

## SQL Day 3 Assignments:

**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.**

*Execution of the SQL commands:*

The screenshot displays the MySQL Workbench interface. The 'SQL Editor' window contains the following SQL script:

```
16
17 • START TRANSACTION;
18
19 -- Insert a new record into the 'orders' table
20 • INSERT INTO orders (customer_id, order_date, total_amount)
21   VALUES (123, '2024-05-13', 100.00);
22 -- Commit the transaction
23 • COMMIT;
24
25 • UPDATE products
26   SET quantity = quantity - 1
27   WHERE product_id = 1;
28
29 -- Rollback the transaction (if needed)
30 • ROLLBACK;
```

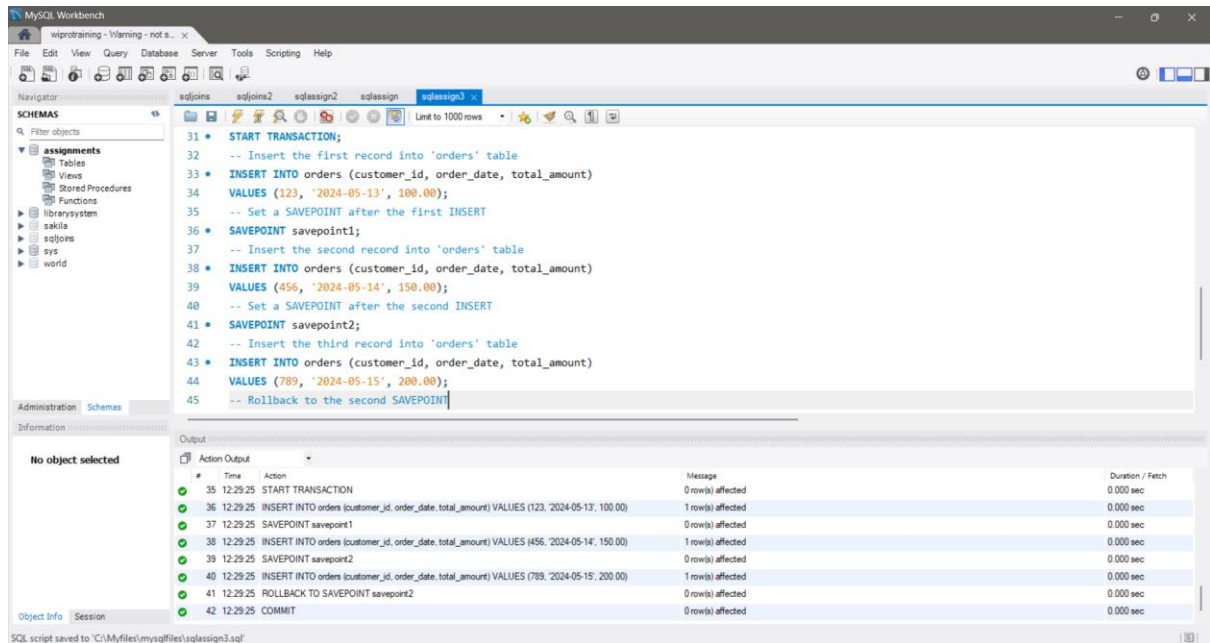
The 'Output' window at the bottom shows the execution results:

#	Time	Action	Message	Duration / Fetch
27	12:24:04	COMMIT	0 row(s) affected	0.000 sec
28	12:24:04	UPDATE products SET quantity = quantity - 1 WHERE product_id = 1	0 row(s) affected Rows matched: 0 Changed: 0 Warnings: 0	0.000 sec
29	12:24:04	ROLL	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL se...	0.000 sec
30	12:24:16	START TRANSACTION	0 row(s) affected	0.000 sec
31	12:24:16	INSERT INTO orders (customer_id, order_date, total_amount) VALUES (123, '2024-05-13', 100.00)	1 row(s) affected	0.000 sec
32	12:24:16	COMMIT	0 row(s) affected	0.016 sec
33	12:24:16	UPDATE products SET quantity = quantity - 1 WHERE product_id = 1	0 row(s) affected Rows matched: 0 Changed: 0 Warnings: 0	0.000 sec
34	12:24:16	ROLLBACK	0 row(s) affected	0.000 sec

SQL script saved to: 'C:\Myfiles\mysqlfiles\sqlassign3.sql'

**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.**

*Execution of the commands:*



The screenshot displays the MySQL Workbench interface. The SQL editor contains the following commands:

```
31 START TRANSACTION;
32 -- Insert the first record into 'orders' table
33 INSERT INTO orders (customer_id, order_date, total_amount)
34 VALUES (123, '2024-05-13', 100.00);
35 -- Set a SAVEPOINT after the first INSERT
36 SAVEPOINT savepoint1;
37 -- Insert the second record into 'orders' table
38 INSERT INTO orders (customer_id, order_date, total_amount)
39 VALUES (456, '2024-05-14', 150.00);
40 -- Set a SAVEPOINT after the second INSERT
41 SAVEPOINT savepoint2;
42 -- Insert the third record into 'orders' table
43 INSERT INTO orders (customer_id, order_date, total_amount)
44 VALUES (789, '2024-05-15', 200.00);
45 -- Rollback to the second SAVEPOINT
```

The Output tab shows the execution results:

#	Time	Action	Message	Duration / Fetch
35	12:29:25	START TRANSACTION	0 row(s) affected	0.000 sec
36	12:29:25	INSERT INTO orders (customer_id, order_date, total_amount) VALUES (123, '2024-05-13', 100.00)	1 row(s) affected	0.000 sec
37	12:29:25	SAVEPOINT savepoint1	0 row(s) affected	0.000 sec
38	12:29:25	INSERT INTO orders (customer_id, order_date, total_amount) VALUES (456, '2024-05-14', 150.00)	1 row(s) affected	0.000 sec
39	12:29:25	SAVEPOINT savepoint2	0 row(s) affected	0.000 sec
40	12:29:25	INSERT INTO orders (customer_id, order_date, total_amount) VALUES (789, '2024-05-15', 200.00)	1 row(s) affected	0.000 sec
41	12:29:25	ROLLBACK TO SAVEPOINT savepoint2	0 row(s) affected	0.000 sec
42	12:29:25	COMMIT	0 row(s) affected	0.000 sec

SQL script saved to: 'C:\Myfiles\mysqlfiles\sqlassign3.sql'

**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.**

*Execution of the given query:*

The screenshot displays the MySQL Workbench interface. The query editor at the top contains the following SQL query:

```
39
40 • SELECT c.customer_id, c.customer_name, o.order_id, o.order_date, o.total_amount
41 FROM customers c
42 LEFT JOIN orders o ON c.customer_id = o.customer_id
43 WHERE c.region = 'East';
44
45
```

The Result Grid below the query shows the following data:

	customer_id	customer_name	order_id	order_date	total_amount
1	Customer A	1	2024-05-01	100.00	
1	Customer A	3	2024-05-03	200.00	
1	Customer A	6	2024-05-01	100.00	
1	Customer A	8	2024-05-03	200.00	
3	Customer C				
6	Customer A				
8	Customer C				

The Output pane at the bottom shows the execution log with the following messages:

#	Time	Action	Message	Duration / Fetch
48	16.06.07	INSERT INTO orders (customer_id, order_date, total_amount) VALUES (1, '2024-05-01', 100.00), (2, '2024-05-03', 200.00), (3, '2024-05-01', 100.00), (4, '2024-05-03', 200.00), (5, '2024-05-01', 100.00), (6, '2024-05-03', 200.00), (7, '2024-05-01', 100.00), (8, '2024-05-03', 200.00);	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.015 sec
49	16.06.19	CREATE DATABASE IF NOT EXISTS sales_db;	1 row(s) affected, 1 warning(s): 1007 Can't create database 'sales_db': database exists	0.000 sec
50	16.06.19	USE sales_db;	0 row(s) affected	0.000 sec
51	16.06.19	CREATE TABLE IF NOT EXISTS customers ( customer_id INT AUTO_INCREMENT PRIMARY KEY, customer_name VARCHAR(50) NOT NULL, region VARCHAR(20) NOT NULL );	0 row(s) affected, 1 warning(s): 1050 Table 'customers' already exists	0.000 sec
52	16.06.19	CREATE TABLE IF NOT EXISTS orders ( order_id INT AUTO_INCREMENT PRIMARY KEY, customer_id INT NOT NULL, order_date DATE NOT NULL, total_amount DECIMAL(10,2) NOT NULL );	0 row(s) affected, 1 warning(s): 1050 Table 'orders' already exists	0.000 sec
53	16.06.19	INSERT INTO customers (customer_name, region) VALUES ('Customer A', 'East'), ('Customer B', 'West'), ('Customer C', 'North');	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.000 sec
54	16.06.19	INSERT INTO orders (customer_id, order_date, total_amount) VALUES (1, '2024-05-01', 100.00), (2, '2024-05-03', 200.00), (3, '2024-05-01', 100.00), (4, '2024-05-03', 200.00), (5, '2024-05-01', 100.00), (6, '2024-05-03', 200.00), (7, '2024-05-01', 100.00), (8, '2024-05-03', 200.00);	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.000 sec
55	16.06.19	SELECT c.customer_id, c.customer_name, o.order_id, o.order_date, o.total_amount FROM customers c LEFT JOIN orders o ON c.customer_id = o.customer_id WHERE c.region = 'East';	7 row(s) returned	0.000 sec / 0.000 sec

**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. Create necessary tables to run this query also.**

*Execution of the given query:*

The screenshot shows the MySQL Workbench interface. The query editor contains the following SQL code:

```

40 SELECT customer_id, customer_name
41 FROM customers
42 WHERE customer_id IN (
43     SELECT o.customer_id
44     FROM orders o
45     WHERE o.total_amount > (
46         SELECT AVG(total_amount)
47         FROM orders
48     )
49 );

```

The Result Grid shows the following data:

customer_id	customer_name
4	Customer D
5	Customer E

The Output pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
49	16:06:19	CREATE DATABASE IF NOT EXISTS sales_db	1 row(s) affected, 1 warning(s): 1007 Can't create database 'sales_db': database exists	0.000 sec
50	16:06:19	USE sales_db	0 row(s) affected	0.000 sec
51	16:06:19	CREATE TABLE IF NOT EXISTS customers ( customer_id INT AUTO_INCREMENT PRIMARY KEY, cu...	0 row(s) affected, 1 warning(s): 1050 Table 'customers' already exists	0.000 sec
52	16:06:19	CREATE TABLE IF NOT EXISTS orders ( order_id INT AUTO_INCREMENT PRIMARY KEY, customer_j...	0 row(s) affected, 1 warning(s): 1050 Table 'orders' already exists	0.000 sec
53	16:06:19	INSERT INTO customers (customer_name, region) VALUES (Customer A', 'East'), (Customer B', 'West'), (Cust...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.000 sec
54	16:06:19	INSERT INTO orders (customer_id, order_date, total_amount) VALUES (1, '2024-05-01', 100.00), (2, '2024-05...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.000 sec
55	16:06:19	SELECT c.customer_id, c.customer_name, o.order_id, o.order_date, o.total_amount FROM customers c LEFT ...	7 row(s) returned	0.000 sec / 0.000 sec
56	16:10:55	SELECT customer_id, customer_name FROM customers WHERE customer_id IN ( SELECT o.customer_id ...	2 row(s) returned	0.016 sec / 0.000 sec

**Final Output:**

The screenshot shows the MySQL Workbench interface. The query editor contains the following SQL code:

```

50
51 SELECT customer_name FROM customers
52 UNION
53 SELECT customer_id FROM orders;
54

```

The Result Grid shows the following data:

customer_name
Customer A
Customer B
Customer C
Customer D
Customer E
1
2
4
5

The Output pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
50	16:06:19	USE sales_db	0 row(s) affected	0.000 sec
51	16:06:19	CREATE TABLE IF NOT EXISTS customers ( customer_id INT AUTO_INCREMENT PRIMARY KEY, cu...	0 row(s) affected, 1 warning(s): 1050 Table 'customers' already exists	0.000 sec
52	16:06:19	CREATE TABLE IF NOT EXISTS orders ( order_id INT AUTO_INCREMENT PRIMARY KEY, customer_j...	0 row(s) affected, 1 warning(s): 1050 Table 'orders' already exists	0.000 sec
53	16:06:19	INSERT INTO customers (customer_name, region) VALUES (Customer A', 'East'), (Customer B', 'West'), (Cust...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.000 sec
54	16:06:19	INSERT INTO orders (customer_id, order_date, total_amount) VALUES (1, '2024-05-01', 100.00), (2, '2024-05...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.000 sec
55	16:06:19	SELECT c.customer_id, c.customer_name, o.order_id, o.order_date, o.total_amount FROM customers c LEFT ...	7 row(s) returned	0.000 sec / 0.000 sec
56	16:10:55	SELECT customer_id, customer_name FROM customers WHERE customer_id IN ( SELECT o.customer_id ...	2 row(s) returned	0.016 sec / 0.000 sec
57	16:11:54	SELECT customer_name FROM customers UNION SELECT customer_id FROM orders	9 row(s) returned	0.000 sec / 0.000 sec

**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.**

### **Report on the Use of Transaction Logs for Data Recovery**

*Introduction:* Transaction logs are an essential component of database management systems that help ensure data integrity and facilitate recovery in case of system failures or unexpected shutdowns. These logs record all transactions made to a database, providing a detailed history of changes to the data.

*Functionality of Transaction Logs:* Transaction logs serve several critical functions in data management:

1. **Record of Changes:** Transaction logs record every change made to the database, including inserts, updates, and deletions.
2. **Recovery Point:** They provide a recovery point in case of system failures or crashes, allowing the database to be restored to a consistent state prior to the failure.
3. **Rollback Capability:** Transaction logs enable the rollback of incomplete transactions, ensuring that partially completed changes do not affect data integrity.
4. **Redo Operations:** In addition to rollback, transaction logs facilitate redo operations, replaying committed transactions to restore the database to its latest consistent state.

*Hypothetical Scenario:* Let's consider a hypothetical scenario where a company's e-commerce database experiences an unexpected shutdown during a peak sales period. Several transactions were in progress at the time of the shutdown, leaving the database in an inconsistent state.

- *Scenario Description:*
  - The e-commerce platform experiences a sudden power outage or server failure during the peak holiday sales period.
  - Multiple customers were in the process of placing orders, updating their cart contents, and making payments.
  - Some orders were successfully processed and recorded in the database, while others were left incomplete due to the abrupt shutdown.
  - The shutdown resulted in potential data loss and inconsistencies in the database.
- *Utilization of Transaction Logs:*
  - The transaction logs captured all transactions executed on the database before the shutdown.
  - Upon system recovery, the database administrator initiates a recovery process using the transaction logs.
  - The logs allow the system to identify incomplete transactions and rollback any changes that were not committed before the shutdown.

- Redo operations replay committed transactions to restore the database to its latest consistent state.
- Orders that were successfully processed before the shutdown remain intact, while incomplete or partially processed orders are reverted to their previous state.
- The recovery process ensures that no data is lost, and the database returns to a consistent state, allowing the e-commerce platform to resume operations seamlessly.

*Conclusion:* Transaction logs play a crucial role in data recovery by providing a detailed record of database transactions. In the event of system failures or unexpected shutdowns, transaction logs enable the restoration of databases to a consistent state, minimizing data loss and ensuring data integrity. As demonstrated in the hypothetical scenario, the utilization of transaction logs is instrumental in recovering from disruptions and maintaining the reliability of database systems.