

DBMS-LAB ASSIGNMENT 7

NAME: DEEPTI P RANJOLKAR

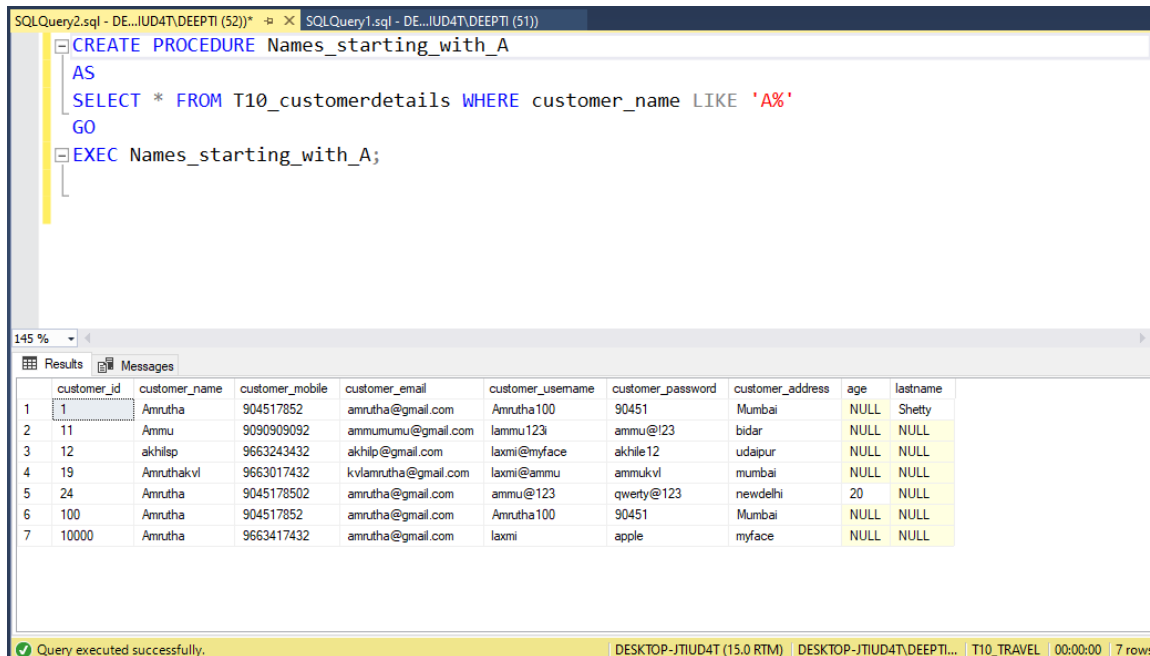
REG. NO.: 19BCS037

1.write two stored Procedures relevant to your database.

QUERY:

```
CREATE PROCEDURE Names_starting_with_A
AS
SELECT * FROM T10_customerdetails WHERE customer_name LIKE 'A%'
GO
EXEC Names_starting_with_A;
```

OUTPUT:



The screenshot displays the SQL Server Enterprise Manager interface. The top pane shows the SQL script for the stored procedure 'Names_starting_with_A', which selects all records from the 'T10_customerdetails' table where the 'customer_name' starts with 'A'. The bottom pane shows the results of the query execution, displaying a table with 7 rows of customer data.

customer_id	customer_name	customer_mobile	customer_email	customer_username	customer_password	customer_address	age	lastname
1	Amrutha	904517852	amrutha@gmail.com	Amrutha100	90451	Mumbai	NULL	Shetty
11	Ammu	9090909092	ammumumu@gmail.com	lammu123	ammu@123	bidar	NULL	NULL
12	akhlip	9663243432	akhlip@gmail.com	laxmi@myface	akhile12	udaipur	NULL	NULL
19	Amruthakvl	9663017432	kvlamrutha@gmail.com	laxmi@ammu	ammukvl	mumbai	NULL	NULL
24	Amrutha	9045178502	amrutha@gmail.com	ammu@123	qwerty@123	newdelhi	20	NULL
100	Amrutha	904517852	amrutha@gmail.com	Amrutha100	90451	Mumbai	NULL	NULL
10000	Amrutha	9663417432	amrutha@gmail.com	laxmi	apple	myface	NULL	NULL

Query executed successfully. DESKTOP-JTIUD4T (15.0 RTM) DESKTOP-JTIUD4T,DEEPTI... T10_TRAVEL 00:00:00 7 rows

QUERY:

```
CREATE PROCEDURE PackageAvgCost
AS SELECT package_amount, AVG(package_amount) AS AVG_COST FROM
T10_packagedetails
GROUP BY package_amount ORDER BY AVG_COST
GO
EXEC PackageAvgCost;
```

OUTPUT:

The screenshot shows a SQL Server Enterprise Manager window with two tabs: 'SQLQuery2.sql - DE...IUD4T\DEEPTI (52))' and 'SQLQuery1.sql - DE...IUD4T\DEEPTI (51)'. The active tab contains the following SQL code:

```
CREATE PROCEDURE PackageAvgCost
AS SELECT package_amount, AVG(package_amount) AS AVG_COST FROM
T10_packagedetails
GROUP BY package_amount ORDER BY AVG_COST
GO
EXEC PackageAvgCost;
```

Below the code editor, the 'Results' tab is selected, displaying a table with two columns: 'package_amount' and 'AVG_COST'. The table contains 7 rows of data:

	package_amount	AVG_COST
1	100	100
2	300	300
3	400	400
4	500	500
5	600	600
6	700	700
7	1000	1000

At the bottom of the window, a status bar indicates 'Query executed successfully.' and provides additional information: 'DESKTOP-JTIUD4T (15.0 RTM) | DESKTOP-JTIUD4T\DEEPTI... | T10_TRAVEL | 00:00:00 | 7 rows'.

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

Query:

```
BEGIN TRAN Transaction_Update
UPDATE T10_CustomerDetails SET age = 21 WHERE customer_id = 11
UPDATE T10_packagedetails SET package_amount = 5000 where customer_id = 11
COMMIT
```

```
select * from T10_customerdetails where customer_id = 11
select * from T10_packagedetails where customer_id = 11
```

output

SQLQuery2.sql - DE...IUD4T\DEEPTI (52)) * SQLQuery1.sql - DE...IUD4T\DEEPTI (51))

```

BEGIN TRAN Transaction_Update
UPDATE T10_CustomerDetails SET age = 21 WHERE customer_id = 11
UPDATE T10_packagedetails SET package_amount = 5000 where customer_id = 11
COMMIT

select * from T10_customerdetails where customer_id = 11
select * from T10_packagedetails where customer_id = 11

```

145 %

Results Messages

	customer_id	customer_name	customer_mobile	customer_email	customer_username	customer_password	customer_address	age	lastname
1	11	Ammu	9090909092	ammumumu@gmail.com	lammu123i	ammu@!23	bidar	21	NULL

	package_id	package_tour_id	package_name	package_amount	package_total	package_type	package_description	customer_id
1	10010	100001	hotel	5000	900	cultural	NULL	11

Query executed successfully. | DESKTOP-JTIUD4T (15.0 RTM) | DESKTOP-JTIUD4T\DEEPTI... | T10_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:

QUERY:

```

USE T10_TRAVEL;
GO
BEGIN TRAN Trans_Isolation
UPDATE T10_customerdetails
SET customer_name = 'JAMES'
WHERE customer_id = 18

```

WINDOW 2:

QUERY:

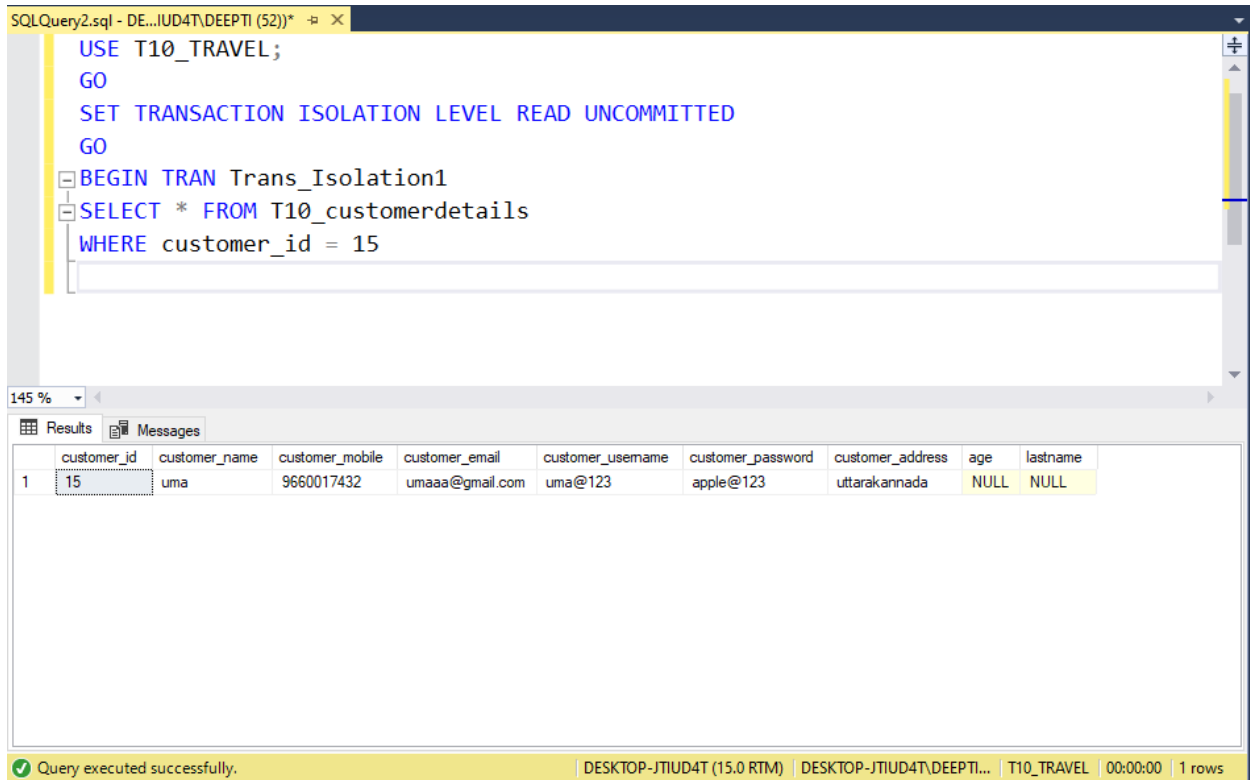
```

USE T10_TRAVEL;
GO
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
GO
BEGIN TRAN Trans_Isolation1

```

```
SELECT * FROM T10_customerdetails
WHERE customer_id = 15
```

OUTPUT:



The screenshot shows a SQL Server Enterprise Manager window with a query window titled 'SQLQuery2.sql - DE...IUD4T\DEEPTI (52))'*. The query window contains the following SQL code:

```
USE T10_TRAVEL;
GO
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
GO
BEGIN TRAN Trans_Isolation1
SELECT * FROM T10_customerdetails
WHERE customer_id = 15
```

Below the query window, the 'Results' tab is active, displaying a table with 10 columns: customer_id, customer_name, customer_mobile, customer_email, customer_username, customer_password, customer_address, age, and lastname. The table contains one row with the following data:

	customer_id	customer_name	customer_mobile	customer_email	customer_username	customer_password	customer_address	age	lastname
1	15	uma	9660017432	umaaa@gmail.com	uma@123	apple@123	uttarakannada	NULL	NULL

At the bottom of the window, a status bar indicates 'Query executed successfully.' and provides details about the execution: 'DESKTOP-JTIUD4T (15.0 RTM) | DESKTOP-JTIUD4T\DEEPTI... | T10_TRAVEL | 00:00:00 | 1 rows'.

When we set the isolation level to read uncommitted, we will be able to see the customer_name set to 'JAMES', called Dirty Read