

**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.**

Answer:

**SELECT \* FROM customers;**

	custid	cname	city	puramt	dob	emailid
▶	11	Akash	Bangalore	250.00	1994-05-23	Akash19940523@example.com
	12	Ravi	Bangalore	NULL	1985-05-22	Ravi19850522@example.com
	13	Bala	Bangalore	650.00	1992-06-21	Bala19920621@example.com
	14	Cathy	Pune	350.00	1999-12-02	Cathy19991202@example.com
	15	David	Chennai	450.00	2001-01-14	David20010114@example.com
	16	Elsa	Chennai	700.00	1987-07-05	Elsa19870705@example.com
	17	Fathima	Bangalore	950.00	1983-09-23	Fathima19830923@example.com
	18	Ganesh	Bangalore	800.00	1977-10-23	Ganesh19771023@example.com

**SELECT cname, emailid FROM customers WHERE city = 'Bangalore';**

29

30 • **SELECT cname, emailid FROM customers WHERE city = 'Bangalore';**

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
cname	emailid		
▶ Akash	Akash19940523@example.com		
Ravi	Ravi19850522@example.com		
Bala	Bala19920621@example.com		
Fathima	Fathima19830923@example.com		
Ganesh	Ganesh19771023@example.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.**

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

In this query

INNER JOIN orders o ON c.customer\_id = o.customer\_id joins the customers table (c) with the orders table (o) on the customer\_id.

WHERE c.region = 'North' filters the results to include only customers from the 'North' region.

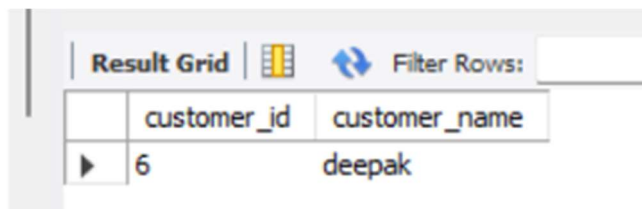
**display all customers including those without orders.**

```
SELECT c.customer_id, c.customer_name
```

```
FROM customers c
```

```
LEFT JOIN orders o ON c.customer_id = o.customer_id
```

```
WHERE o.order_id IS NULL;
```



The screenshot shows a database interface with a 'Result Grid' tab. It contains a table with two columns: 'customer\_id' and 'customer\_name'. The first row has the values '6' and 'deepak' respectively. There is a 'Filter Rows:' button with a funnel icon to the right of the column headers.

customer_id	customer_name
6	deepak

**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.**

**Subquery to Find Customers Who Have Placed Orders Above the Average Order Value**

```
SELECT DISTINCT c.customer_id, c.customer_name
```


```
FROM customers c
```

```
INNER JOIN orders o ON c.customer_id = o.customer_id
```

```
WHERE o.quantity > (
```

```
    SELECT AVG(quantity) FROM orders
```

```
);
```

Result Grid    Filter Rows: <input type="text"/>		
	customer_id	customer_name
▶	1	asha
	3	manish
	2	malar
	5	girish

#### **UNION query to combine customers:**

```
SELECT customer_id, customer_name, region
FROM customers
WHERE region = 'North'
UNION
SELECT customer_id, customer_name, region
FROM customers
WHERE region = 'South';
```

**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.**

Answer:

-- BEGIN a transaction

```
BEGIN TRANSACTION;
```

-- INSERT a new record into the 'orders' table

```
INSERT INTO orders (order_id, customer_id, product_id, quantity, order_date)
VALUES (1, 123, 456, 3, '2024-05-21');
```

-- COMMIT the transaction

```
COMMIT;
```

-- UPDATE the 'products' table

```
UPDATE products
```

```
SET price = price * 1.1
```

WHERE product\_id = 456;

-- ROLLBACK the transaction

ROLLBACK;

**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.**

Answer:

-- BEGIN a transaction

BEGIN TRANSACTION;

-- INSERT a new record into the 'orders' table and set a SAVEPOINT

SAVEPOINT savepoint1;

INSERT INTO orders (order\_id, customer\_id, product\_id, quantity, order\_date)

VALUES (1, 123, 456, 3, '2024-05-21');

-- INSERT another record into the 'orders' table and set another SAVEPOINT

SAVEPOINT savepoint2;

INSERT INTO orders (order\_id, customer\_id, product\_id, quantity, order\_date)

VALUES (2, 456, 789, 5, '2024-05-22');

-- Rollback to the second SAVEPOINT

ROLLBACK TO SAVEPOINT savepoint2;

-- COMMIT the overall transaction

COMMIT;

**BEGIN TRANSACTION:** This statement starts a new transaction. Transactions are used to group multiple SQL statements into a single unit of work that either succeeds or fails as a whole.

**SAVEPOINT savepoint1:** This statement sets a savepoint within the transaction. Savepoints allow you to mark a point within a transaction to which you can later rollback if needed.

**INSERT INTO orders ...:** These statements perform INSERT operations into the orders table. Each INSERT adds a new record to the table.

**SAVEPOINT savepoint2:** This statement sets another savepoint within the transaction, after the second INSERT operation.

ROLLBACK TO SAVEPOINT savepoint2: This statement rolls back the transaction to the state after the second INSERT operation. This means that any changes made after the second savepoint will be undone.

COMMIT: This statement commits the overall transaction. It makes all changes within the transaction permanent. However, in this script, since there was a rollback before the commit, only the changes up to the rollback point are committed.

**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.**

Transaction logs are crucial for data recovery in the event of an unexpected shutdown or system failure. They record all the transactions that have occurred in a database, allowing for the recovery of data in case of any issues.

Transaction logs are a critical component of database management systems (DBMS) that ensure data integrity and facilitate data recovery in the event of system failures or unexpected shutdowns. These logs record all modifications made to the database, providing a detailed history of transactions. In this report, we will explore the importance of transaction logs for data recovery and illustrate their significance through a hypothetical scenario.

**Importance of Transaction Logs for Data Recovery:**

- **Redundant Record of Changes:** Transaction logs serve as a redundant record of all changes made to the database. They capture every insert, update, and delete operation, along with the corresponding metadata such as timestamps and transaction IDs.
- **Point-in-Time Recovery:** Transaction logs enable point-in-time recovery, allowing DBAs to restore the database to a specific moment before the occurrence of a failure or corruption. By replaying transactions from the logs, it's possible to recreate the database state up to the desired point in time.
- **Rollback and Roll forward Operations:** Transaction logs support rollback and rollforward operations. Rollback reverses incomplete transactions, restoring the database to its state before the transaction began. Rollforward applies committed transactions that were not yet reflected in the database due to a system failure.
- **Minimized Data Loss:** With transaction logs, data loss is minimized in the event of a system crash or unexpected shutdown. Even if the most recent changes were not yet written to disk, they can be recovered by replaying transactions from the logs.

In a hypothetical scenario, let's say that a database server unexpectedly shuts down due to a power outage. When the server is restarted, the transaction logs can be used to recover any data that was not yet committed to the database before the shutdown. This is done by replaying the transactions recorded in the logs, which will bring the database back to the state it was in before the shutdown.

1. The database server unexpectedly shuts down due to a power outage.
2. When the server is restarted, the transaction logs are scanned to identify any transactions that were not yet committed to the database before the shutdown.
3. These transactions are then replayed, bringing the database back to the state it was in before the shutdown.
4. Any data that was lost during the shutdown is recovered, and the database is now up-to-date and consistent.

-- Restore the database from a backup

```
RESTORE DATABASE mydatabase FROM DISK = 'C:\backups\mydatabase.bak';
```

-- Recover the database using the transaction logs

```
RESTORE LOG mydatabase FROM DISK = 'C:\backups\mydatabase_log.trn' WITH  
RECOVERY;
```

In this example, the RESTORE DATABASE command is used to restore the database from a backup, and the RESTORE LOG command is used to recover the database using the transaction logs. The WITH RECOVERY option tells SQL Server to bring the database online and make it available for use after the recovery process is complete.

Database Engine to run the **recovery process** for the database to be brought online.

Recovery is the process used by SQL Server for each database to start in a transactionally consistent - or clean - state.

In case of a failover or other non-clean shut down, the databases may be left in a state where some modifications were never written from the buffer cache to the data files, and there may be some modifications from incomplete transactions in the data files. When an

instance of SQL Server is started, it runs a recovery of each database, which consists of three phases, based on the last database checkpoint:

- **Phase 1** is the **Analysis Phase** that analyses the transaction log to determine what is the last checkpoint, and creates the *Dirty Page Table* (DPT) and the *Active Transaction Table* (ATT). The DPT contains records of pages that were dirty at the time the database was shut down. The ATT contains records of transactions that were active at the time the database wasn't cleanly shut down.
- **Phase 2** is the **Redo Phase** that rolls forwards every modification recorded in the log that may not have been written to the data files at the time the database was shut down. The minimum log sequence number (minLSN) required for a successful database-wide recovery is found in the DPT, and marks the start of the redo operations needed on all dirty pages. At this phase, the SQL Server Database Engine writes to disk all dirty pages belonging to committed transactions.
- **Phase 3** is the **Undo Phase** that rolls back incomplete transactions found in the ATT to make sure the integrity of the database is preserved. After rollback, the database goes online, and no more transaction log backups can be applied to the database.