

DBMS-LAB ASSIGNMENT 7

NAME: KASHA SINGH

REG. NO.: 19BCS051

1. Write two stored Procedures relevant to your database.

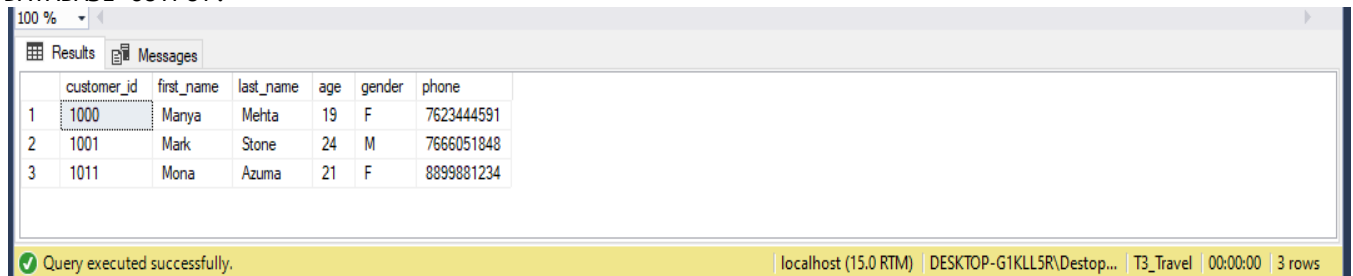
QUERY:

```
USE T3_Travel;
```

```
CREATE PROCEDURE Names_starting_with_M
AS
SELECT * FROM T3_CustomerDetails WHERE first_name LIKE 'M%'
GO
```

```
EXEC Names_starting_with_M;
```

DATABASE OUTPUT:



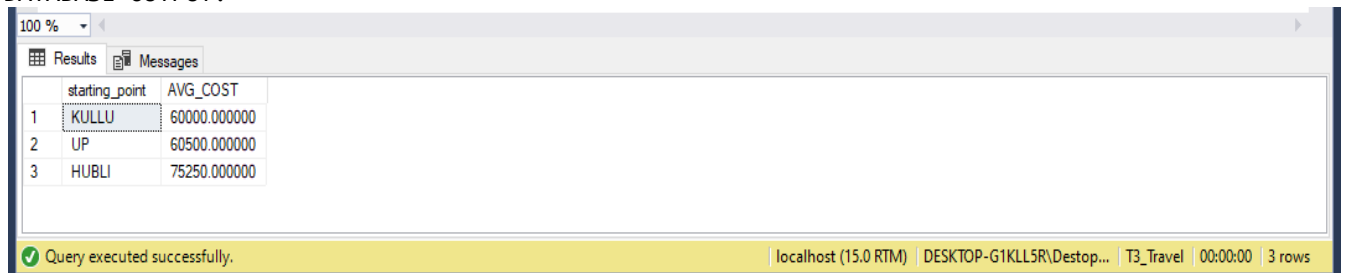
	customer_id	first_name	last_name	age	gender	phone
1	1000	Manya	Mehta	19	F	7623444591
2	1001	Mark	Stone	24	M	7666051848
3	1011	Mona	Azuma	21	F	8899881234

Query executed successfully. | localhost (15.0 RTM) | DESKTOP-G1KLL5R\Destop... | T3_Travel | 00:00:00 | 3 rows

```
CREATE PROCEDURE PackageAvgCost
AS
SELECT starting_point, AVG(cost) AS AVG_COST FROM T3_PackageDetails
GROUP BY starting_point ORDER BY AVG_COST
GO
```

```
EXEC PackageAvgCost;
```

DATABASE OUTPUT:



	starting_point	AVG_COST
1	KULLU	60000.000000
2	UP	60500.000000
3	HUBLI	75250.000000

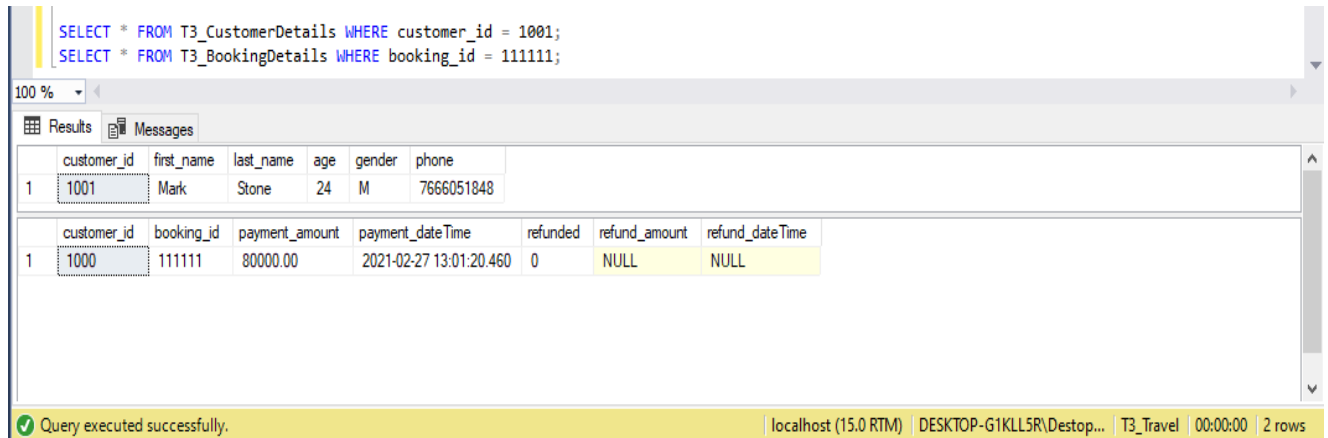
Query executed successfully. | localhost (15.0 RTM) | DESKTOP-G1KLL5R\Destop... | T3_Travel | 00:00:00 | 3 rows

2. Write a transaction to illustrate atomicity (related to your database)

QUERY:

```
--Atomicity--  
BEGIN TRAN Transaction_Update  
UPDATE T3_CustomerDetails SET age = 24 WHERE customer_id = 1001  
UPDATE T3_BookingDetails SET payment_amount = 80000 where booking_id = 111111  
COMMIT
```

DATABASE OUTPUT:



The screenshot shows a SQL Server query window with the following SQL code:

```
SELECT * FROM T3_CustomerDetails WHERE customer_id = 1001;  
SELECT * FROM T3_BookingDetails WHERE booking_id = 111111;
```

The results pane displays two tables. The first table, T3_CustomerDetails, has columns: customer_id, first_name, last_name, age, gender, and phone. The second table, T3_BookingDetails, has columns: customer_id, booking_id, payment_amount, payment_dateTime, refunded, refund_amount, and refund_dateTime.

customer_id	first_name	last_name	age	gender	phone
1001	Mark	Stone	24	M	7666051848

customer_id	booking_id	payment_amount	payment_dateTime	refunded	refund_amount	refund_dateTime
1000	111111	80000.00	2021-02-27 13:01:20.460	0	NULL	NULL

The status bar at the bottom indicates: Query executed successfully. | localhost (15.0 RTM) | DESKTOP-G1KLL5R\Destop... | T3_Travel | 00:00:00 | 2 rows

As the transaction is atomic, both of the updates on the two separate tables will commit together, or they will rollback together.

3. Write a transaction to illustrate isolation level. It can be on commit or uncommit read (related to your database)

Window 1:

```
--Isolation Property--  
USE T3_Travel;  
GO  
BEGIN TRAN Trans_Isolation  
  
UPDATE T3_CustomerDetails  
SET last_name = 'Bond'  
WHERE customer_id = 1105
```

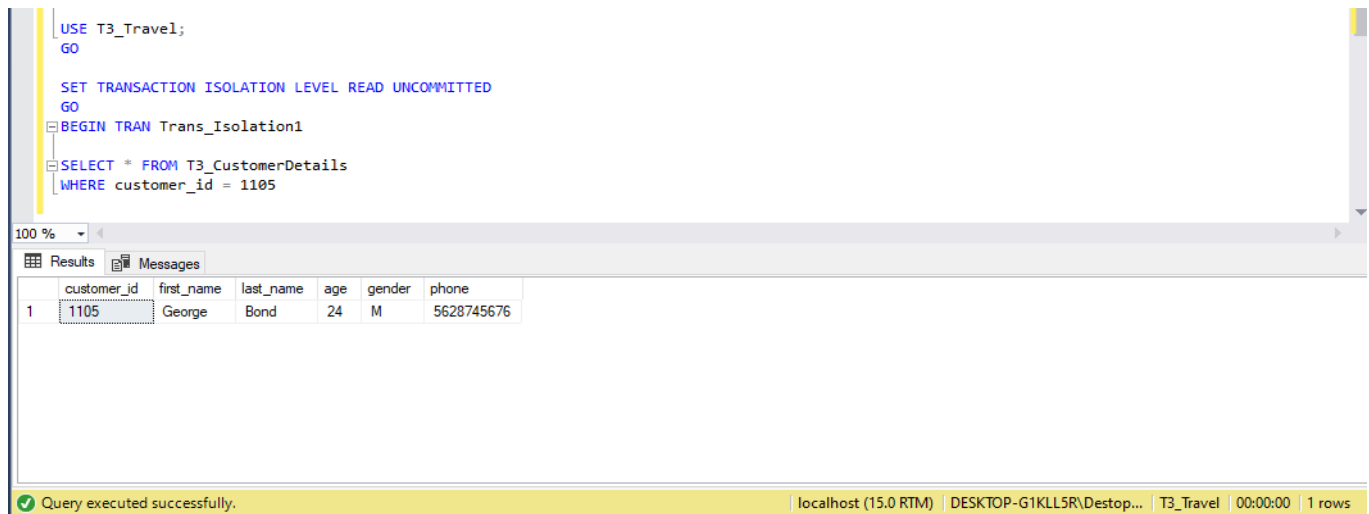
Window 2:

```
USE T3_Travel;  
GO
```

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
GO
BEGIN TRAN Trans_Isolation1

SELECT * FROM T3_CustomerDetails
WHERE customer_id = 1105
```

DATABASE OUTPUT:



The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays a query script with the following commands:

```
USE T3_Travel;
GO

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
GO
BEGIN TRAN Trans_Isolation1
SELECT * FROM T3_CustomerDetails
WHERE customer_id = 1105
```

The bottom pane shows the results of the query in a table format. The table has 6 columns: customer_id, first_name, last_name, age, gender, and phone. There is one row of data.

	customer_id	first_name	last_name	age	gender	phone
1	1105	George	Bond	24	M	5628745676

At the bottom of the window, a status bar indicates: "Query executed successfully. localhost (15.0 RTM) | DESKTOP-G1KLL5R\Destop... | T3_Travel | 00:00:00 | 1 rows".

When we set the isolation level to read uncommitted, we will be able to see the last_name set to 'Bond', called Dirty Read.