

ASSIGNMENT 1-6

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.

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.

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.

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.

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.

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.

ASSIGNMENT-1

The screenshot shows a MySQL IDE window with multiple tabs. The active tab is 'SQL File 7'. The SQL editor contains the following queries:

```
1 • show tables;
2
3 • select * from orders;
4 • select * from products;
5
6 • alter table customers add (city varchar(29) default "bengaluru");
7 • update customers set city="kolkata" where customer_id in (1,5);
8 • update customers set city="mumbai" where customer_id in (2,4);
9 • select * from customers;
```

Below the editor is the 'Result Grid' showing the output of the last query. It displays a table with 5 columns: customer_id, customer_name, customer_email, and city. The data is as follows:

customer_id	customer_name	customer_email	city
1	asha	asha@gmail.com	kolkata
2	malar	malar@gmail.com	mumbai
3	manish	manish@gmail.com	bengaluru
4	divya	divya@gmail.com	mumbai
5	girish	girish@gmail.com	kolkata
6	deepak	deepak@gmail.com	bengaluru
NULL	NULL	NULL	NULL

The screenshot shows the same MySQL IDE window, but now the 'SQL File 7' tab is active and the SQL editor contains the following queries:

```
2
3 • select * from orders;
4 • select * from products;
5
6 • alter table customers add (city varchar(29) default "bengaluru");
7 • update customers set city="kolkata" where customer_id in (1,5);
8 • update customers set city="mumbai" where customer_id in (2,4);
9 • select * from customers;
10 • select customer_name, customer_email from customers where city="bengaluru";
```

The 'Result Grid' shows the output of the last query, displaying a table with 2 columns: customer_name and customer_email. The data is as follows:

customer_name	customer_email
manish	manish@gmail.com
deepak	deepak@gmail.com

ASSIGNMENT-2

```
4 • select * from products;
5
6 • alter table customers add (city varchar(29) default "bengaluru");
7 • update customers set city="kolkata" where customer_id in (1,5);
8 • update customers set city="mumbai" where customer_id in (2,4);
9 • select * from customers;
10 • select customer_name, customer_email from customers where city="bengaluru";
11
12 • select * from customers c inner join orders o on c.customer_id = o.customer_id where city="bengaluru";
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

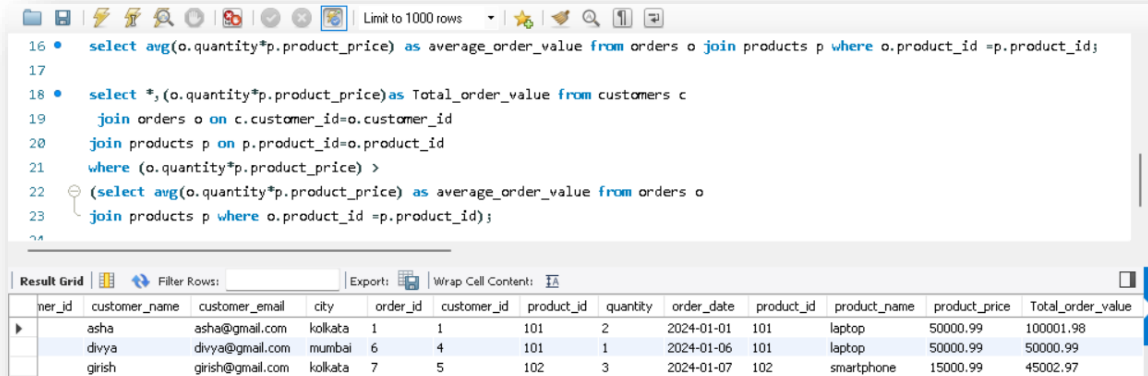
	customer_id	customer_name	customer_email	city	order_id	customer_id	product_id	quantity	order_date
▶	3	manish	manish@gmail.com	bengaluru	3	3	103	3	2024-01-03

```
6 • alter table customers add (city varchar(29) default "bengaluru");
7 • update customers set city="kolkata" where customer_id in (1,5);
8 • update customers set city="mumbai" where customer_id in (2,4);
9 • select * from customers;
10 • select customer_name, customer_email from customers where city="bengaluru";
11
12 • select * from customers c inner join orders o on c.customer_id = o.customer_id where city="bengaluru";
13
14 • select distinct c.customer_name from customers c left outer join orders o on c.customer_id = o.customer_id ;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

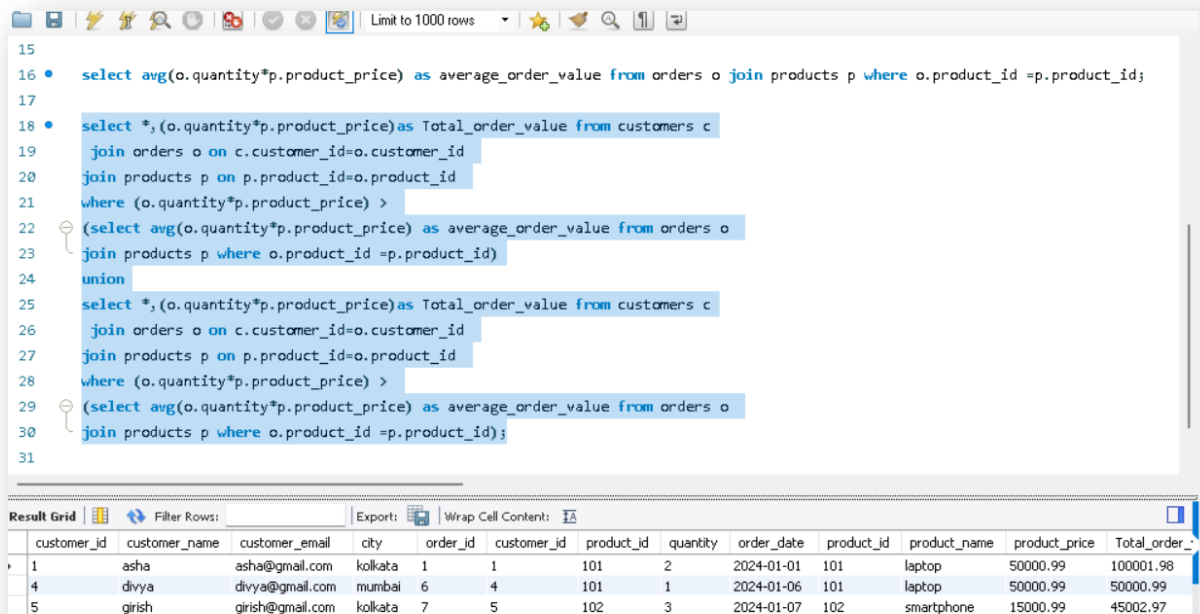
	customer_name
▶	asha
	malar
	manish
	divya
	girish
	deepak

ASSIGNMENT-3



```
16 • select avg(o.quantity*p.product_price) as average_order_value from orders o join products p where o.product_id =p.product_id;
17
18 • select *,(o.quantity*p.product_price)as Total_order_value from customers c
19   join orders o on c.customer_id=o.customer_id
20   join products p on p.product_id=o.product_id
21   where (o.quantity*p.product_price) >
22   (select avg(o.quantity*p.product_price) as average_order_value from orders o
23     join products p where o.product_id =p.product_id);
```

her_id	customer_name	customer_email	city	order_id	customer_id	product_id	quantity	order_date	product_id	product_name	product_price	Total_order_value
	asha	asha@gmail.com	kolkata	1	1	101	2	2024-01-01	101	laptop	50000.99	100001.98
	divya	divya@gmail.com	mumbai	6	4	101	1	2024-01-06	101	laptop	50000.99	50000.99
	girish	girish@gmail.com	kolkata	7	5	102	3	2024-01-07	102	smartphone	15000.99	45002.97



```
15
16 • select avg(o.quantity*p.product_price) as average_order_value from orders o join products p where o.product_id =p.product_id;
17
18 • select *,(o.quantity*p.product_price)as Total_order_value from customers c
19   join orders o on c.customer_id=o.customer_id
20   join products p on p.product_id=o.product_id
21   where (o.quantity*p.product_price) >
22   (select avg(o.quantity*p.product_price) as average_order_value from orders o
23     join products p where o.product_id =p.product_id)
24   union
25   select *,(o.quantity*p.product_price)as Total_order_value from customers c
26   join orders o on c.customer_id=o.customer_id
27   join products p on p.product_id=o.product_id
28   where (o.quantity*p.product_price) >
29   (select avg(o.quantity*p.product_price) as average_order_value from orders o
30     join products p where o.product_id =p.product_id);
31
```

customer_id	customer_name	customer_email	city	order_id	customer_id	product_id	quantity	order_date	product_id	product_name	product_price	Total_order_value
1	asha	asha@gmail.com	kolkata	1	1	101	2	2024-01-01	101	laptop	50000.99	100001.98
4	divya	divya@gmail.com	mumbai	6	4	101	1	2024-01-06	101	laptop	50000.99	50000.99
5	girish	girish@gmail.com	kolkata	7	5	102	3	2024-01-07	102	smartphone	15000.99	45002.97

ASSIGNMENT-4

```
32
33 • start transaction;
34 • insert into orders values(8,3,103,5,date('2024-11-5'));
35
36 • commit;
37
38
39
40
```

order_id	customer_id	product_id	quantity	order_date
5	2	105	2	2024-01-05
6	4	101	1	2024-01-06
7	5	102	3	2024-01-07
8	3	103	5	2024-11-05
NULL	NULL	NULL	NULL	NULL

orders 15 x

Output

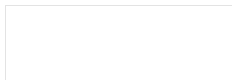
#	Time	Action	Message
32	15:06:54	start transaction	0 row(s) affected
33	15:07:05	insert into orders values(8,3,103,5,date('2024-11-5'))	1 row(s) affected
34	15:07:21	select * from orders LIMIT 0, 1000	8 row(s) returned
35	15:09:01	commit	0 row(s) affected

```
37 • SET SQL_SAFE_UPDATES=0;
38 • start transaction;
39 • update products set product_price=0;
40 • select * from products;
41
42 • rollback;
43
44
45
46
```

product_id	product_name	product_price
101	laptop	50000.99
102	smartphone	15000.99
103	headphones	1000.99
104	tablet	12000.99
105	wireless mouse	800.99

products 17 x

Output



ASSIGNMENT-5

```
44
45 • start transaction;
46 • insert into orders values(9,1,102,7,date('2024-1-9'));
47 • savepoint A;
48 • insert into orders values(10,1,101,6,date('2024-4-13'));
49 • savepoint B;
50 • insert into orders values(11,3,103,9,date('2024-8-11'));
51 • savepoint C;
52 • insert into orders values(12,2,101,2,date('2024-12-30'));
53 • savepoint D;
54 • rollback to C;
55 • select * from orders;
56
57 • commit;
```

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

order_id	customer_id	product_id	quantity	order_date
8	3	103	5	2024-11-05
9	1	102	7	2024-01-09
10	1	101	6	2024-04-13
11	3	103	9	2024-08-11
NULL	NULL	NULL	NULL	NULL

orders 20 x

Apply

Revert

Output

Action Output

#	Time	Action	Message
55	15:33:12	select * from orders LIMIT 0, 1000	11 row(s) returned
56	15:33:35	commit	0 row(s) affected
57	15:33:43	select * from orders LIMIT 0, 1000	11 row(s) returned

ASSIGNMENT-6

Report on the Use of Transaction Logs for Data Recovery

Introduction: Transaction logs are essential components of database management systems that record all changes made to a database. They serve as a crucial tool for data recovery in case of unexpected shutdowns or system failures. This report explores the significance of transaction logs in data recovery and presents a hypothetical scenario to illustrate their effectiveness.

Significance of Transaction Logs:

1. **Record of Changes:** Transaction logs meticulously track every modification, insertion, or deletion made to the database, ensuring a detailed history of data transactions.
2. **Point-in-Time Recovery:** By replaying transactions recorded in the log, databases can be restored to a specific point in time, minimizing data loss and maintaining data integrity.
3. **Undo and Redo Operations:** Transaction logs facilitate both undo and redo operations, enabling recovery from both user errors and system failures.
4. **High Availability:** Transaction logs contribute to the high availability of databases by providing a reliable mechanism for restoring data in the event of failures.

Hypothetical Scenario: Consider a retail company with an extensive customer database. One day, their database server experiences an unexpected shutdown due to a power outage. Upon rebooting, it's discovered that a significant portion of the customer data is missing or corrupted.

Utilizing Transaction Logs for Recovery:

1. **Identifying the Point of Failure:** The database administrators analyze the transaction logs to pinpoint the last successful transaction before the shutdown occurred.
2. **Replaying Transactions:** Using the information from the transaction log, the administrators replay the transactions leading up to the point of failure, restoring the database to its state just before the outage.
3. **Data Validation:** After the recovery process, data validation procedures are conducted to ensure the integrity and accuracy of the restored data.
4. **Resuming Operations:** With the database successfully restored, the retail company can resume normal operations, minimizing disruption to their business activities.

Conclusion: Transaction logs play a crucial role in data recovery by providing a detailed record of database transactions. In the event of unexpected shutdowns or system failures, transaction logs enable organizations to restore data to a consistent state, ensuring data integrity and minimizing downtime. Implementing robust transaction logging mechanisms is essential for maintaining the reliability and availability of databases in today's digital landscape.