Software Solution for Metro Ticket System

BSc Software Engineering Level 1
1012Y

Lecturer: Mrs. Soulakshmee Nagowah SIS
University of Mauritius

Prashant Dursun	**2114647**
Dharamveer Shamboo	**2118333**
Vaibhav Ramdass	**2114819**
Esaie Moos	**2114762**

Submission date: 21 JULY 2022

Table of Contents

1. INTRODUCTION

1.1 Overall Description

The proposed system will be an online booking system that allows people to book a ticket and see available destination, departure time and dates without having to go there and queue for their turn. Moreover, the customer will have to register for an account first and then be able to book, reschedule, view details or cancel their reservation. Everything will be done online.

1.2 Important features

1.21 Registering for an account:

The customer will enter the system and be allowed to create username and a password and will then be asked to enter personal details which will be stored in the customer relation.

1.2.2 Making a booking

The customer logs into the system using the username and password chosen by himself or herself while registering for an account. The system will validate credentials and a list of destination, departure time and dates will be provided for user selection.

Upon confirmation of the booking, a text message will be sent to the customer's phone number which will be a reference for the booking scheduled.

The booking is recorded in the booking relation.

1.2.3 Payment

After booking, the customer goes to the payment section and the total amount will be displayed to him or her. The customer choses which type of payment he or she wants to do. A transactionID will be given to the customer as a result of successful payment.

The payment is recorded in the payment relation.

1.3 Scope

1. The scope of the system is targeted for busy people for example; people going on a vacation, shift employees and anyone in general.
2. The system will be in the form of an online website since most people have access to internet nowadays.

3. All payment such as booking and total amount will be done online itself.

1.4 Roles

- Client

2. USER AND SYSTEM REQUIREMENTS

2.1 User Requirements

- The customer shall be able to register for an account by inserting his or her personal information.
- The customer shall be able to make a reservation.
- The customer should be able to update his or her personal details.
- The customer shall be able to view the total amount due.

2.2 Functional Requirements

- The system shall let a new customer to register an account by clicking on the 'Register' button and inserting his/her relevant information.
- The system shall store the information inserted by the customer in customer database.
- The system shall let the customer to choose his/her username and password when registering his/her account.
- The system shall verify the username and password inserted by the customer against the credentials on the database upon login before granting access to the customer.
- The system shall allow the customer to make his/her reservation by selecting appropriate date and time.
- The system shall store the current reservation date and time in the booking database.
- The system shall send a confirmation message to the customer whenever a booking is made.
- The system shall record all the information regarding in the booking database whenever a booking is made.
- The system shall message the information about any adjustments done concerning booking to the customer.

2.3 Non-Functional Requirements

- The system shall check that the username is unique while registering for an account.
- The system shall update the database within 5 seconds of any booking being made, cancelled or postponed.
- The system shall ensure that phone number entered is 8 digits long and is valid.

3. DESIGN

3.1 Entity Relationship Diagram

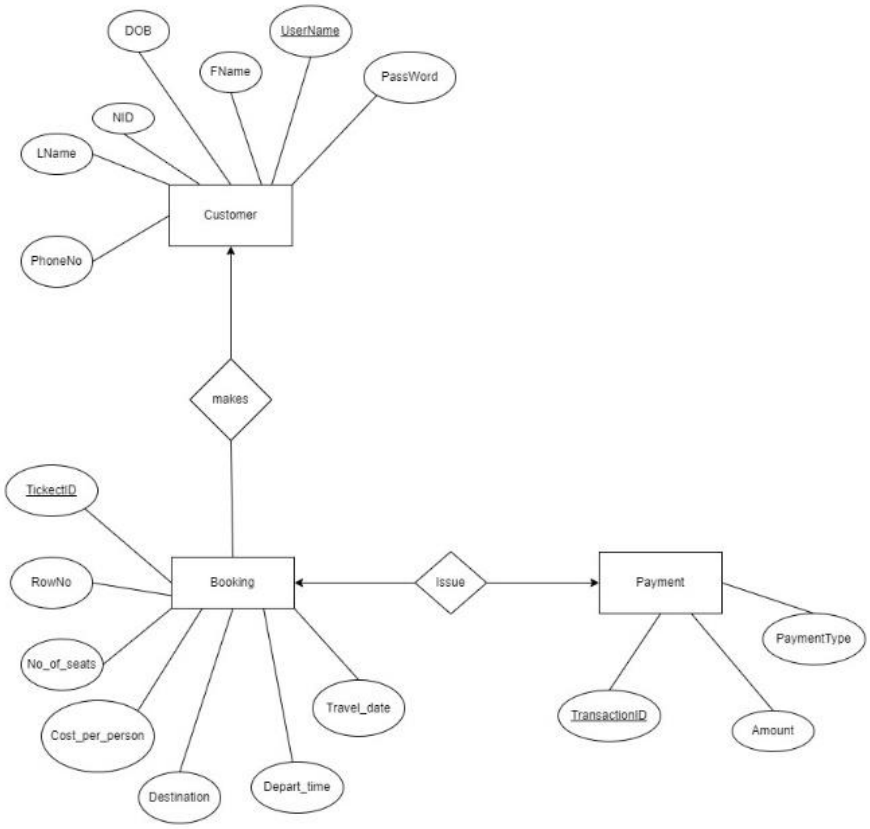


Figure 1: ERD

3.2 Schema (Reduction to Tables From ERD)

Customer (Username, Password, NID, FName, Lname, Sex, DOB, PhoneNo)

Booking (TicketID, No_of_Seats, RowNo, Destination, Depart_time, Cost_per_person, Travel_Date)

Payment (TransactionID, Amount, PaymentType)

3.2 Metro Express Ticket Reservation Table

NID	FName	Lname	Sex	DOB	PhoneNo	Username	Password	TicketID	BookingTime	No_of_Seats	RowNo	Destination	Depart_time	Cost_per_Person	Travel_Date	TransactionI D	Amount	PaymentType
1234	Sam	Don	M	12/09/1989	5678902	Sam_Don6	1234	001	12:00	1	4	Rose-Belle	13:00	35	24/10/2022	1111	35	Juice
2345	King	Philip	M	15/07/1991	5234971	KPhillip23	9875	002	16:00	3	6	Vacoas	16:00	40	2/12/2022	1112	120	Credit card
6789	Green	Island	F	17/05/2000	5209473	GIsland98	Fs-09	003	17:30	5	8	Port-louis	18:00	50	29/11/2022	1113	250	Credit card
1112	Jack	Daniels	M	12/12/1992	5792037	Joker09	mH09	004	09:30	2	4	Bambous	10:00	25	25/12/2022	1114	50	juice

3.4 Normalization

0NF

Reservation (NID, FName, LName, Sex, DOB, PhoneNo, Username, Password, TicketID, No_of_Seats, RowNo, Destination, Depart_time, Cost_per_Person, Travel_Date, TransactionID, Amount, PaymentType)

Possible primary keys: Username(unique), TicketID

1NF

Calculated Field: Amount (No_of_Seats * Cost_per_Person)

Reservation (Username, NID, FName, LName, Sex, DOB, PhoneNo, Password, **TicketID**, No_of_Seats, RowNo, Destination, Depart_time, Cost_per_Person, Travel_Date, TransactionID, PaymentType)

Functional dependencies

Username → Password, NID, FName, LName, Sex, DOB, PhoneNo, TransactionID, PaymentType, Cost_per_Person, No_of_Seats

TicketID → RowNo, Destination, Depart_time, Travel_Date

2NF

Customer (Username, Password, NID, FName, LName, Sex, DOB, PhoneNo, TransactionID, PaymentType, Cost_per_Person, No_of_Seats, TicketID)

Ticket (TicketID, RowNo, Destination, Depart_time, Travel_Date)

Booking (Username, TicketID, BookingTime)

Transitive dependencies

TransactionID → PaymentType, Cost_per_Person

3NF

Customer (Username, Password, NID, FName, LName, Sex, DOB, PhoneNo, No_of_Seats, TransactionID, TicketID)

Booking (Username, TicketID, BookingTime)

Payment (TransactionID, PaymentType, Cost_per_Person)

Ticket (TicketID, RowNo, Destination, Depart_time, Travel_Date)

Assumptions:

1. BookingTime is the initial time input by customer
2. Amount is a calculated field (Amount= No_of_seats * Cost_per_Person)

4. Implementation

4.1 TABLE DESCRIPTIONS:

Customer Table Definition:

CUsername	Unique client's username	VARCHAR	20
CPassword	Client's password	VARCHAR	40
NID	Client's NID	VARCHAR	4
FName	Client's First name	VARCHAR	40
LName	Client's last name	VARCHAR	40
Sex	Client's sex: male(M) or female(F)	CHAR	1
DOB	Client's date of birth	DATE	
PhoneNo	Client's phone Number	INT	7
No_of_seats	The number of tickets book by customer	INT	

TicketID	A unique code assign to every tickect bought	INT	3
TransactionID	A unique code assign to every transaction made	INT	4

Table 2: Customer Table Definition

Booking Table Definition:

Field Name	Description	Field Type	Field Size
CUsername	Unique client's username	VARCHAR	20
TicketID	A unique code assign to every tickect bought	INT	3
BookingTime	The initial time of travel requested by Customer	Time	

Table 3: Booking Table Definition

Payment Table Definition

Field Name	Description	Field Type	Field Size
TransactionID	A unique code assign to every transaction made	INT	4
PaymentType	The type of payment opted by the customer (Juice or Credit Card)	VARCAHR	11
Cost_per_Person	The travelling fee of one passenger	Real	

Table 4: Payment Table Definition

Customer Ticket Definition

Field Name	Description	Field Type	Field Size
TicketID	A unique code assign to every ticket bought	INT	3
RowNo	The row number where the customer was assigned	INT	
Depart_time	Time the Metro will leave the station	TIME	
Travel_Date	Date of reservation	DATE	

Table 5: Ticket Table Definition

4.2 SQL CODES:

1. DDL: DATA DEFINITION LANGUAGE (CREATE, ALTER, DROP).

CREATE COMMAND:

(i) Creating Table Customer

```
CREATE TABLE Customer(
CUsername VARCHAR(20) PRIMARY KEY NOT NULL,
CPassword VARCHAR(20) NOT NULL,
NID VARCHAR(14) NOT NULL,
```

```
FName VARCHAR(40) NOT NULL,
LName VARCHAR(40) NOT NULL,
Sex CHAR(1) CHECK(sex IN('M', 'F')),
DOB DATE,
PhoneNo INT NOT NULL,
No_of_seats INT NOT NULL,
TicketID INT FOREIGN KEY REFERENCES Ticket(TicketID),
TransactionID INT FOREIGN KEY REFERENCES Payment(TransactionID));
```

Username	Password	NID	FName	LName	Sex	DOB	PhoneNo	No_of_seats	TicketID	TransactionID

Figure 2: Table Customer

(ii) Creating Table Booking

```
CREATE TABLE Booking(
availableDates DATE NOT NULL,
availableDepartTime TIME NOT NULL,
Username VARCHAR(40) FOREIGN KEY REFERENCES Customer(Username),
TransactionID INT FOREIGN KEY REFERENCES Payment(TransactionID));
```

BookingTime	Username	TransactionID

Figure 3: Table Booking

(iii) Creating Table Payment

```
CREATE TABLE Payment(
TransactionID INT PRIMARY KEY NOT NULL,
PaymentType VARCHAR(15) NOT NULL,
Cost_per_Person REAL NOT NULL);
```

TransactionID	PaymentType	Cost_per_Person

Figure 4: Table Payment

(iv) Creating Table Ticket

```
CREATE TABLE Ticket(
TickeID INT PRIMARY KEY NOT NULL,
RowNo INT NOT NULL,
Destination VARCHAR(25) NOT NULL,
Depart_time TIME NOT NULL,
Travel_Date DATE NOT NULL);
```

DROP TABLE:

We already had some issues when creating the tables then ending up dropping Booking Table to add more stuffs, so we used the ALTER SQL Commands:

```
DROP TABLE Booking;
```

ALTER TABLE:

(i) Adding a new column, EmailAddress to have contact info in case the customer is not answering his/her phone

```
ALTER TABLE Customer
ADD EmailAddress VARCHAR(40) CHECK(EmailAddress like '%@%.%') NOT NULL;
```

(ii) Adding a constraint to make sure that the PaymentType is either 'Juice' or 'Credit card'.

```
ALTER TABLE Payment
ADD CHECK (PaymentType = 'Juice' OR PaymentType = 'Credit card');
```

b. DML: DATA MANIPULATION LANGUAGE (SELECT, INSERT, UPDATE, DELETE).

INSERT COMMAND:

(i) INSERT Table Customer

```
INSERT INTO Customer
VALUES ('Joker09', 'mH09', 1112, 'Jack', 'Daniels',
'M', '12/12/1992', 5792037, 2, 004, 1114, 'jack34@umail.com');
```

```
INSERT INTO Customer
VALUES('Sam_Don6', 1234,1234, 'Sam', 'Don',
'M', '12/09/19981', 5678902, 1, 001, 1111, 'sam123@umail.com');
```

CUsername	CPassword	NID	FName	LName	Sex	DOB	PhoneNo	No_of_seats	TicketID	TransactionID	EmailAddress
Joker09	mH09	1112	Jack	Daniels	M	1992-12-12	5792037	2	4	1114	jack34@umail.com
Sam_Don6	1234	1234	Sam	Don	M	1998-12-09	5678902	1	1	1111	sam123@umail.com

(ii) INSERT table Booking

```
INSERT INTO Booking
VALUES ('12:00', 'SamDon06', 1111);
```

Results Messages

	BookingTime	Username	TransactionID
1	12:00	SamDon06	1111

(iii) INSERT table Payment

```
INSERT INTO Payment
VALUES (1111, 'Juice', 35);

INSERT INTO Payment
VALUES (1112, 'Credit card', 40);

INSERT INTO Payment
VALUES (1113, 'Credit card', 50);

INSERT INTO Payment
VALUES (1114, 'Juice', 25);
```

Results Messages

	TransactionID	PaymentType	Cost_per_Person
1	1111	Juice	35
2	1112	Credit card	40
3	1113	Credit card	50
4	1114	Juice	25

(iv) INSERT Table Ticket

```
INSERT INTO Ticket
VALUES (001, 4, 'Rose-Belle', 13:00, 24/10/2022);

INSERT INTO Ticket
VALUES (002, 6, 'Vacoas', 16:00, 02/12/2022);

INSERT INTO Ticket
VALUES (003, 8, 'Port-Louis', 18:00, 29/11/2022);
```

Results Messages

	BookingTime	Username	TransactionID
1	12:00	SamDon06	1111
2	16:00	KPhillip23	1112
3	17:30	Glsland98	1113
4	09:300	Joker09	1114

SELECT COMMAND:

Display all tables

Customer Table:

```
SELECT *
FROM Customer;
```

Results Messages

	CUsername	CPassword	NID	FName	LName	Sex	DOB	PhoneNo	No_of_seats	TicketID	TransactionID	EmailAddress
1	Glsland09	fs-09	6789	Green	Island	F	0	5209473	5	3	1113	Island@umail.com
2	Joker09	mH09	1112	Jack	Daniels	M	12/12/1992	5792037	2	4	1114	jack34@umail.com
3	KPhillip23	9817	2345	King	Phillip	M	15/07/1991	5234971	3	2	1112	Kphillip@umail.com
4	Sam_Don6	1234	1234	Sam	Don	M	12/09/1998	5678902	1	1	1111	sam123@umail.com

Booking Table:

```
SELECT *
FROM Booking;
```

Results Messages

	TransactionID	PaymentType	Cost_per_Person
1	1111	Juice	35
2	1112	Credit card	40
3	1113	Credit card	50
4	1114	Juice	25

Payment Table:

```
SELECT *
FROM Payment;
```

Results Messages

	TickeID	RowNo	Destination	Depart_time	Travel_Date
1	1	4	Rose-Belle	13:00	24/10/2022
2	2	6	Vacoas	16:00	02/12/2022
3	3	8	Port-Louis	18:00	29/11/2022
4	4	2	Bambous	10:00	25/12/2022

Ticket Table:

```
SELECT *
FROM Ticket;
```

Results | Messages

	TickelD	RowNo	Destination	Depart_time	Travel_Date
1	1	4	Rose-Belle	13:00	24/10/2022
2	2	6	Vacoas	16:00	02/12/2022
3	3	8	Port-Louis	18:00	29/11/2022

- List the contact Details (phone number and email address) of Mr. Sam Don.

```
SELECT PhoneNO, EmailAddress
FROM Customer
WHERE FName = 'Sam' AND LName = 'Don';
```

Results | Messages

	PhoneNO	EmailAddress
1	5678902	sam123@umail.com

- Display the first and last name of the customer with ticket

```
SELECT FName, LName
FROM Customer
WHERE TicketID= 001;
```

Results | Messages

	FName	LName
1	Sam	Don

- Display the total number of customers who has use the system to book a ticket.

```
SELECT COUNT(CUsername) AS TotalCustomers
From Customer;
```

Results | Messages

	TotalCustomers
1	4

- Display the NID of every customer who used MCB ‘Juice’ to pay

```
SELECT NID, TransactionID
FROM Customer
WHERE TransactionID IN
(SELECT TransactionID
FROM Payment
WHERE PaymentType='Juice');
```

Results | Messages

	NID	TransactionID
1	1112	1114
2	1234	1111

- Display the username of every customer who is travelling to ‘Rose-Belle’ and bought tickets for more than 2 seats and paid by ‘credit card’

```
SELECT Username,TicketID, Destination, PaymentType
FROM Customer c, Ticket t, Payment p
WHERE c.TicketID = t.TicketID AND t.Destination = 'Rose-Belle' AND
p.Payment = 'Credit Card';
```

Results | Messages

	CUsername	TicketID	Destination	PaymentType
1	Sam_Don6	1	Rose-Belle	Credit card

UPDATE COMMAND:

- Update the destination of Ticket ID 1113 from 'Port-Louis' to 'Dagotiere'

```
UPDATE Ticket
SET Destination = 'Dagotiere'
WHERE TicketID= 0003;
```

Results | Messages

	TickelD	RowNo	Destination	Depart_time	Travel_Date
1	1	4	Rose-Belle	13:00	24/10/2022
2	2	6	Vacoas	16:00	02/12/2022
3	3	8	Dagotiere	18:00	29/11/2022
4	4	2	Bambous	10:00	25/12/2022

- Update the username of Sam Don from Sam_Don6 to Sam123 and update his phone number to 58074567

```
UPDATE Customer
SET Username = 'Sam123'
WHERE FName = 'Sam' And LName = 'Don';

UPDATE Customer
SET PhoneNo = '50874567'
WHERE FName = 'Sam' And LName = 'Don';
```

Results | Messages

	CUsername	CPassword	NID	FName	LName	Sex	DOB	PhoneNo	No_of_seats	TicketID	TransactionID	EmailAddress
1	Glsland09	fs-09	6789	Green	Island	F	0	5209473	5	3	1113	Island@umail.com
2	Joker09	mH09	1112	Jack	Daniels	M	12/12/1992	5792037	2	4	1114	jack34@umail.com
3	KPhillip23	9817	2345	King	Phillip	M	15/07/1991	5234971	3	2	1112	Kphillip@umail.com
4	Sam123	1234	1234	Sam	Don	M	12/09/1998	50874567	1	1	1111	sam123@umail.com

Delete Command:

- Mr. Jack Daniels decided to erase all his personal information, delete his information.

```
DELETE FROM Customer
WHERE FName = 'Jack' AND LName = 'Daniels';
```

Messages

```
(1 row affected)

Completion time: 2022-07-21T14:02:08.4302559+04:00
```

Results | Messages

	CUsername	CPassword	NID	FName	LName	Sex	DOB	PhoneNo	No_of_seats	TicketID	TransactionID	EmailAddress
1	Glsland09	fs-09	6789	Green	Island	F	0	5209473	5	3	1113	Island@umail.com
2	KPhillip23	9817	2345	King	Phillip	M	15/07/1991	5234971	3	2	1112	Kphillip@umail.com
3	Sam123	1234	1234	Sam	Don	M	12/09/1998	50874567	1	1	1111	sam123@umail.com

STORED PROCEDURES

(I) Stored procedure to insert a new customer

```
CREATE PROCEDURE sp_customer @username VARCHAR(10), @password VARCHAR(20), @nid
VARCHAR(14), @fname VARCHAR(40), @lname VARCHAR(40), @sex CHAR(1), @dob DATE, @phone
INTEGER, @seats INTEGER, @ticketID INTEGER, @transID INTEGER
AS
BEGIN
BEGIN TRY
           INSERT INTO Customer

(Username,Password,Nid,Fname,Lname,Sex,Dob,PhoneNo,No_of_seats,TicketID,TransactionID)
           VALUES
          (@username, @password, @nid, @fname, @lname, @sex, @dob, @phone, @seats,
@ticketID, @transID)
          PRINT 'Insert Successful'
END TRY
     BEGIN CATCH
     SELECT
     ERROR_MESSAGE() AS ErrorMessage;
     END CATCH
END
```

The stored procedure will be executed by the below code:

```
EXEC sp_customer @username='Sam_Don6', @password=1234, @nid=1234, @fname='Sam',
@lname='Don', @sex='M', @dob='1998/09/12', @phone=5678902, @seats=3, @ticketID=1112,
@transID=002;
```

Results Messages

	Username	Password	NID	FName	LName	Sex	DOB	PhoneNo	No_of_seats	TicketID	TransactionID
1	Sam_	1234	1234	Sam	Don	M	1998-09-12	5678902	3	1112	2

(II) Stored procedure to insert a new payment

```
CREATE PROCEDURE sp_payment @transID INTEGER, @type VARCHAR(15), @cost REAL
AS
BEGIN
BEGIN TRY
           INSERT INTO Payment
           (TransactionID,PaymentType,Cost_per_Person)
           VALUES
           (@transID, @type, @cost)
           Print 'Insert Successful'
     END TRY
           BEGIN CATCH
           SELECT
           ERROR_MESSAGE() AS ErrorMessage;
           END CATCH
END
```

The stored procedure will be executed by the below code:

```
EXEC sp_payment @transID=1111, @type='Juice', @cost=35;
```

Results | Messages

	TransactionID	PaymentType	Cost_per_Person
1	1111	Juice	35

(III) Stored procedure to insert a new ticket

```
CREATE PROCEDURE sp_ticket @ID INTEGER, @rowNo INTEGER, @destination VARCHAR(25),
@depart TIME, @date
AS
BEGIN
BEGIN TRY
            INSERT INTO Ticket
            (TickeID, RowNo, Destination,Depart_time,Travel_Date)
            VALUES
            (@ID, @rowNo, @destination, @depart, @date)
            Print 'Insert Successful'
 END TRY
           BEGIN CATCH
           SELECT
           ERROR_MESSAGE() AS ErrorMessage;
           END CATCH
END
```

The stored procedure will be executed by the below code:

```
EXEC sp_ticket @ID=001, @rowNo=4, @destination= 'Rose-Belle', @depart='13:00:00',
@date='2022/10/24';
```

Results | Messages

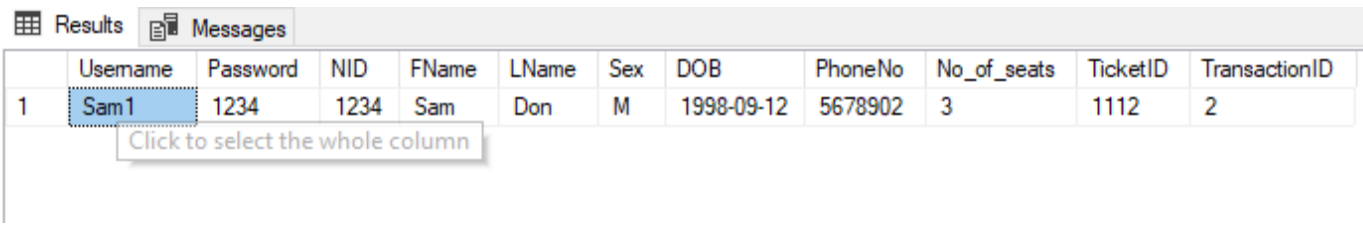
	TickeID	RowNo	Destination	Depart_time	Travel_Date
1	1	4	Rose-Belle	13:00:00.0000000	2022-10-24

(IV) Stored procedure sp_updatecustomer which updates Customer Table

```
CREATE PROCEDURE sp_updatecustomer (@username varchar (10), @fname varchar (10),
@lname varchar (10))
AS
BEGIN
UPDATE Customer
SET Username = @username
WHERE Fname = @fname AND Lname = @lname
END
```

The stored procedure will be executed by the below code:

```
EXEC sp_updatecustomer 'Sam1', 'Sam', 'Don'
```



Results | Messages

	Username	Password	NID	FName	LName	Sex	DOB	PhoneNo	No_of_seats	TicketID	TransactionID
1	Sam1	1234	1234	Sam	Don	M	1998-09-12	5678902	3	1112	2

Click to select the whole column

(V) Stored procedure sp_deletecustomer which deletes a record from Customer Table.

```
CREATE PROCEDURE sp_deletecustomer @fname varchar (10), @lname varchar (10)
AS
BEGIN
DELETE FROM Customer
WHERE Fname = @fname AND Lname = @lname
END
```

The stored procedure will be executed by the below code:

```
EXEC sp_deletecustomer 'Jack', 'Daniels'
```

Before:

Results Messages

	Username	Password	NID	FName	LName	Sex	DOB	PhoneNo	No_of_seats	TicketID	TransactionID
1	jak	1234	1234	Jack	Daniels	M	1998-07-12	59318248	3	1112	2
2	Sam_	1234	1234	Same	Don	M	1998-09-12	5678902	3	1112	2
3	Sam1	1234	1234	Sam	Don	M	1998-09-12	5678902	3	1112	2

After:

Results Messages

	Username	Password	NID	FName	LName	Sex	DOB	PhoneNo	No_of_seats	TicketID	TransactionID
1	Sam_	1234	1234	Same	Don	M	1998-09-12	5678902	3	1112	2
2	Sam1	1234	1234	Sam	Don	M	1998-09-12	5678902	3	1112	2

TRIGGERS

(i) TRIGGER TO CHECK CLIENT

```
CREATE TRIGGER tg_check_client
ON Customer
INSTEAD OF INSERT
AS
DECLARE @FName VARCHAR(40)
DECLARE @LName VARCHAR(40)

SET @FName = (Select FName from INSERTED)
SET @LName = (Select LName from INSERTED)
IF @FName IS NULL
BEGIN
     PRINT 'Please enter First Name.'
END
IF @LName IS NULL
BEGIN
     PRINT 'Please enter Last Name.'
END
IF EXISTS (Select LName from Customer) AND EXISTS (Select @LName from INSERTED)
BEGIN
     IF EXISTS (Select FName from Customer) AND EXISTS (Select @FName from INSERTED)

     BEGIN
     PRINT 'Client already exist in database!'
        END
END


DROP TRIGGER tg_check_client
```

(ii) TRIGGER TO DELETE BOOKING

```
ON Booking
AFTER DELETE, UPDATE
AS

DECLARE @availableDates DATE
DECLARE @availableDepartTime TIME(15)
DECLARE @Username VARCHAR(25)
DECLARE @TransactionID INT

SET @availableDates = (SELECT availableDates from DELETED)
SET @availableDepartTime = (SELECT availableDepartTime from DELETED)
SET @Username = (SELECT Username from DELETED)
SET @TransactionID = (SELECT TransactionID from DELETED)

INSERT INTO Booking_Old(availableDates, availableDepartTime, Username, TransactionID)
SELECT @availableDates, @availableDepartTime, @Username, @TransactionID from DELETED

DROP TRIGGER tg_old_Booking

DROP TABLE Booking_Old
```

Results | Messages

	availableDates	availableDepartTime	Username	TransactionID
1	2015-02-20	10:30:00	koo4	6969

(iii) TRIGGER TO VERIFY BOOKING DETAILS

```
CREATE TRIGGER tg_old_Booking
CREATE TRIGGER tg_display_new_booking_details
ON Ticket
AFTER INSERT, UPDATE
AS
DECLARE @TicketID INTEGER
DECLARE @Destination VARCHAR(25)
DECLARE @Depart_time DATE(15)
DECLARE @Travel_Date DATE(15)

SET @TicketID = (Select TicketID from INSERTED)
SET @Destination = (Select Destination from INSERTED)
SET @Depart_time = (Select Depart_time from INSERTED)
SET @Travel_Date = (Select Travel_Date from INSERTED)

PRINT 'Successfully updated travel arrangements!'

PRINT  @TicketID
PRINT 'Destination: ' + @Destination
PRINT  @Depart_time
PRINT  @Travel_Date


DROP TRIGGER tg_display_new_booking_details
```

```
Messages
  Successfully updated travel arrangements!
  1
  Destination: Rose-Belle
  13:00
  24/10/2022

  (1 row affected)

  Completion time: 2022-07-21T12:08:00.2151527+04:00
```

(iv) TRIGGER TO DELETE AND UPDATE PAYMENTS

```
CREATE TABLE Payment_Old
(TransactionID INT,
 PaymentType VARCHAR(15),
 Cost_per_Person REAL);

CREATE TRIGGER tg_old_Payment
ON Payment
AFTER DELETE
AS

DECLARE @TransactionID INT
DECLARE @PaymentType VARCHAR(15)
DECLARE @Cost_per_Person REAL


SET @TransactionID = (SELECT TransactionID from DELETED)
SET @Cost_per_Person = (SELECT Cost_per_Person from DELETED)
SET @PaymentType = (SELECT PaymentType from DELETED)


INSERT INTO Payment_Old(TransactionID, Cost_per_Person, PaymentType)
SELECT @TransactionID, @Cost_per_Person, @PaymentType from DELETED

DROP TRIGGER tg_old_Payment

DROP TABLE Payment_Old
```

After Deletion:

Results | Messages

	TransactionID	PaymentType	Cost_per_Person
1	1111	Juice	35

After Update:

Results | Messages

	TransactionID	PaymentType	Cost_per_Person
1	1112	Credit card	40

(v) TRIGGER TO DELETE CUSTOMER DETAILS

```
CREATE TABLE Customer_Old
(Username VARCHAR(25),
 PhoneNo INT,
 No_of_seats INT);


CREATE TRIGGER tg_old_Customer
ON Customer
AFTER DELETE
AS

DECLARE @Username VARCHAR(25)
DECLARE @PhoneNo INT
DECLARE @No_of_seats INT


SET @Username= (SELECT Username from DELETED)
SET @PhoneNo = (SELECT PhoneNo from DELETED)
SET @No_of_seats = (SELECT No_of_seats from DELETED)

INSERT INTO Customer_Old(Username, PhoneNo, No_of_seats)
SELECT @Username, PhoneNo, No_of_seats from DELETED

DROP TRIGGER tg_old_Customer

DROP TABLE Customer_Old
```

Results | Messages

	Username	PhoneNo	No_of_seats
1	Joker09	5792037	2

Conclusion

While most transactions are performed virtually, this concept on metro reservation should help quite a lot of people timewise moreover giving them the benefit of not displacing to book a ticket and queue for their turn.

The database was modelled and designed using UML diagrams and tables, constraints, stored procedures and triggers with the tables normalised till the third normal form (3NF). The database was implemented using Microsoft SQL Server and test data were input to test the tables. Some stored procedures were also created to implement with the table.

In the end, the whole database was successfully executed based on the design and the expected outputs were received.