

Name:Vaishnavi Vinod Ingle

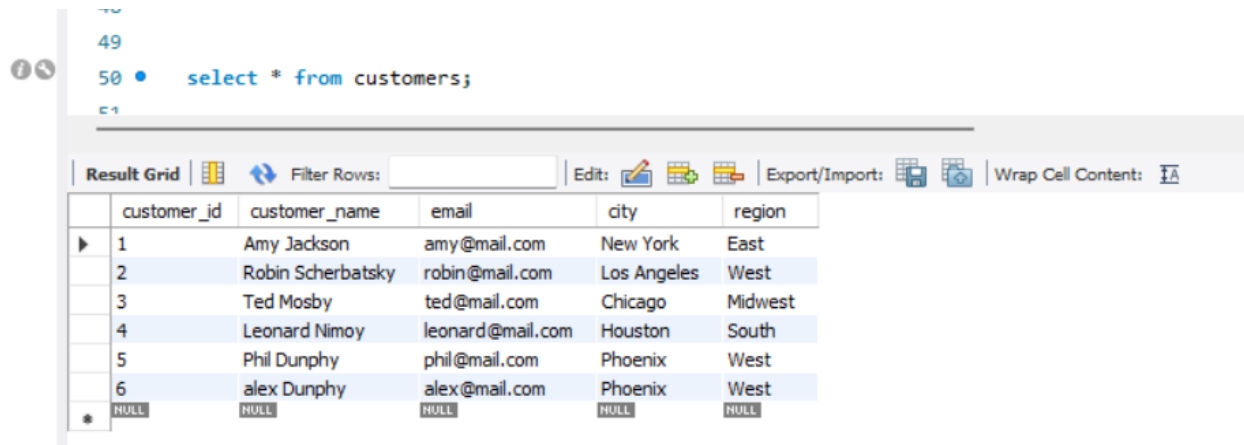
Mail:vaishnavingle54@gmail.com

"**Assignment 1:** Write a SELECT query to retrieve all columns from a 'customers' table, and modify it to return only the customer name and email address for customers in a specific city.

Solution:

select * from customers;

select customer_name,email from customers where city="phoenix";



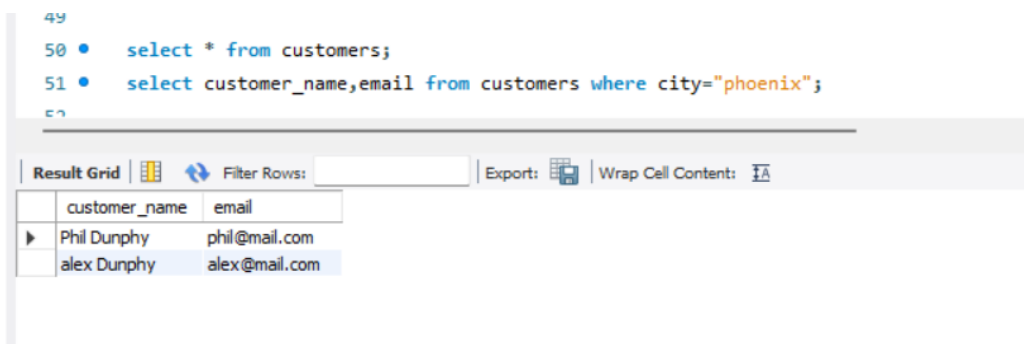
The screenshot shows a SQL IDE interface. The query editor contains the following SQL code:

```
49
50 • select * from customers;
51
```

The result grid displays the following data:

	customer_id	customer_name	email	city	region
▶	1	Amy Jackson	amy@mail.com	New York	East
	2	Robin Scherbatsky	robin@mail.com	Los Angeles	West
	3	Ted Mosby	ted@mail.com	Chicago	Midwest
	4	Leonard Nimoy	leonard@mail.com	Houston	South
	5	Phil Dunphy	phil@mail.com	Phoenix	West
	6	alex Dunphy	alex@mail.com	Phoenix	West
*	NULL	NULL	NULL	NULL	NULL

ictice
dent
dentdb
;
lo
odb



The screenshot shows a SQL IDE interface. The query editor contains the following SQL code:

```
49
50 • select * from customers;
51 • select customer_name,email from customers where city="phoenix";
52
```

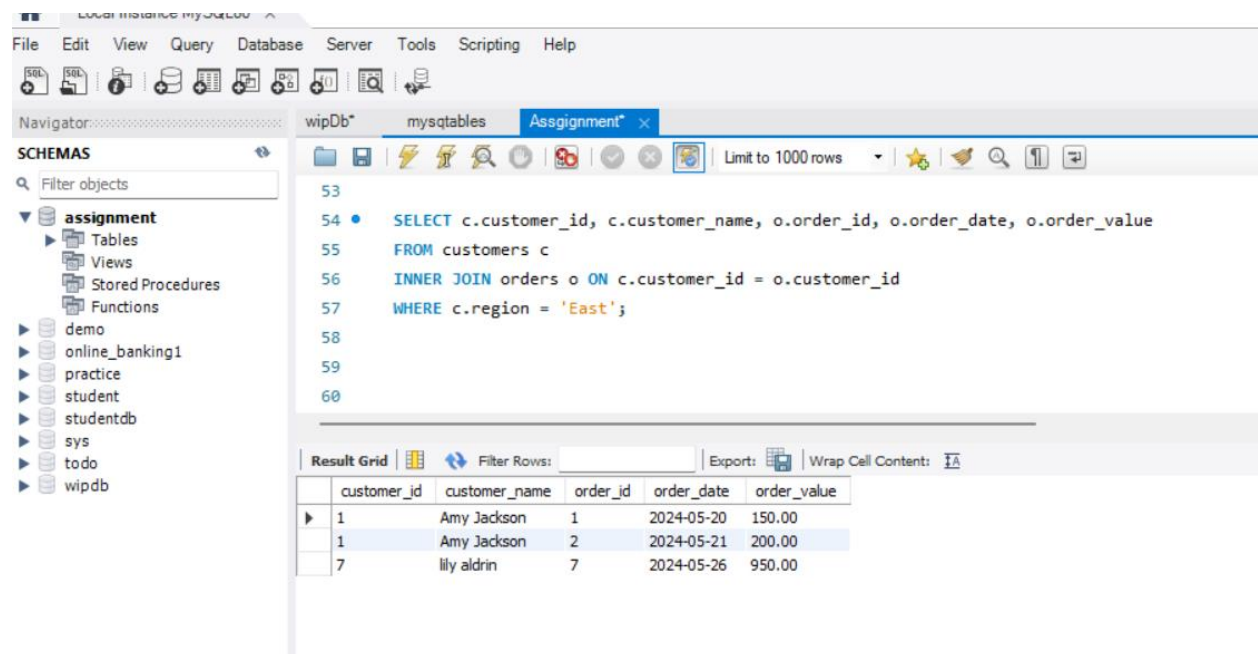
The result grid displays the following data:

	customer_name	email
▶	Phil Dunphy	phil@mail.com
	alex Dunphy	alex@mail.com

Assignment 2: Craft a query using an INNER JOIN to combine 'orders' and 'customers' tables for customers in a specified region, and a LEFT JOIN to display all customers including those without orders.

Solution:

```
SELECT c.customer_id, c.customer_name, o.order_id, o.order_date, o.order_value
FROM customers c
INNER JOIN orders o ON c.customer_id = o.customer_id
WHERE c.region = 'East';
```



The screenshot shows a MySQL IDE interface. On the left, the 'SCHEMAS' panel lists databases including 'assignment', 'demo', 'online_banking1', 'practice', 'student', 'studentdb', 'sys', 'todo', and 'wipdb'. The 'assignment' database is selected. The main editor displays a SQL query:

```
53
54 • SELECT c.customer_id, c.customer_name, o.order_id, o.order_date, o.order_value
55 FROM customers c
56 INNER JOIN orders o ON c.customer_id = o.customer_id
57 WHERE c.region = 'East';
58
59
60
```

Below the query editor, the 'Result Grid' shows the following data:

	customer_id	customer_name	order_id	order_date	order_value
▶	1	Amy Jackson	1	2024-05-20	150.00
	1	Amy Jackson	2	2024-05-21	200.00
	7	lily aldrin	7	2024-05-26	950.00

```
SELECT customers.customer_id, customers.customer_name, orders.order_id, orders.order_date,
orders.order_value
FROM customers
LEFT JOIN orders ON customers.customer_id = orders.customer_id;
```

Filter objects

assignment

Tables

Views

Stored Procedures

Functions

demo

online_banking1

practice

student

studentdb

sys

todo

wipdb

57 WHERE c.region = 'East';

58

59

60 • SELECT customers.customer_id, customers.customer_name, orders.oi

61 FROM customers

62 LEFT JOIN orders ON customers.customer_id = orders.customer_id;

63

64

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

	customer_id	customer_name	order_id	order_date	order_value
▶	1	Amy Jackson	1	2024-05-20	150.00
	1	Amy Jackson	2	2024-05-21	200.00
	2	Robin Scherbatsky	3	2024-05-22	250.00
	3	Ted Mosby	4	2024-05-23	300.00
	4	Leonard Nimoy	5	2024-05-24	350.00
	5	Phil Dunphy	NULL	NULL	NULL
	6	alex Dunphy	6	2024-05-24	350.00
	7	lily aldrin	7	2024-05-26	950.00

Result Grid

Form Editor

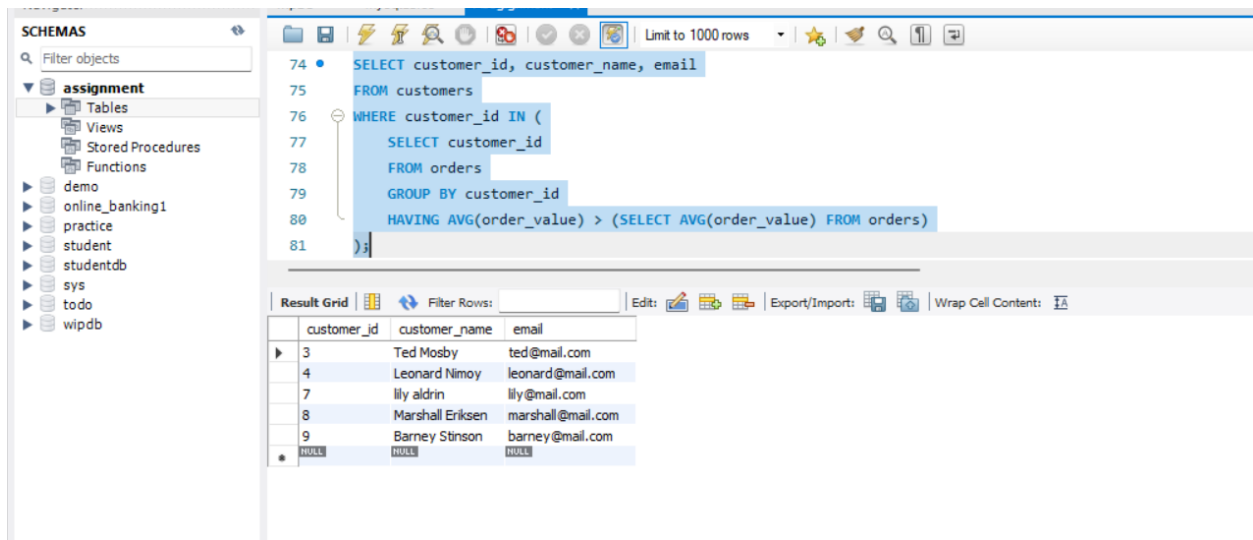
Field Types

Result 2 x Read Only

Assignment 3: Utilize a subquery to find customers who have placed orders above the average order value, and write a UNION query to combine two SELECT statements with the same number of columns.

Solution:

```
SELECT customer_id, customer_name, email
FROM customers
WHERE customer_id IN (
    SELECT customer_id
    FROM orders
    GROUP BY customer_id
    HAVING AVG(order_value) > (SELECT AVG(order_value) FROM orders)
);
```



The screenshot shows a database IDE interface. On the left, the 'SCHEMAS' panel lists various databases including 'assignment', 'demo', 'online_banking1', 'practice', 'student', 'studentdb', 'sys', 'todo', and 'wipdb'. The main editor displays a SQL query that filters customers based on their average order value. The query is as follows:

```
74 SELECT customer_id, customer_name, email
75 FROM customers
76 WHERE customer_id IN (
77     SELECT customer_id
78     FROM orders
79     GROUP BY customer_id
80     HAVING AVG(order_value) > (SELECT AVG(order_value) FROM orders)
81 );
```

Below the query editor, the 'Result Grid' shows the output of the query. It contains a table with three columns: 'customer_id', 'customer_name', and 'email'. The results are as follows:

customer_id	customer_name	email
3	Ted Mosby	ted@mail.com
4	Leonard Nimoy	leonard@mail.com
7	lily aldrin	lily@mail.com
8	Marshall Eriksen	marshall@mail.com
9	Barney Stinson	barney@mail.com
HIDE	HIDE	HIDE

```
SELECT customer_id, customer_name, email
FROM customers
WHERE city = 'New York'
UNION
SELECT customer_id, customer_name, email
FROM customers
```

WHERE city = 'Los Angeles';

The screenshot shows a database management tool interface. On the left is a 'Navigator' pane with a 'SCHEMAS' tree. The 'assignment' schema is expanded, showing 'Tables', 'Views', 'Stored Procedures', and 'Functions'. Below these are several database schemas: demo, online_banking1, practice, student, studentdb, sys, todo, and wipdb. The main window is titled 'wipDb* mysqltables Assignment*'. It contains a SQL editor with the following query:

```
83
84 • SELECT customer_id, customer_name, email
85 FROM customers
86 WHERE city = 'New York'
87 UNION
88 SELECT customer_id, customer_name, email
89 FROM customers
90 WHERE city = 'Los Angeles';
```

Below the editor is a 'Result Grid' showing the results of the query. It has columns for 'customer_id', 'customer_name', and 'email'. The results are as follows:

customer_id	customer_name	email
1	Amy Jackson	amy@mail.com
9	Barney Stinson	barney@mail.com
2	Robin Scherbatsky	robin@mail.com

Assignment 4: Compose SQL statements to BEGIN a transaction, INSERT a new record into the 'orders' table, COMMIT the transaction, then UPDATE the 'products' table, and ROLLBACK the transaction.

use Assignment;

Start TRANSACTION;

INSERT INTO orders (order_id, customer_id, order_date, order_value)
VALUES (14, 9, '2024-06-01', 1000.00);

COMMIT;

Ranking: 99

db

```
100 • Start TRANSACTION;
101
102
103 • INSERT INTO orders (order_id, customer_id, order_date, order_value)
104   VALUES (14, 9, '2024-06-01', 1000.00);
105
106
107 • COMMIT;
```

Schemas

selected

Output

Action Output

#	Time	Action	Message
✓ 1	08:36:01	Start TRANSACTION	0 row(s) affected
✗ 2	08:36:11	INSERT INTO orders (order_id, customer_id, order_date, order_value) VALUES (14, 9, '2024-06-01', 1000.00)	Error Code: 1046. No database selected Select the default DB to be used
✓ 3	08:36:21	use assignment	0 row(s) affected
✓ 4	08:36:25	Start TRANSACTION	0 row(s) affected
✓ 5	08:36:30	INSERT INTO orders (order_id, customer_id, order_date, order_value) VALUES (14, 9, '2024-06-01', 1000.00)	1 row(s) affected
✓ 6	08:36:35	COMMIT	0 row(s) affected

Start TRANSACTION;

UPDATE products

SET price = price + 100






WHERE product_id = 1;




ROLLBACK;

```

110 • Start TRANSACTION;
111
112
113 • UPDATE products
114     SET price = price + 100
115     WHERE product_id = 1;
116 • select* from products;

```

Result Grid   Filter Rows: <input type="text"/> Edit:   				
	product_id	product_name	price	stock
	1	Laptop	1099.99	50
	2	Smart TV	799.99	30
	3	AirPods	199.99	150
	4	iPad	499.99	75
	5	Headphones	99.99	200
	NULL	NULL	NULL	NULL

products 1 x <input type="text"/>		
Output 		
 Action Output 		
#	Time	Action
4	08:36:25	Start TRANSACTION
5	08:36:30	INSERT INTO orders (order_id, customer_id, order_date, orde
6	08:36:35	COMMIT
7	08:37:40	Start TRANSACTION
8	08:37:47	UPDATE products SET price = price + 100 WHERE product_
9	08:38:01	select* from products LIMIT 0, 1000



```
117
118 • ROLLBACK;
119
120 • select* from products;
121
```

Result Grid   Filter Rows: | Edit:    | Export/Import:   | Wrap Cell Content: 

	product_id	product_name	price	stock
▶	1	Laptop	999.99	50
	2	Smart TV	799.99	30
	3	AirPods	199.99	150
	4	iPad	499.99	75
	5	Headphones	99.99	200
•	NULL	NULL	NULL	NULL

products 2 x

Output

 Action Output 

	#	Time	Action	Message
✓	6	08:36:35	COMMIT	0 row(s) affected
✓	7	08:37:40	Start TRANSACTION	0 row(s) affected
✓	8	08:37:47	UPDATE products SET price = price + 100 WHERE product_id = 1	1 row(s) affected Rows matched: 1
✓	9	08:38:01	select* from products LIMIT 0, 1000	5 row(s) returned
✓	10	08:38:42	ROLLBACK	0 row(s) affected
✓	11	08:38:53	select* from products LIMIT 0, 1000	5 row(s) returned

Assignment 5: Begin a transaction, perform a series of INSERTs into 'orders', setting a SAVEPOINT after each, rollback to the second SAVEPOINT, and COMMIT the overall transaction.

```
start TRANSACTION;
```

```
INSERT INTO orders (order_id, customer_id, order_date, order_value)
```

```
VALUES (15, 8, '2024-06-02', 1100.00);
```

```
SAVEPOINT savepoint1;
```

The screenshot shows a database IDE with a SQL script in the editor and its execution results in the Output pane. The script includes a SELECT statement followed by a transaction block starting with START TRANSACTION, an INSERT INTO statement, a SAVEPOINT, and a ROLLBACK. The Output pane shows the execution of these statements with their respective messages and row counts.

```
119
120 • select* from products;
121
122 • Start TRANSACTION;
123 • INSERT INTO orders (order_id, customer_id, order_date, order_value)
124   VALUES (15, 8, '2024-06-02', 1100.00);
125 • SAVEPOINT savepoint1;
126
127
```

product_id	product_name	price	stock
1	Laptop	999.99	50
2	Smart TV	799.99	30
3	AirPods	199.99	150
4	iPad	499.99	75
5	Headphones	99.99	200
NULL	NULL	NULL	NULL

#	Time	Action	Message
✓ 10	08:38:42	ROLLBACK	0 row(s) affected
✓ 11	08:38:53	select* from products LIMIT 0, 1000	5 row(s) returned
✓ 12	08:40:20	select* from products LIMIT 0, 1000	5 row(s) returned
✓ 13	08:40:20	Start TRANSACTION	0 row(s) affected
✓ 14	08:40:20	INSERT INTO orders (order_id, customer_id, order_date, order_value) VALUES (15, 8, '2024-06-02', 1100.00)	1 row(s) affected
✓ 15	08:40:20	SAVEPOINT savepoint1	0 row(s) affected

The screenshot shows a database IDE with a SQL script in the editor and its execution results in the Output pane. The script includes a SELECT statement followed by a transaction block starting with START TRANSACTION, an INSERT INTO statement, a SAVEPOINT, and a ROLLBACK. The Output pane shows the execution of these statements with their respective messages and row counts.

```
119
120 • select* from products;
121
```

product_id	product_name	price	stock
1	Laptop	999.99	50
2	Smart TV	799.99	30
3	AirPods	199.99	150
4	iPad	499.99	75
5	Headphones	99.99	200
NULL	NULL	NULL	NULL

```
INSERT INTO orders (order_id, customer_id, order_date, order_value)
```

```
VALUES (16, 9, '2024-06-03', 1200.00);
```

```
SAVEPOINT savepoint2;
```

The screenshot displays a database management tool interface. On the left, a sidebar lists various database objects including 'todo' and 'wipdb'. The main window is divided into two panes. The top pane shows the execution log with the following entries:

#	Time	Action	Message
13	08:40:20	Start TRANSACTION	0 row(s) affected
14	08:40:20	INSERT INTO orders (order_id, customer_id, order_date, order_value) VALUES (15, 8, '2024-06-02', 1100.00)	1 row(s) affected
15	08:40:20	SAVEPOINT savepoint1	0 row(s) affected
16	08:42:30	select*from products LIMIT 0, 1000	5 row(s) returned
17	08:43:09	INSERT INTO orders (order_id, customer_id, order_date, order_value) VALUES (16, 9, '2024-06-03', 1200.00)	1 row(s) affected
18	08:43:09	SAVEPOINT savepoint2	0 row(s) affected

The bottom pane shows a SQL editor with the following code:

```
127 • SAVEPOINT savepoint1;  
128  
129  
130 • INSERT INTO orders (order_id, customer_id, order_date, order_value)
```

Below the editor is a 'Result Grid' showing a table with 4 columns: 'order_id', 'customer_id', 'order_date', and 'order_value'. The table contains 12 rows of data:

order_id	customer_id	order_date	order_value
1	1	2024-05-20	150.00
2	1	2024-05-21	200.00
3	2	2024-05-22	250.00
4	3	2024-05-23	300.00
5	4	2024-05-24	350.00
6	6	2024-05-24	350.00
7	7	2024-05-26	950.00
8	5	2024-05-25	400.00
9	8	2024-05-26	500.00
10	9	2024-05-27	600.00
11	8	2024-05-28	700.00
12	3	2024-05-29	800.00

```
INSERT INTO orders (order_id, customer_id, order_date, order_value)
```

```
VALUES (17, 8, '2024-06-04', 1300.00);
```

```
SAVEPOINT savepoint3;
```

student
studentdb
sys
todo
wipdb

```
129
130 • INSERT INTO orders (order_id, customer_id, order_date, order_value)
131   VALUES (16, 9, '2024-06-03', 1200.00);
132 • SAVEPOINT savepoint2;
133 • select* from orders;
134
135 • INSERT INTO orders (order_id, customer_id, order_date, order_value)
136   VALUES (17, 8, '2024-06-04', 1300.00);
137 • SAVEPOINT savepoint3;
138 • select* from orders;
139
```

Administration Schemas

Information

No object selected

Object Info Session

Output

Action Output

#	Time	Action	Message
17	08:43:09	INSERT INTO orders (order_id, customer_id, order_date, order_value) VALUES (16, 9, '2024-06-03', 1200.00)	1 row(s) affected
18	08:43:09	SAVEPOINT savepoint2	0 row(s) affected
19	08:43:32	select* from products LIMIT 0, 1000	5 row(s) returned
20	08:44:37	select* from orders LIMIT 0, 1000	16 row(s) returned
21	08:45:07	INSERT INTO orders (order_id, customer_id, order_date, order_value) VALUES (17, 8, '2024-06-04', 1300.00)	1 row(s) affected
22	08:45:07	SAVEPOINT savepoint3	0 row(s) affected

anking1

Result Grid

Filter Rows:

Edit:

Export/Import:

	order_id	customer_id	order_date	order_value
▶	1	1	2024-05-20	150.00
	2	1	2024-05-21	200.00
	3	2	2024-05-22	250.00
	4	3	2024-05-23	300.00
	5	4	2024-05-24	350.00
	6	6	2024-05-24	350.00
	7	7	2024-05-26	950.00
	8	5	2024-05-25	400.00
	9	8	2024-05-26	500.00
	10	9	2024-05-27	600.00
	11	8	2024-05-28	700.00
	12	3	2024-05-29	800.00
	13	4	2024-05-30	900.00
	14	9	2024-06-01	1000.00

orders 7 ×

Schemas

Output

Action Output

```
INSERT INTO orders (order_id, customer_id, order_date, order_value)
VALUES (18, 9, '2024-06-05', 1400.00);
SAVEPOINT savepoint4;
```

Administration Schemas

Information

No object selected

```
137 • SAVEPOINT savepoint3;
138 • select* from orders;
139
140 • INSERT INTO orders (order_id, customer_id, order_date, order_value)
141 VALUES (18, 9, '2024-06-05', 1400.00);
142 • SAVEPOINT savepoint4;
143
144
```

Output

Action Output

#	Time	Action	Message
20	08:44:37	select* from orders LIMIT 0, 1000	16 row(s) returned
21	08:45:07	INSERT INTO orders (order_id, customer_id, order_date, order_value) VALUES (17, 8, '2024-06-04', 1300.00)	1 row(s) affected
22	08:45:07	SAVEPOINT savepoint3	0 row(s) affected
23	08:45:25	select* from orders LIMIT 0, 1000	17 row(s) returned
24	08:46:02	INSERT INTO orders (order_id, customer_id, order_date, order_value) VALUES (18, 9, '2024-06-05', 1400.00)	1 row(s) affected
25	08:46:02	SAVEPOINT savepoint4	0 row(s) affected

orders 8 x

order_id	customer_id	order_date	order_value
1	1	2024-05-20	150.00
2	1	2024-05-21	200.00
3	2	2024-05-22	250.00
4	3	2024-05-23	300.00
5	4	2024-05-24	350.00
6	6	2024-05-24	350.00
7	7	2024-05-26	950.00
8	5	2024-05-25	400.00
9	8	2024-05-26	500.00
10	9	2024-05-27	600.00
11	8	2024-05-28	700.00
12	3	2024-05-29	800.00
13	4	2024-05-30	900.00
14	9	2024-06-01	1000.00

Administration Schemas

Output

ROLLBACK TO SAVEPOINT savepoint2;

```
142 • SAVEPOINT savepoint4;
143
144 • ROLLBACK TO SAVEPOINT savepoint2;
145
```

Administration Schemas

Information

No object selected

Output

Action Output

#	Time	Action	Message
22	08:45:07	SAVEPOINT savepoint3	0
23	08:45:25	select* from orders LIMIT 0, 1000	1
24	08:46:02	INSERT INTO orders (order_id, customer_id, order_date, order_value) VALUES (18, 9, '2024-06-05', 1400.00)	1
25	08:46:02	SAVEPOINT savepoint4	0
26	08:46:19	select* from orders LIMIT 0, 1000	1
27	08:47:05	ROLLBACK TO SAVEPOINT savepoint2	0

Session

online_banking1

practice

student

studentdb

sys

todo

wipdb

Result Grid

Filter Rows:

Edit: | Export/Import: | Wrap Cell Content: |

	order_id	customer_id	order_date	order_value
▶	1	1	2024-05-20	150.00
	2	1	2024-05-21	200.00
	3	2	2024-05-22	250.00
	4	3	2024-05-23	300.00
	5	4	2024-05-24	350.00
	6	6	2024-05-24	350.00
	7	7	2024-05-26	950.00
	8	5	2024-05-25	400.00
	9	8	2024-05-26	500.00
	10	9	2024-05-27	600.00
	11	8	2024-05-28	700.00
	12	3	2024-05-29	800.00
	13	4	2024-05-30	900.00
	14	9	2024-06-01	1000.00

orders 9 x

Administration

Schemas

Information

No object selected

Output

Action Output

#	Time	Action	Message
✓ 23	08:45:25	select* from orders LIMIT 0, 1000	17 row(s) returned
✓ 24	08:46:02	INSERT INTO orders (order_id, customer_id, order_date, order_value) VALUES (18, 9, '2024-06-05', 1400.00)	1 row(s) affected
✓ 25	08:46:02	SAVEPOINT savepoint4	0 row(s) affected
✓ 26	08:46:19	select* from orders LIMIT 0, 1000	18 row(s) returned
✓ 27	08:47:05	ROLLBACK TO SAVEPOINT savepoint2	0 row(s) affected
✓ 28	08:47:22	select* from orders LIMIT 0, 1000	16 row(s) returned

Object Info

Session

COMMIT;

Assignment 6: Draft a brief report on the use of transaction logs for data recovery and create a hypothetical scenario where a transaction log is instrumental in data recovery after an unexpected shutdown.

Solution:

Transaction logs are crucial components in database management systems (DBMS) that help ensure data integrity and enable recovery from unexpected failures. They record all changes made to the database, including inserts, updates, deletes, and transaction controls such as commits and rollbacks.

Importance of Transaction Logs

Data Integrity: Transaction logs ensure that all database operations are completed successfully. If a system failure occurs, the transaction log can help identify incomplete transactions and roll back or complete them to maintain data consistency.

Recovery from Failures: In the event of a system crash, power failure, or unexpected shutdown, transaction logs provide a way to restore the database to its last consistent state.

Audit Trail: Transaction logs keep a detailed record of all database operations, which can be used for auditing purposes to track changes and identify unauthorized access or modifications.

How Transaction Logs Work

Recording Transactions: Every database operation (insert, update, delete) is recorded in the transaction log before being applied to the database. Each log entry includes details like the transaction ID, type of operation, affected data, and timestamps.

Commit and Rollback: When a transaction is committed, a commit record is added to the log, indicating that the changes are permanent. If a rollback is issued, a rollback record is added, and any changes made by the transaction are undone.

Crash Recovery: After a crash, the DBMS uses the transaction log to determine the state of active transactions at the time of the failure. It replays committed transactions to ensure they are fully applied and rolls back uncommitted transactions to restore the database to a consistent state.

Hypothetical Scenario: Data Recovery After an Unexpected Shutdown

Imagine an online store with a database that handles customer orders. During a busy shopping day, the database server experiences an unexpected shutdown due to a power failure. Here's how transaction logs can help recover the database:

Before the Shutdown: Several transactions are in progress. Customers are placing orders (inserts), updating their shipping addresses (updates), and some transactions are being committed while others are pending.

Unexpected Shutdown: The power failure causes an abrupt shutdown, leaving some transactions incomplete. The database is now in an inconsistent state, with some operations partially applied.

Restart and Recovery:

Analyzing the Log: When the server restarts, the DBMS reads the transaction log to identify all transactions that were active at the time of the shutdown.

Rolling Forward: The DBMS replays all the committed transactions from the log, ensuring that their changes are fully applied to the database.

Rolling Back: It then identifies transactions that were not committed and rolls back their changes to undo any partial operations.

Restored State: The database is now restored to its last consistent state, with all completed transactions applied and incomplete transactions undone, ensuring data integrity and consistency.

