'Hasen Sakir Road Plaine Verte');
INSERT INTO Customer_Table
VALUES (20118,'Parker','Mary','1998 -06-02’, ‘MaryParker@gmail.com',
        '52456987', 'La Caverne Road Vacoas');
INSERT INTO Customer_Table

VALUES (20119,'West','Kim', '1998-08-02', 'KimWest@gmail.com',

        '56458214', 'Balfour Road Beau Bassin');
```

❖ Invoice table

```
INSERT INTO Invoice VALUES (10201,2000);
INSERT INTO Invoice VALUES (10202,1200);
INSERT INTO Invoice VALUES (10203,1500);
INSERT INTO Invoice VALUES (10204,500);
INSERT INTO Invoice VALUES (10205,700);
```

❖ Employee_Table

```
INSERT INTO Employee_Table
VALUES (921,'Tasleemah', 'Auhammad', '57845134', 'Depinay St Valle
       Pitot', 'Tasleemahauhammad@gmail.com', 'Salesperson');
INSERT INTO Employee_Table
VALUES (922,'Yusra','Badoorally','58754696','Basgeeth Road Brisee
       Verdiere', 'Yusrabadoorally@gmail.com', 'Cashier');
INSERT INTO Employee_Table
VALUES (923,'Anne-Lise','Chinnapaiyan','58974452','st Julien Flacq',
       'AnneLisechin@gmail.com', 'Salesperson');
INSERT INTO Employee_Table
VALUES (924,'Vandana','Buttoo','52145879','Royal Road Ripailles',
       'Vandanabuttoo@gmail.com', 'Manager');
INSERT INTO Employee_Table
VALUES (925,'Saziah','Boodoo','54896321','Caroline Bel air',
       'Saziahboodoo@gmail.com', 'Salesperson');
```

❖ Product_Type_Table

```
INSERT INTO Product_Type_Table VALUES (9210,'Sandals');
INSERT INTO Product_Type_Table VALUES (9220,'Heels');
INSERT INTO Product_Type_Table VALUES (9230,'Boots');
INSERT INTO Product_Type_Table VALUES (9240,'Sneakers');
INSERT INTO Product_Type_Table VALUES (9250,'Flats');
```

❖ Shoe_Table

```
INSERT INTO Shoe_Table
VALUES (8112,36,'Green, leather','Nike',001);
INSERT INTO Shoe_Table
VALUES (8113,39,'Blue, textile','Addidas',003);
INSERT INTO Shoe_Table
VALUES (8114,30,'Grey, fabric','Fila',002);
INSERT INTO Shoe_Table
VALUES (8115,45,'Pink, Artificial Leather','Nike',10);
INSERT INTO Shoe_Table
VALUES (8116,42,'Gold, synthetics','Caterpillar',007);
INSERT INTO Shoe_Table
VALUES (8119,42,'Gold, synthetics','Caterpillar',007);
```

❖ Product table

```sql
INSERT INTO Product VALUES (123,'Woman Sandals',9210,8112,500);
INSERT INTO Product VALUES (124,'Woman Heels',9220,8113, 2000);
INSERT INTO Product VALUES (125,'Man Boots',9230,8114, 1024);
INSERT INTO Product VALUES (126,'Man Sneakers',9240,8115, 1200);
INSERT INTO Product VALUES (127,'Woman Flat',9250,8116, 600);
```

❖ Order table

```sql
INSERT INTO Order_Table
VALUES (101,'2020-08-04','vat=15%, Discount = 0%',10201,123);
INSERT INTO Order_Table
VALUES (102,'2021-01-04','Vat=15%, Discount = 10%',10202,124);
INSERT INTO Order_Table
VALUES (103,'2021-02-14','vat= 15%, Discount= 50%',10203,125);
INSERT INTO Order_Table
VALUES (104,'2020-07-14','Vat=15%, Discount= 20%',10204,126);
INSERT INTO Order_Table
VALUES (105,'2020-09-24','Vat= nil, Discount = 75%',10205,127);
```

❖ Customer_order table

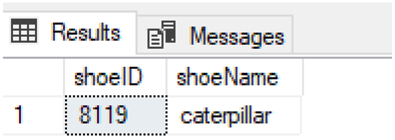
```sql
INSERT INTO Customer_Order VALUES (101, 20115,921);
INSERT INTO Customer_Order VALUES (102, 20115,922);
INSERT INTO Customer_Order VALUES (103, 20117,923);
INSERT INTO Customer_Order VALUES (104, 20117,923);
INSERT INTO Customer_Order VALUES (105,20119,925);
```

4.4 Stored procedures

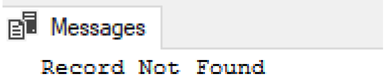
1) Stored procedure, sp_disp_shoe, will take a shoe's name as argument. The procedure will display the shoe ID as well as its name for those shoe(s) that is/are gold.

```
CREATE PROCEDURE sp_disp_shoe @shoeName VARCHAR (30)
AS
BEGIN
   DECLARE @shoeID INTEGER
   SELECT @shoeID = Shoe_Id
   FROM Shoe_Table
   WHERE Shoe_Name = @shoeName AND Shoe_Desc LIKE 'Gold%'
   IF @@ROWCOUNT = 0
         PRINT 'Record Not Found'
   ELSE
         SELECT 'shoeID' = @shoeID,'shoeName' = @shoeName
END
GO
```

Test Value = 'Caterpillar'



	shoeID	shoeName
1	8119	caterpillar



```
Record Not Found
```

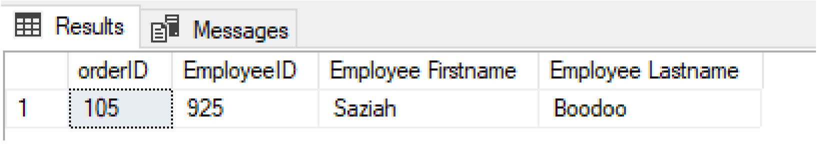
Test Value = 'Gucci'

2) Stored procedure, sp_disp_servedEmployee, will take an order ID as argument and display the first name, last name and ID of the employee who served that order.

```
CREATE PROCEDURE sp_disp_servedEmployee @orderID INTEGER
AS
BEGIN
   DECLARE @employeeID INTEGER, @fname VARCHAR(30), @Lname
   VARCHAR(30)
   SELECT @employeeID = Employee_Table.Emp_Id, @fname =
   Emp_FirstName, @Lname = Emp_LastName
   FROM Customer_Order, Employee_Table
   WHERE Order_Id = @orderID AND Customer_Order.Emp_Id =
   Employee_Table.Emp_Id
   IF @@ROWCOUNT = 0
         PRINT 'Record Not Found'
   ELSE
```

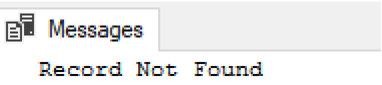
```
        SELECT 'orderID' = @orderid,'EmployeeID' = @employeeID,
        'Employee Firstname' =@fname,'Employee Lastname' = @Lname
END
```

Test Value = 105



	orderID	EmployeeID	Employee Firstname	Employee Lastname
1	105	925	Saziah	Boodoo

Test Value = 300



```
Record Not Found
```

3) Stored procedure, sp_insert_Employee, inserts the following record for a new employee:

ID: 927

First Name: Alisha

Last Name: Imrit

Phone Number: 54786325

Address: Latapie Road Bon Accueil

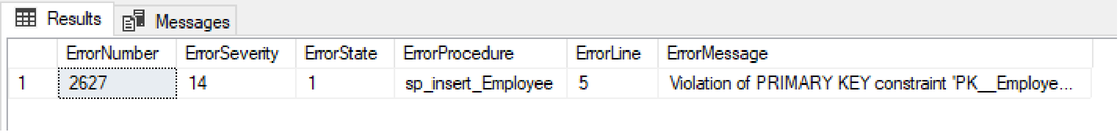
Email: AlishaImrit@gmail.com

Role: Cleaner

```
CREATE PROCEDURE sp_insert_Employee @id INTEGER, @Fname
VARCHAR(30), @Lname VARCHAR(30), @phone VARCHAR(12), @address
VARCHAR(40), @email VARCHAR(30), @role VARCHAR(20)
AS
BEGIN
   BEGIN TRY
        INSERT INTO Employee_Table(Emp_Id, Emp_FirstName,
        Emp_LastName, Emp_Telephone, Emp_Address, Emp_Email,
        Emp_Role)
        VALUES (@id, @Fname, @Lname, @phone, @address, @email,
        @role)
        PRINT 'Insert Successful'
   END TRY
   BEGIN CATCH
        SELECT
            ERROR_NUMBER() AS ErrorNumber
            ,ERROR_SEVERITY() AS ErrorSeverity
            ,ERROR_STATE() AS ErrorState
            ,ERROR_PROCEDURE() AS ErrorProcedure
```

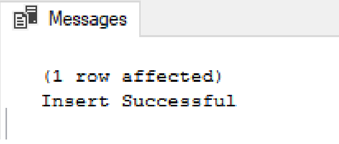
```
                ,ERROR_LINE() AS ErrorLine
                ,ERROR_MESSAGE() AS ErrorMessage;
    END CATCH
END
GO
```

Test Value = ‘921’- (Existing Primary Key)

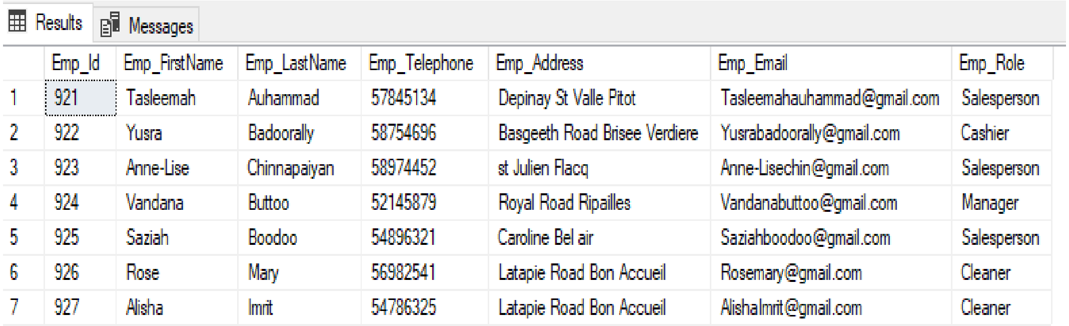


	ErrorNumber	ErrorSeverity	ErrorState	ErrorProcedure	ErrorLine	ErrorMessage
1	2627	14	1	sp_insert_Employee	5	Violation of PRIMARY KEY constraint 'PK__Employe...

All data entered correctly



```
(1 row affected)
Insert Successful
```

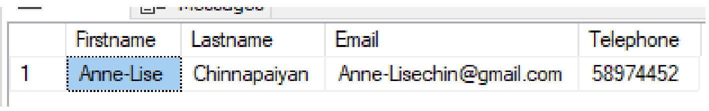


	Emp_Id	Emp_FirstName	Emp_LastName	Emp_Telephone	Emp_Address	Emp_Email	Emp_Role
1	921	Tasleemah	Auhammad	57845134	Depinay St Valle Pitot	Tasleemahauhammad@gmail.com	Salesperson
2	922	Yusra	Badoorally	58754696	Basgeeth Road Brisee Verdiere	Yusrabadoorally@gmail.com	Cashier
3	923	Anne-Lise	Chinnapaiyan	58974452	st Julien Flacq	Anne-Lisechin@gmail.com	Salesperson
4	924	Vandana	Buttoo	52145879	Royal Road Ripailles	Vandanabuttoo@gmail.com	Manager
5	925	Saziah	Boodoo	54896321	Caroline Bel air	Saziahboodoo@gmail.com	Salesperson
6	926	Rose	Mary	56982541	Latapie Road Bon Accueil	Rosemary@gmail.com	Cleaner
7	927	Alisha	Imrit	54786325	Latapie Road Bon Accueil	AlishaImrit@gmail.com	Cleaner

4) Stored procedure, sp_disp_EmployeeDetails, takes the role of an employee as argument and display his/her first name, last name, email and phone number of the employee who lives at Flacq.

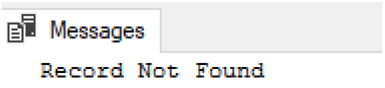
```
CREATE PROCEDURE sp_disp_EmployeeDetails @EmpRole VARCHAR (20)
AS
BEGIN
    DECLARE @EmpFirstName VARCHAR(20), @EmpLastName VARCHAR(20),
    @EmpEmail VARCHAR(30), @EmpphoneNum VARCHAR (12)
    SELECT @EmpFirstName = Emp_FirstName, @EmpLastName =
    Emp_LastName, @EmpEmail = Emp_Email, @EmpphoneNum =
    Emp_Telephone
    FROM Employee_Table
    WHERE Emp_Role = @EmpRole AND Emp_Address LIKE '%Flacq'
    IF @@ROWCOUNT = 0
            PRINT 'Record Not Found'
    ELSE
        SELECT 'Firstname' = @EmpFirstName, 'Lastname' =
@EmpLastName, 'Email' = @EmpEmail, 'Telephone' = @EmpphoneNum
END
```

Test Value = ‘Salesperson’



	Firstname	Lastname	Email	Telephone
1	Anne-Lise	Chinnapaiyan	Anne-Lisechin@gmail.com	58974452

Test Value = ‘Cashier’

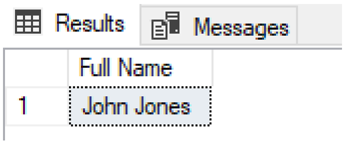


```
Record Not Found
```

5) Stored procedure, sp_disp_customerName, takes an order ID as argument and displays the first name and last name of the customer concatenated together under the column named as full name.

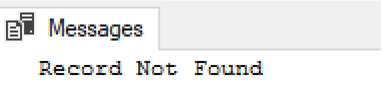
```
CREATE PROCEDURE sp_disp_CustomerName @orderID INTEGER
AS
BEGIN
    DECLARE @Fname VARCHAR (30), @Lname VARCHAR (30)
    SELECT @Fname = Cust_First_Name, @Lname = Cust_Last_Name
    FROM Customer_Order, Customer_Table
    WHERE Order_Id = @orderID AND Customer_Table.Cust_Id =
    Customer_Order.Cust_Id
    IF @@ROWCOUNT = 0
        PRINT 'Record Not Found'
    ELSE
        SELECT 'Full Name' = @Fname + ' ' + @Lname
END
```

Test Value = 103



	Full Name
1	John Jones

Test Value = 110

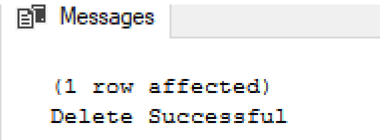


```
Record Not Found
```

6) Stored procedure, sp_del_shoe, takes a shoe ID as argument and delete the record from the shoe table.

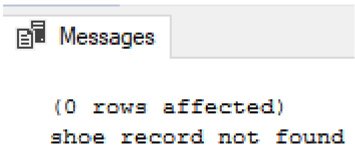
```
CREATE PROCEDURE sp_del_shoe @shoeId INTEGER
AS
BEGIN
   BEGIN TRY
         DELETE FROM Shoe_Table WHERE Shoe_Id = @shoeId
         IF @@ROWCOUNT = 0
               PRINT 'Shoe record not found'
         ELSE
               PRINT 'Delete Successful'
   END TRY
   BEGIN CATCH
         SELECT 'Error: ' + ERROR_MESSAGE() AS ErrorMessage;
   END CATCH
END
```

Test Value = 8119



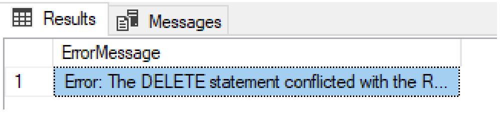
```
(1 row affected)
Delete Successful
```

Test Value = 8008



```
(0 rows affected)
shoe record not found
```

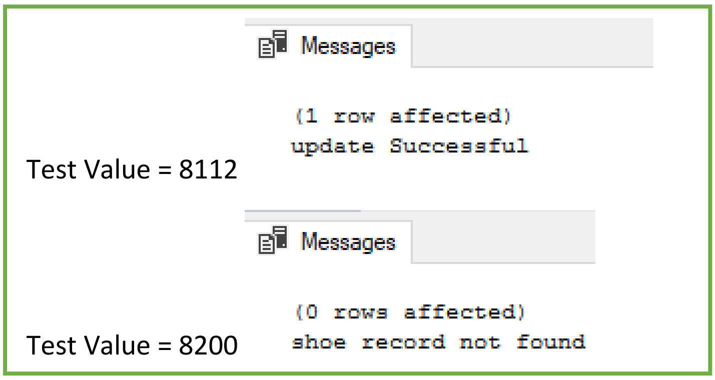
Test value = 8116



	ErrorMessage
1	Error: The DELETE statement conflicted with the R...

7) Stored procedure, sp_upd_stock, takes the shoe ID and an integer value y as arguments. The procedure should change the stock level of that specific shoe to this integer value for shoes with stock level less than 2.

```
CREATE PROCEDURE sp_upt_stock @shoeID INTEGER, @y INTEGER
AS
BEGIN
   BEGIN TRY
   UPDATE Shoe_Table
   SET Shoe_Stock = @y
   WHERE Shoe_Id =@shoeID AND Shoe_Stock <=2
   IF @@ROWCOUNT = 0
              PRINT 'shoe record not found'
        ELSE
              PRINT 'update Successful'
   END TRY
   BEGIN CATCH
        SELECT 'Error: ' + ERROR_MESSAGE() AS ErrorMessage;
   END CATCH
END
```



Test Value = 8112

Messages

```
(1 row affected)
update Successful
```

Test Value = 8200

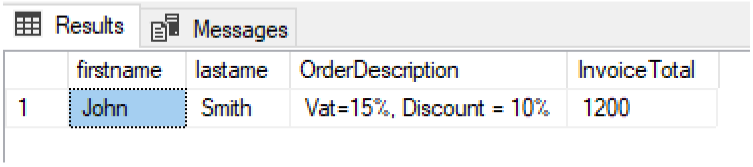
Messages

```
(0 rows affected)
shoe record not found
```

8) Stored procedure, sp_disp_oderDescInvoiceTotal, takes the first and last names of a customer and display his order description and invoice total.

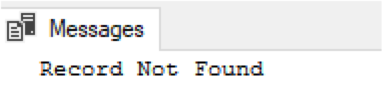
```
CREATE PROCEDURE sp_disp_oderDescInvoiceTotal @Fname VARCHAR(30),
@Lname VARCHAR(30)
AS
BEGIN
   DECLARE @orderDesc VARCHAR(80), @invoiceTotal REAL
   SELECT @orderDesc = Order_Description, @invoiceTotal =
   Invoice_Total
   FROM  Customer_Table, Order_Table, Invoice, Customer_Order
   WHERE Cust_First_Name = @Fname AND Cust_Last_Name = @Lname AND
   Customer_Table.Cust_Id = Customer_Order.Cust_Id AND
   Customer_Order.Order_Id = Order_Table.Order_ID AND
   Order_Table.Invoice_Num = Invoice.Invoice_Num
   IF @@ROWCOUNT = 0
         PRINT 'Record Not Found'
   ELSE
         SELECT 'firstname' = @Fname,'lastame' = @Lname,
         'OrderDescription' = @orderDesc, 'InvoiceTotal' =
         @invoiceTotal
END
GO
```

Test Value = @Fname: ‘John’, @Lname: ‘Smith’



	firstname	lastame	OrderDescription	InvoiceTotal
1	John	Smith	Vat=15%, Discount = 10%	1200

Test Value = @Fname: ‘Lisa’, @Lname: ‘Ally’



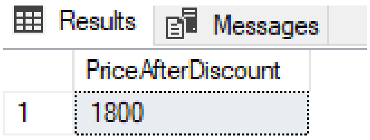
```
Record Not Found
```

9) Stored procedure, sp_discount, calculates the final price of a product after applying the discount. The function will take as arguments the type of the item and the price before application of discount. Items of type ‘Sneakers’ and ‘Boots’ are exempt of discount. A discount of 25% is applied to items of type ‘sandals’ and ‘flats’ and a discount of 10% is applied to all other item types.

```
CREATE PROCEDURE sp_discount @typeofItem varchar(30)
AS
DECLARE @priceBeforediscount float
SELECT @priceBeforediscount = Product_Price
FROM Product, Product_Type_Table
WHERE ProductType_Desc = @typeofItem AND
Product_Type_Table.ProductType_Id = Product.ProductType_Id
DECLARE @finalPrice float, @discount float
SET @finalPrice = 0
SET @discount = 0
IF (@typeofItem = 'Sneakers') OR (@typeofItem = 'Boots')
   BEGIN
         SET @discount = 0
   END
ELSE IF (@typeofItem = 'sandals') OR (@typeofItem = 'flats')
   BEGIN
         SET @discount = 0.25 * @priceBeforediscount
   END
ELSE
   BEGIN
         SET @discount = 0.10 * @priceBeforediscount
   END
SET @finalPrice = @priceBeforediscount - @discount
PRINT @finalPrice
```

```
GO
```

Test Value = ‘Heel’



```
   PriceAfterDiscount
1  1800
```

10) Stored procedure, `sp_ins_order_details,` inserts the following details:

Customer Name: Smith John

Order ID: 110

Order Date: 30 May 2020

Order Description: Vat = 0%, Discount = 0%

Product ID: 10210

InvoiceNum: 130

Invoice Total: Rs 2000

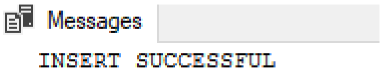
A message will be displayed on successful insert.

Note: this procedure inserts data into 2 tables. If the shoe Id already exists in the shoe table, the same shoe Id is used, otherwise an insert is also done in the shoe table.

```
CREATE PROCEDURE sp_ins_order_details @lastname varchar(40),
@firstname varchar(40), @orderID INTEGER, @orderdate date,
@orderDesc varchar(80), @invoiceNum INTEGER, @productID
INTEGER, @invoiceTotal REAL
AS
BEGIN
        DECLARE @custID INTEGER
        SELECT @custID = Cust_Id
        FROM Customer_Table
        where Cust_Last_Name = @lastname AND Cust_First_Name =
        @firstname

        IF (@custID = ' ')
        BEGIN
        DECLARE @maximumid integer
        SELECT @maximumid = Max(a_id)
        FROM Customer_Table
        SET @custID = @maximumid +1
        END
```

```
INSERT INTO Order_Table(Order_ID, OrdEr_Date
,Order_Description, Invoice_Num, Product_ID)
VALUES (@orderID, @orderdate, @orderDesc, @invoiceNum,
@productID)
PRINT 'Insert Successful'

INSERT INTO Invoice(Invoice_Num, Invoice_Total)
VALUES (@invoiceNum, @invoiceTotal)
PRINT 'INSERT SUCCESSFUL'
END
GO
```

All data above has been inserted correctly



```
INSERT SUCCESSFUL
```

4.5 Triggers

This section shows the sample codes for triggers.

1. Trigger checks if

   - the customer’s last name or first name is left blank
   - checks if customer already exists in table (checks first name & last name of customer)
   - checks if email is not valid or valid (email must contain @ and .)
   - checks if phone number is valid (phone number must not contain alphabets)

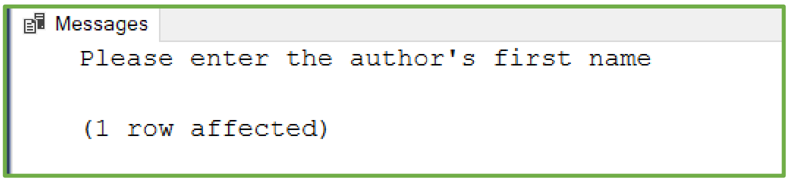
in table Customer_Table.

```
CREATE TRIGGER tg_chk_customer
ON Customer_Table
INSTEAD OF INSERT
AS
DECLARE @firstname VARCHAR (30)
DECLARE @lastname VARCHAR (30)
DECLARE @email VARCHAR (20)
DECLARE @phone_number VARCHAR(12)
DECLARE @email_chk_1 VARCHAR(1) = '@'
DECLARE @email_chk_2 VARCHAR(1) = '.'

SET @firstname = (SELECT Cust_First_Name FROM INSERTED)
SET @lastname = (SELECT Cust_Last_Name FROM INSERTED)
SET @email = (SELECT Cust_Email FROM INSERTED)
SET @phone_number = (SELECT Cust_Telephone FROM INSERTED)

IF (@firstname is NULL)
     BEGIN
          PRINT 'Please enter the author''s first name'
          RETURN
     END
IF (@lastname is NULL)
     BEGIN
          PRINT 'Please enter the author''s last name'
          RETURN
     END

IF EXISTS (
     SELECT Cust_First_Name,Cust_Last_Name FROM Customer_Table
WITH(NOLOCK)
     WHERE Cust_First_Name = @firstname AND Cust_Last_Name =
@lastname)
     BEGIN
          PRINT 'The author already exists in the table.'
     END
ELSE
     BEGIN
          PRINT 'The author doesn''t exist in the table.'
     END

IF (@email LIKE '%' + @email_chk_1 + '%' + @email_chk_2 + '%')
     BEGIN
          PRINT 'The email is valid.'
     END
ELSE
     BEGIN
          PRINT 'Please enter a valid email.'
          RETURN
     END

IF (@phone_number LIKE '%[a-zA-Z]%' )
     BEGIN
          PRINT 'The phone number is invalid. Please enter a
correct phone number.'
     END
```

```
    ELSE
          BEGIN
                INSERT INTO Customer_Table
                (Cust_First_Name,Cust_Last_Name,Cust_Email,Cust_Tele
                phone )
                SELECT @firstname, @lastname, @email, @phone_number
    FROM INSERTED
          END
```

→ First test value:

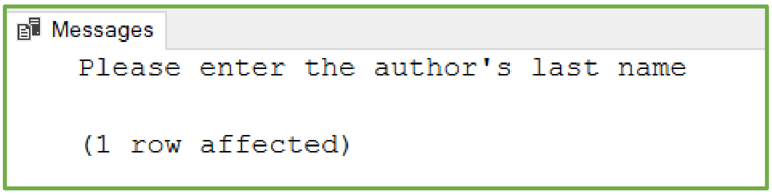
```
INSERT INTO Customer_Table (Cust_First_Name, Cust_Last_Name,
                              Cust_Email, Cust_Telephone)
VALUES (null, 'Bhuttoo', 'vb@gmail.com', '54905550')
```



```
Please enter the author's first name

(1 row affected)
```

→ Second test value:

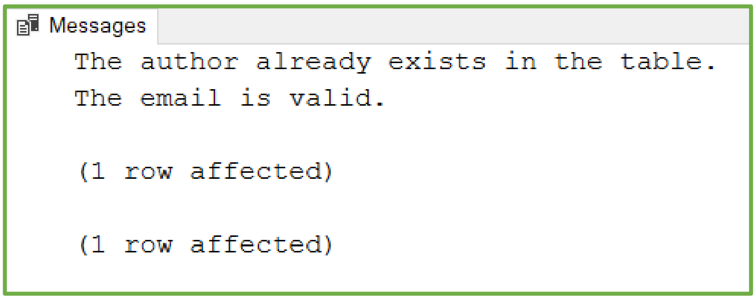
```
INSERT INTO Customer_Table (Cust_First_Name, Cust_Last_Name,
                              Cust_Email, Cust_Telephone)
VALUES ('Varsha', null, 'vb@gmail.com', '54905550')
```



```
Please enter the author's last name

(1 row affected)
```

→ Third test value:

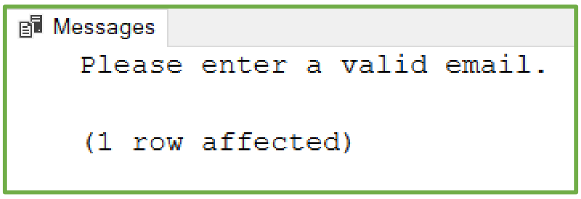
```
INSERT INTO Customer_Table (Cust_First_Name, Cust_Last_Name,
                              Cust_Email, Cust_Telephone)
VALUES ('John', 'Smith', JohnSmith@Yahoo.com', ' 59845624')
```



```
The author already exists in the table.
The email is valid.

(1 row affected)

(1 row affected)
```

→ Fourth test value:

```
INSERT INTO Customer_Table (Cust_First_Name, Cust_Last_Name,
                            Cust_Email, Cust_Telephone)

VALUES ('Varsha', 'Bhuttoo', 'vbprodigycom', '54905550')
```

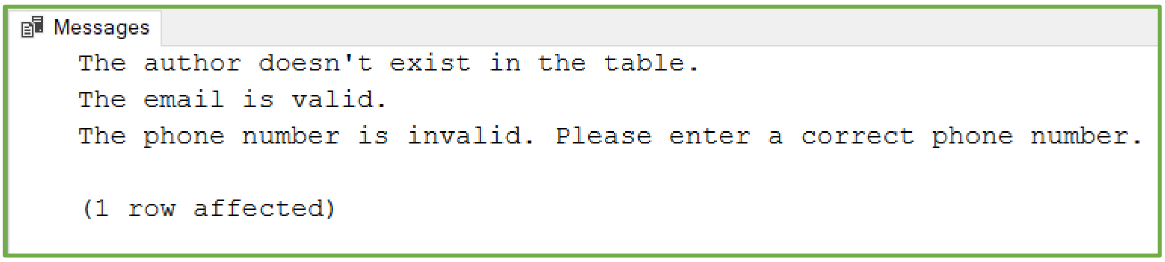


Messages

```
Please enter a valid email.

(1 row affected)
```

→ Fifth test value:

```
INSERT INTO Customer_Table (Cust_First_Name, Cust_Last_Name,
                            Cust_Email, Cust_Telephone)

VALUES ('Varsha', 'Bhuttoo', 'vb@prodigy.com', '5asds6567')
```

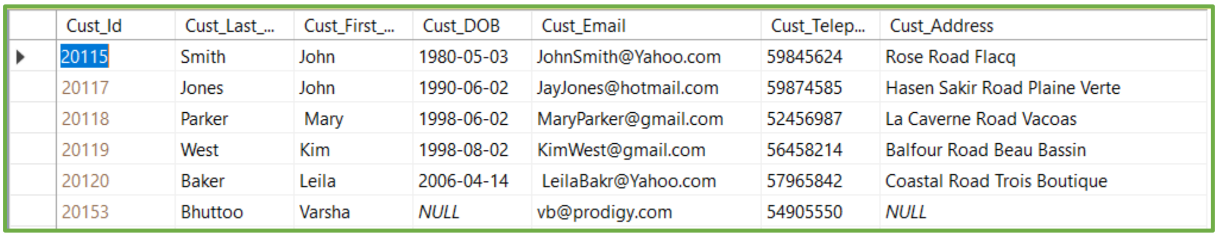


Messages

```
The author doesn't exist in the table.
The email is valid.
The phone number is invalid. Please enter a correct phone number.

(1 row affected)
```

→ Sixth test value:

```
INSERT INTO Customer_Table (Cust_First_Name, Cust_Last_Name,
                            Cust_Email, Cust_Telephone)

VALUES ('Varsha', 'Bhuttoo', 'vb@prodigy.com', '54905550')
```



	Cust_Id	Cust_Last_...	Cust_First_...	Cust_DOB	Cust_Email	Cust_Telep...	Cust_Address
▶	20115	Smith	John	1980-05-03	JohnSmith@Yahoo.com	59845624	Rose Road Flacq
	20117	Jones	John	1990-06-02	JayJones@hotmail.com	59874585	Hasen Sakir Road Plaine Verte
	20118	Parker	Mary	1998-06-02	MaryParker@gmail.com	52456987	La Caverne Road Vacoas
	20119	West	Kim	1998-08-02	KimWest@gmail.com	56458214	Balfour Road Beau Bassin
	20120	Baker	Leila	2006-04-14	LeilaBakr@Yahoo.com	57965842	Coastal Road Trois Boutique
	20153	Bhuttoo	Varsha	*NULL*	vb@prodigy.com	54905550	*NULL*

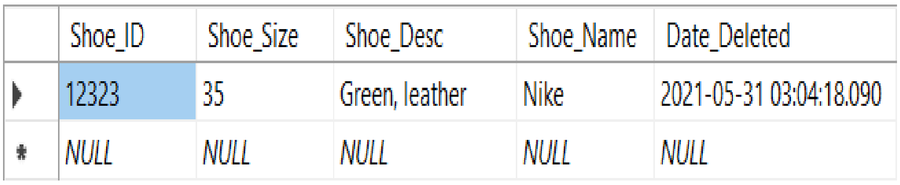
2. Trigger inserts deleted records of the Shoe_Table into another table deleted_shoe_records upon deletion.

```
CREATE TABLE deleted_shoe_records
(
Shoe_ID INTEGER,
Shoe_Size INTEGER,
Shoe_Desc VARCHAR(52),
Shoe_Name VARCHAR (40),
Date_Deleted DATETIME
);

CREATE TRIGGER tg_deletedShoeRecords
ON Shoe_Table
INSTEAD OF DELETE
AS
DECLARE @shoeID INTEGER
DECLARE @shoeSize INTEGER
DECLARE @shoeDesc VARCHAR (52)
DECLARE @shoeName VARCHAR (40)
DECLARE @DateDeleted DATETIME

SET @shoeID = (SELECT Shoe_Id FROM DELETED)
SET @shoeSize = (SELECT Shoe_Size FROM DELETED)
SET @shoeDesc = (SELECT Shoe_Desc FROM DELETED)
SET @shoeName = (SELECT Shoe_Name FROM DELETED)

BEGIN
   INSERT INTO deleted_shoe_records
   VALUES (@shoeID,@shoeSize,@shoeDesc,@shoeName,GETDATE());
   DELETE FROM Shoe_Table WHERE (Shoe_ID = @shoeID AND Shoe_Name
= @shoeName)
END
```

→ First test value:

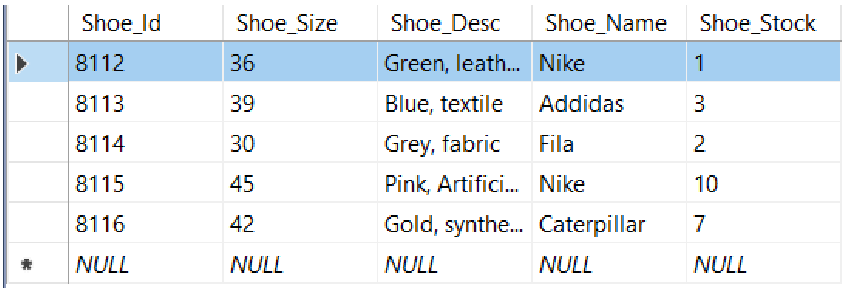
```
INSERT INTO Shoe_Table (Shoe_Id, Shoe_Size, Shoe_Desc, Shoe_Name,
                           Shoe_Stock)
VALUES (12323,35,'Green, leather', 'Nike', 3);
DELETE FROM Shoe_Table WHERE (Shoe_ID = '12323' AND Shoe_Name =
                                 'Nike')
```

Insertion in table deleted_shoe_records

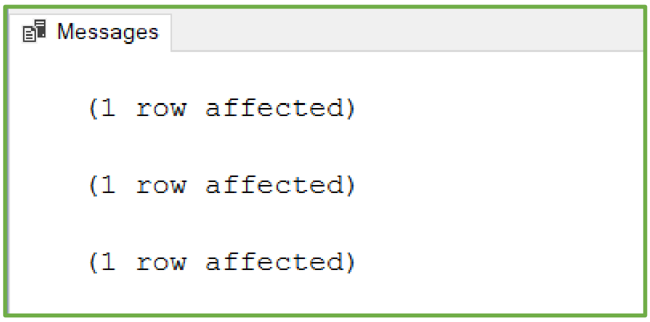


	Shoe_ID	Shoe_Size	Shoe_Desc	Shoe_Name	Date_Deleted
▶	12323	35	Green, leather	Nike	2021-05-31 03:04:18.090
*	NULL	NULL	NULL	NULL	NULL

Deletion in table Shoe_Table



	Shoe_Id	Shoe_Size	Shoe_Desc	Shoe_Name	Shoe_Stock
▶	8112	36	Green, leath...	Nike	1
	8113	39	Blue, textile	Addidas	3
	8114	30	Grey, fabric	Fila	2
	8115	45	Pink, Artifici...	Nike	10
	8116	42	Gold, synthe...	Caterpillar	7
*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*



```
(1 row affected)

(1 row affected)

(1 row affected)
```

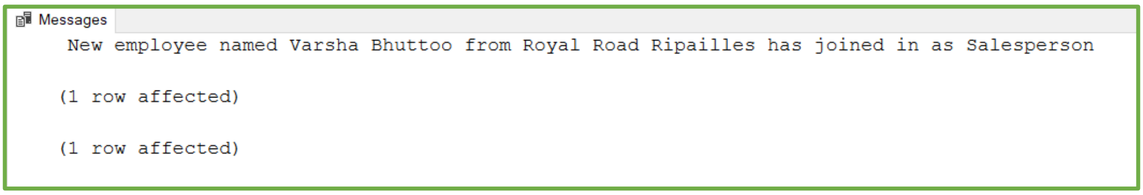
3. Trigger displays "New employee 'Emp_FirstName' 'Emp_LastName' from 'Emp_Address' has joined in as 'Emp_Role' when a new record of employee is inserted in Employee_Table.

```
CREATE TRIGGER tg_disp_newEmployee
ON Employee_Table
INSTEAD OF INSERT
AS
DECLARE @firstname VARCHAR (20)
DECLARE @lastname VARCHAR (20)
DECLARE @Telephone VARCHAR (12)
DECLARE @Address VARCHAR (40)
DECLARE @email VARCHAR (30)
DECLARE @empRole VARCHAR (20)

SET @firstname = (SELECT Emp_FirstName FROM INSERTED)
SET @lastname = (SELECT Emp_LastName FROM INSERTED)
SET @Telephone = (SELECT Emp_Telephone FROM INSERTED)
SET @Address = (SELECT Emp_Address FROM INSERTED)
SET @email = (SELECT Emp_Email FROM INSERTED)
SET @empRole = (SELECT Emp_Role FROM INSERTED)

BEGIN
```

```
        PRINT ' New employee named ' + CONCAT(@firstname, ' ' ,
        @lastname) + ' from ' + @Address + ' has joined in as ' +
        @empRole;
        INSERT INTO Employee_Table (Emp_FirstName, Emp_LastName,
        Emp_Telephone, Emp_Address, Emp_Email, Emp_Role)
        VALUES (@firstname, @lastname, @Telephone, @Address, @email,
        @empRole)
END
```

→ First test value:

```
INSERT INTO Employee_Table (Emp_FirstName, Emp_LastName,
      Emp_Telephone, Emp_Address, Emp_Email, Emp_Role)
VALUES ('Varsha','Bhuttoo','54905550','Royal Road Ripailles',
        'vb@gmail.com','Salesperson')
```



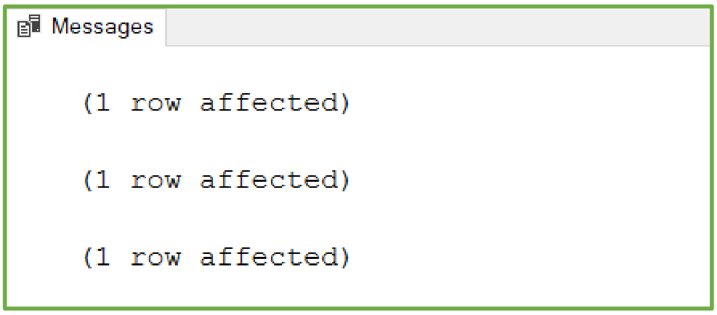
```
 New employee named Varsha Bhuttoo from Royal Road Ripailles has joined in as Salesperson

(1 row affected)

(1 row affected)
```

4. Trigger takes into consideration any change in the price in Product Table and this change is recorded in another table shoePrice_audit along with the user who made the change and the date the price was changed.

```
CREATE TABLE shoePrice_audit
(
ProductId INTEGER,
ProductName VARCHAR (40),
oldPrice REAL,
newPrice REAL,
user_changed SYSNAME,
date_changed DATETIME
);

GO
CREATE TRIGGER tg_shoePrice_Audit
ON Product
INSTEAD OF UPDATE
AS
DECLARE @productId INTEGER
DECLARE @productName VARCHAR (40)
DECLARE @newprice REAL
DECLARE @oldprice REAL

SET @productId = (SELECT Product_Id FROM INSERTED)
SET @productName = (SELECT Product_Name FROM INSERTED)
SET @oldprice = (SELECT Product_Price FROM DELETED)
SET @newprice = (SELECT Product_Price FROM INSERTED)

BEGIN
   SELECT @productId = Product_Id, @productName = Product_Name
   FROM Product
   WHERE @oldprice = Product_Price
   INSERT INTO shoePrice_audit
   VALUES
(@productId,@productName,@oldprice,@newprice,CURRENT_USER,
GETDATE());
   UPDATE Product
   SET Product_Price = @newprice
   WHERE Product_Id = @productId
END
```

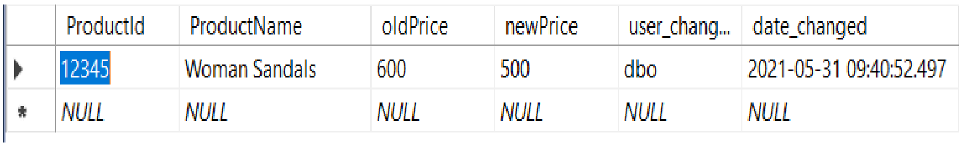
→ First test value:

```
INSERT INTO Product (Product_Id,Product_Name,Product_Price)
VALUES (12345,'Woman Sandals',600);

UPDATE Product
SET Product_Price = 500
WHERE Product_Id = 12345
```



```
(1 row affected)

(1 row affected)

(1 row affected)
```

” NUMBER”: Insertion in table shoePrice_audit



	ProductId	ProductName	oldPrice	newPrice	user_chang...	date_changed
▶	12345	Woman Sandals	600	500	dbo	2021-05-31 09:40:52.497
*	NULL	NULL	NULL	NULL	NULL	NULL

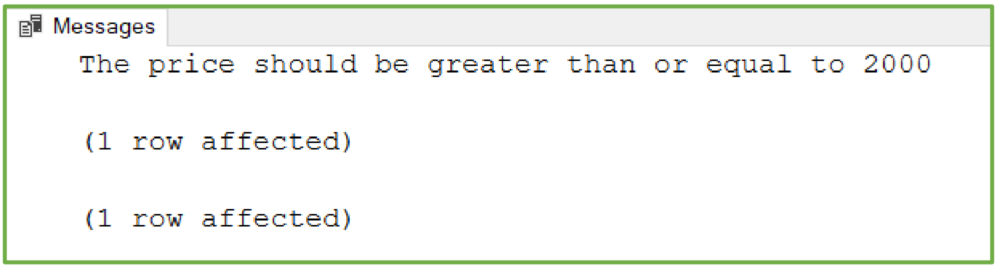
5. Trigger checks the price of a product. If the product name is ‘Woman Heels’, and its price is less than 2000, a message will be printed to indicate that the price should be more than 2000 and an increase by 10% in the difference between 2000 and price of the product should be made upon insertion. If the product name is ‘Man Sneakers’, and its price is less than 1000, a message will be printed to indicate that the price should be more than 1000 and an increase by 20% in the difference between 2000 and price of the product should be made upon insertion.

```
CREATE TRIGGER trg_check_product_price
ON Product
INSTEAD OF INSERT
AS
DECLARE @productID INTEGER
DECLARE @productName VARCHAR (40)
DECLARE @productPrice REAL
DECLARE @newprice REAL

SET @productID = (SELECT Product_Id FROM INSERTED)
SET @productName = (SELECT Product_Name FROM INSERTED)
SET @productPrice = (SELECT Product_Price FROM INSERTED)

BEGIN
   IF (@productName= 'Woman Heels')
        BEGIN
              IF (@productPrice < 2000)
                   BEGIN
                           SET @newprice = (2000 -
                           @productPrice)*1.1 + @productPrice
                           PRINT 'The price should be greater
                           than or equal to 2000';
                           INSERT INTO Product (Product_Id,
                           Product_Name, Product_Price)
                           VALUES (@productID, @productName,
                           @newprice)
                   END
              ELSE
                   BEGIN
                        PRINT 'The price entered is correct';
                           INSERT INTO Product(Product_Id,
                           Product_Name, Product_Price)
```

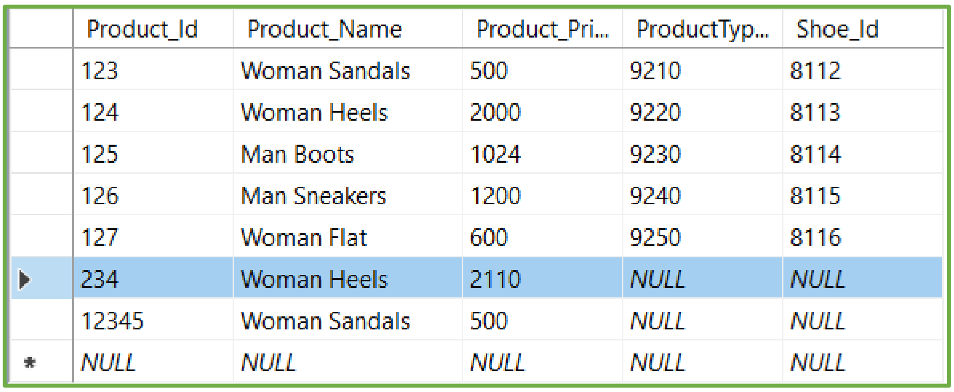
```
                              VALUES (@productID, @productName,
                              @productPrice)
                      END
           END
     ELSE IF (@productName= 'Man Sneakers')
                BEGIN
                IF (@productPrice < 1000)
                      BEGIN
                              SET @newprice = (1000 -
                              @productPrice)*1.2 + @productPrice
                              PRINT 'The price should be greater
                              than or equal to 1000';
                              INSERT INTO Product
                              (Product_Id,Product_Name,Product_Price
                              )
                              VALUES (@productID, @productName,
                              @newprice)
                      END
                ELSE
                      BEGIN
                           PRINT 'The price entered is correct';
                              INSERT INTO Product(Product_Id,
                              Product_Name, Product_Price)
                              VALUES (@productID, @productName,
                              @productPrice)
                      END
           END
   END
```

→ First test value:

```
INSERT INTO Product (Product_Id,Product_Name,Product_Price)

VALUES (234,'Woman Heels',900)
```



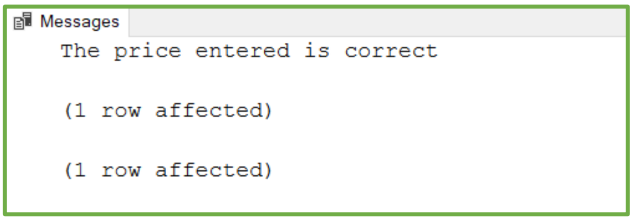
```
The price should be greater than or equal to 2000

(1 row affected)

(1 row affected)
```



	Product_Id	Product_Name	Product_Pri...	ProductTyp...	Shoe_Id
	123	Woman Sandals	500	9210	8112
	124	Woman Heels	2000	9220	8113
	125	Man Boots	1024	9230	8114
	126	Man Sneakers	1200	9240	8115
	127	Woman Flat	600	9250	8116
▶	234	Woman Heels	2110	NULL	NULL
	12345	Woman Sandals	500	NULL	NULL
*	NULL	NULL	NULL	NULL	NULL

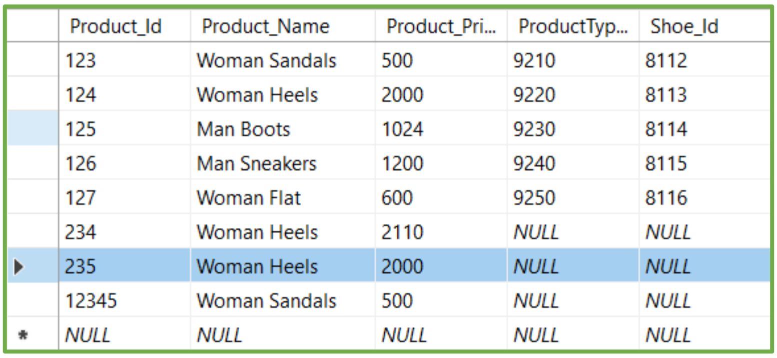
→ Second test value:

```
INSERT INTO Product (Product_Id,Product_Name,Product_Price)
VALUES (235,'Woman Heels',2000)
```



Messages

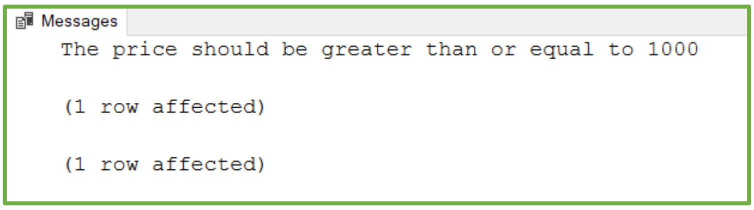
```
The price entered is correct

(1 row affected)

(1 row affected)
```



	Product_Id	Product_Name	Product_Pri...	ProductTyp...	Shoe_Id
	123	Woman Sandals	500	9210	8112
	124	Woman Heels	2000	9220	8113
	125	Man Boots	1024	9230	8114
	126	Man Sneakers	1200	9240	8115
	127	Woman Flat	600	9250	8116
	234	Woman Heels	2110	NULL	NULL
▶	235	Woman Heels	2000	NULL	NULL
	12345	Woman Sandals	500	NULL	NULL
*	NULL	NULL	NULL	NULL	NULL

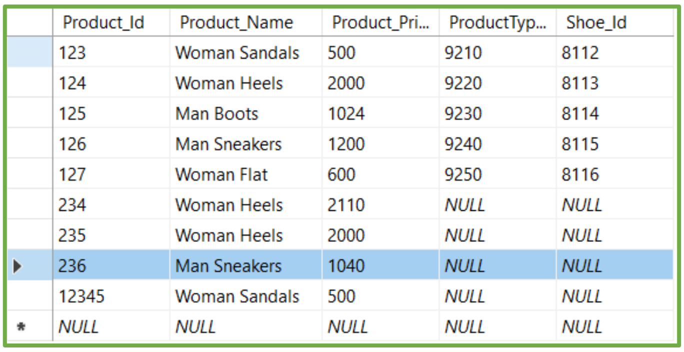
→ Third test value:

```
INSERT INTO Product (Product_Id,Product_Name,Product_Price)
VALUES (236,'Man Sneakers',800)
```



Messages

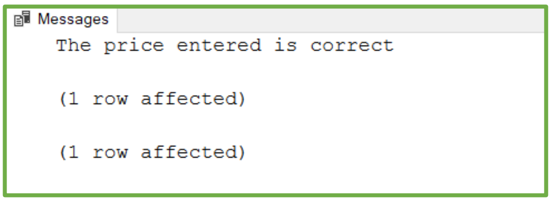
```
The price should be greater than or equal to 1000

(1 row affected)

(1 row affected)
```



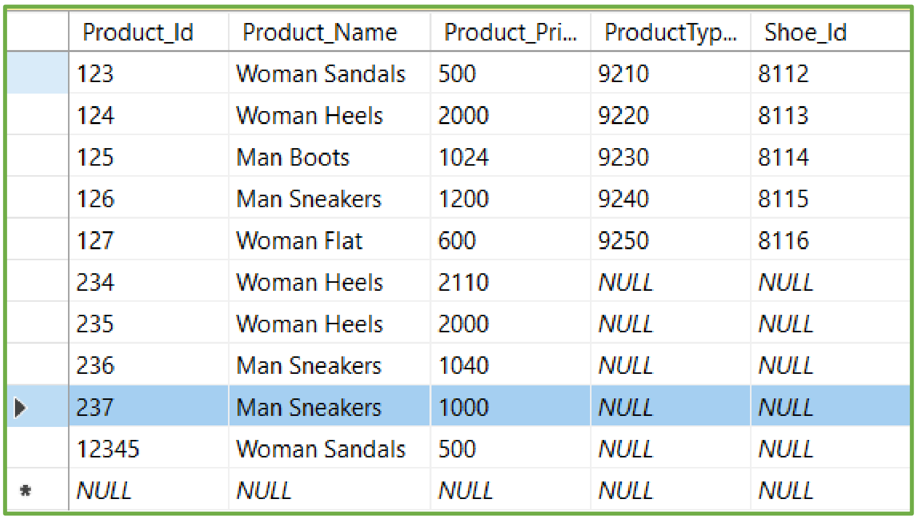
	Product_Id	Product_Name	Product_Pri...	ProductTyp...	Shoe_Id
	123	Woman Sandals	500	9210	8112
	124	Woman Heels	2000	9220	8113
	125	Man Boots	1024	9230	8114
	126	Man Sneakers	1200	9240	8115
	127	Woman Flat	600	9250	8116
	234	Woman Heels	2110	NULL	NULL
	235	Woman Heels	2000	NULL	NULL
▶	236	Man Sneakers	1040	NULL	NULL
	12345	Woman Sandals	500	NULL	NULL
*	NULL	NULL	NULL	NULL	NULL

→ Fourth test value:

```
INSERT INTO Product (Product_Id,Product_Name,Product_Price)
VALUES (237,'Man Sneakers',1000)
```



```
The price entered is correct

(1 row affected)

(1 row affected)
```



	Product_Id	Product_Name	Product_Pri...	ProductTyp...	Shoe_Id
	123	Woman Sandals	500	9210	8112
	124	Woman Heels	2000	9220	8113
	125	Man Boots	1024	9230	8114
	126	Man Sneakers	1200	9240	8115
	127	Woman Flat	600	9250	8116
	234	Woman Heels	2110	NULL	NULL
	235	Woman Heels	2000	NULL	NULL
	236	Man Sneakers	1040	NULL	NULL
▶	237	Man Sneakers	1000	NULL	NULL
	12345	Woman Sandals	500	NULL	NULL
*	NULL	NULL	NULL	NULL	NULL

5. Conclusion

Our user-friendly shop database system allows easy access and storage of input data. The system stores details on the customers, orders, products, product types, shoes, invoices and employees. All the tables above are normalised till third normal form (3NF). This assignment satisfies all the criteria including system definitions, system designs and the implementations and thus has been successfully implemented. Moreover, SQL codes and screenshots are included in the document to provide further understanding of the implementation process.

6. References

W3 Resource. *MySQL Triggers - w3resource*. [Online]. Available from: https://www.w3resource.com/mysql/mysql-triggers.php [Accessed: 31 May 2021a].

W3 School. *SQL Stored Procedures*. [Online]. Available from: https://www.w3schools.com/sql/sql_stored_procedures.asp [Accessed: 31 May 2021b].