

SQL AND RDBMS ASSIGNMENTS

Assignment1: Analyse a given business scenario and create an ER diagram that includes entities, relationships, attributes, and cardinality. Ensure that the diagram reflects proper normalization up to the third normal form.

Entities:

1. Patient
2. Doctor
3. Test

Relationships:

1. Patient visits Doctor
2. Doctor treats Patient
3. Doctor works in Department
4. Patient has Appointment
5. Patient undergoes Test

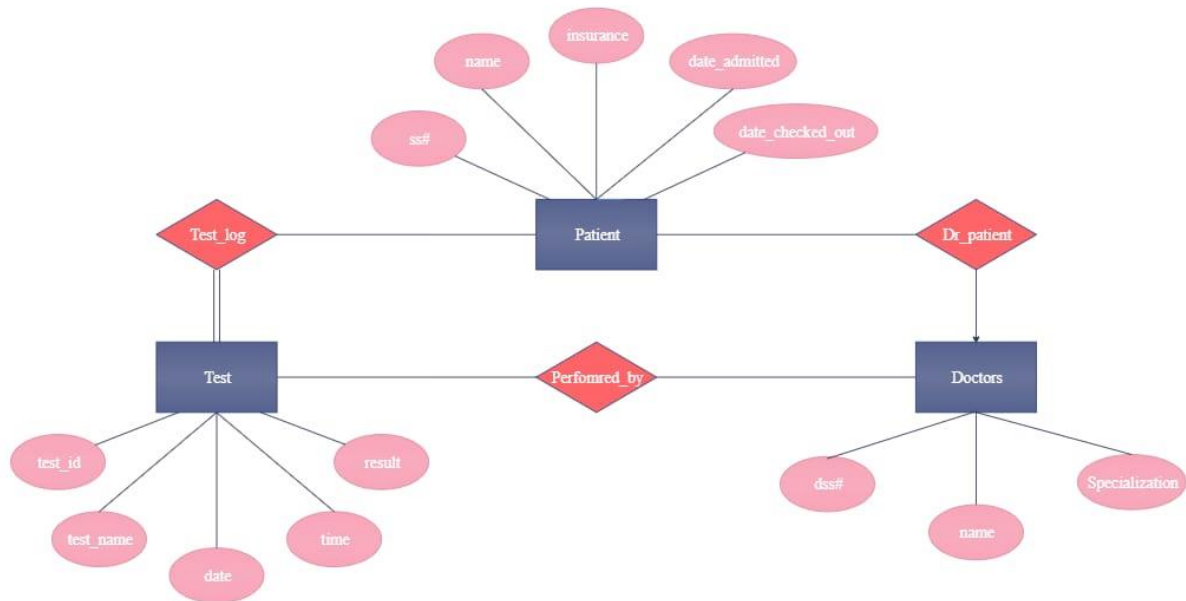
Attributes:

1. Patient: SS#, name, insurance, date_admitted, date_checked_out
2. Doctor: DSS, Name, Specialization
4. Test: test_ID, test_name, Date, time, result

Cardinality:

1. One patient can have multiple appointments.
2. One doctor can have multiple appointments, treat multiple patients, and perform multiple tests.
3. One nurse can assist multiple doctors
4. One department can have multiple doctors.

ER diagram of Hospital



=====

Assignment2: Design a database schema for a library system, including tables, fields, and constraints like NOT NULL, UNIQUE, and CHECK. Include primary and foreign keys to establish relationships between tables.

```
CREATE TABLE Authors (  
    author_id INT PRIMARY KEY,  
    name VARCHAR(100) NOT NULL,  
    nationality VARCHAR(50) NOT NULL  
);  
  
CREATE TABLE Books (  
    book_id INT PRIMARY KEY,  
    title VARCHAR(200) NOT NULL,  
    author_id INT NOT NULL,  
    genre VARCHAR(50),  
    publication_year INT,  
    FOREIGN KEY (author_id) REFERENCES Authors(author_id)  
);  
  
CREATE TABLE Members (  
    member_id INT PRIMARY KEY,  
    name VARCHAR(100) NOT NULL,  
    address VARCHAR(200) NOT NULL,  
    phone VARCHAR(20) NOT NULL,  
    email VARCHAR(50) NOT NULL,  
    FOREIGN KEY (member_id) REFERENCES Members(member_id)
```

```

member_id INT PRIMARY KEY,

name VARCHAR (100) NOT NULL,

email VARCHAR (100) NOT NULL,

address VARCHAR (200) NOT NULL

);

CREATE TABLE Transactions (

transaction_id INT PRIMARY KEY,

book_id INT NOT NULL,

member_id INT NOT NULL,

checkout_date DATE NOT NULL,



return_date DATE NOT NULL,



FOREIGN KEY (book_id) REFERENCES Books(book_id),

FOREIGN KEY (member_id) REFERENCES Members(member_id),

CHECK (checkout_date <= return_date);

```

Result Grid						
Filter Rows: <input type="text"/>						
Export:  Wrap Cell Content: 						
	Field	Type	Null	Key	Default	Extra
▶	member_id	int	NO	PRI	NULL	
	name	varchar(150)	NO		NULL	
	email	varchar(200)	NO		NULL	
	address	varchar(200)	NO		NULL	

Result Grid						
Filter Rows: <input type="text"/>						
Export:  Wrap Cell Content: 						
	Field	Type	Null	Key	Default	Extra
▶	transcation_id	int	NO	PRI	NULL	
	book_id	int	NO	MUL	NULL	
	member_id	int	NO	MUL	NULL	
	checkout_date	date	NO		NULL	
	return_date	date	NO		NULL	

Result Grid						
	Field	Type	Null	Key	Default	Extra
▶	book_id	int	NO	PRI	NULL	
	title	varchar(200)	NO		NULL	
	author	varchar(200)	NO		NULL	
	genre	varchar(150)	YES		NULL	
	publication_year	int	YES		NULL	

=====

Assignment3: Explain the ACID properties of a transaction in your own words. Write SQL statements to simulate a transaction that includes locking and demonstrate different isolation levels to show concurrency control.

Acid Properties:

- 1. Atomicity:** This property ensures that either all the operations within a transaction are successfully completed, or none of them are. If any part of the transaction fails, the entire transaction is rolled back to its original state.
- 2. Consistency:** Consistency ensures that the database remains in a valid state before and after the transaction. All integrity constraints, such as foreign key constraints or uniqueness constraints, must be satisfied.
- 3. Isolation:** Isolation ensures that the concurrent execution of transactions results in a state that could be obtained if transactions were executed serially. Isolation levels define the degree to which the operations within one transaction are isolated from the operations of other concurrent transactions.
- 4. Durability:** Durability guarantees that once a transaction has been committed, the changes made by it will persist even in the event of system failure.

```
CREATE TABLE bank_accounts (
    account_id INT PRIMARY KEY,
    balance DECIMAL(10, 2)
);

INSERT INTO bank_accounts (account_id, balance) VALUES
(1, 2000.00),
(2, 4000.00);

BEGIN TRANSACTION;

-- Withdrawal operation

UPDATE bank_accounts
```

```

SET balance = balance - 500.00

WHERE account_id = 1;

UPDATE bank_accounts

SET balance = balance + 500.00

WHERE account_id = 2;

Commit;

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

SELECT balance FROM bank_accounts WHERE account_id = 1;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

SELECT balance FROM bank_accounts WHERE account_id = 1;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

SELECT balance FROM bank_accounts WHERE account_id = 1;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SELECT balance FROM bank_accounts WHERE account_id = 1;

```

=====

Assignment4: Write SQL statements to CREATE a new database and tables that reflect the library schema you designed earlier. Use ALTER statements to modify the table structures and DROP statements to remove a redundant table.

```

create database library;

use library;

create table books (book_id int primary key, title varchar (50) not null, author varchar (50),
publication_year year, ISBN int unique);

alter table books add column copies_available int;

alter table books add column author_id int;

alter table books drop column author;

desc books;

```

output

Field	Type	Null	Key
book_id	int	NO	PRI

title	varchar (50	NO	
publication_year	year	YES	
ISBN	bigint	YES	UNI
author_id	int	YES	

```
create table authors (author_id int primary key, author_name varchar (50));
```

```
desc authors;
```

output

```
/*
```

Field	Type	Null	Key
author_id	int	NO	PRI
author name	varchar (50)	YES	

```
*/
```

```
create table borrowings (borrowing_id int primary key, book_id int, member_id int, borrow_date date,
return_date date, status varchar (50));
```

```
desc borrowings;
```

output

```
/*
```

Field	Type	Null	Key
Borrowing	int	NO	PRI
book_id	int	YES	
member_id	int	YES	
borrow_date	date	YES	
return_date	date	YES	
status	varchar (50)	YES	

```
*/
```

```
create table members (member_id int primary key, member_name varchar (50), email varchar (100),
phone_number varchar(20), address varchar(50));
```

```
desc members;
```

output

/*

Field	Type	Null	Key
member_id	int	NO	PRI
member_name	varchar(50)	YES	
email	varchar(100)	YES	
phone_number	varchar(20)	YES	
address	varchar(50)	YES	

*/

alter table borrowings modify status varchar(100);

create table books(book_title varchar (20), book_price float);

drop table books;

=====

Assignment5: Demonstrate the creation of an index on a table and discuss how it improves query performance. Use a DROP INDEX statement to remove the index and analyse the impact on query execution.

```
USE library;

CREATE TABLE authors(
    authorID INT AUTO_INCREMENT PRIMARY KEY,
    firstName VARCHAR(100) NOT NULL,
    lastNmae VARCHAR(100) NOT NULL
);

DESC authors;

ALTER TABLE authors RENAME COLUMN lastNmae TO lastName;

INSERT INTO authors (firstName, lastName) VALUES
("Sai", "Chandana"),
("sanvi", "sher");

SELECT * FROM authors;

CREATE TABLE books(
```

```

    bookID INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    authorID INT NOT NULL,
    FOREIGN KEY(authorID) REFERENCES authors(authorID)
);

ALTER TABLE books DROP COLUMN bookAvailable;

DESCRIBE books;

INSERT INTO books (title, authorID) VALUES
("title1", 1),
("title2", 2),
("title3", 1),
("title4", 2);

SHOW INDEX FROM books;

CREATE INDEX ID_title ON books (title);
DROP INDEX ID_title ON books;

SELECT * FROM books;
SELECT * FROM books WHERE title = "title2";

```

When we use an index in a table, it helps in fast retrieval, especially when the table has a large number of rows. Without an index, the database has to check each row individually. Indexes enhance performance and reduce the time needed to execute queries. Dropping an index removes this benefit, causing queries to take longer as the database must perform full table scans.

=====

Assignment6: Create a new database user with specific privileges using the CREATE USER and GRANT commands. Then, write a script to REVOKE certain privileges and DROP the user.

```

-- Step 1: Create the database if it does not exist
CREATE DATABASE IF NOT EXISTS newdatabase;

-- Step 2: Use the new database
USE newdatabase;

-- Step 3: Create a new user

```



```

CREATE USER 'sai_chandana'@'%' IDENTIFIED BY 'preethi';

-- Step 4: Grant all privileges on the new database to the new user
GRANT ALL PRIVILEGES ON newdatabase.* TO 'sai_chandana'@'%';

-- Step 5: Revoke the INSERT privilege from the new user
REVOKE INSERT ON newdatabase.* FROM 'sai_chandana'@'%';

-- Step 6: Drop the user
DROP USER 'sai_chandana'@'%';

```

=====

Assignment7: Prepare a series of SQL statements to INSERT new records into the library tables, UPDATE existing records with new information, and DELETE records based on specific criteria. Include BULK INSERT operations to load data from an external source.

```

-- Create the database if it does not exist
CREATE DATABASE IF NOT EXISTS library;

-- Use the library database
USE library;

-- Create the members table
CREATE TABLE members (
    memberID INT AUTO_INCREMENT PRIMARY KEY,
    firstName VARCHAR(50),
    lastName VARCHAR(20),
    email VARCHAR(100),
    phone VARCHAR(15), -- Consider using VARCHAR if phone numbers may contain
characters
    address VARCHAR(200)
);

-- Insert sample data into the members table
INSERT INTO members (firstName, lastName, email, phone, address) VALUES
("Sai", "Chandana", "Saichandana@gmail.com", "9182289543", "Hyderabad"),
("Yuva", "Raj", "yuvaraj@gmail.com", "8788778789", "Vizag"),
("Priya", "Krishnan", "priyakrishnan@gmail.com", "7998889898", "Delhi");

-- Update email for member "Priya"
UPDATE members SET email = "priyakrishnan@gmail.com" WHERE firstName = "Priya";

```

```

-- Delete member "Priya"
DELETE FROM members WHERE firstName='Priya';

-- Create the authors table
CREATE TABLE authors (
    authorID INT AUTO_INCREMENT PRIMARY KEY,
    firstName VARCHAR(100) NOT NULL,
    lastName VARCHAR(100) NOT NULL
);

-- Insert sample data into the authors table
INSERT INTO authors (firstName, lastName) VALUES
("Chetan", "Bhagat"),
("Chetan", "Bhagat"),
("Balwant", "Gargi");

-- Create the books table
CREATE TABLE books (
    bookID INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    authorID INT NOT NULL,
    FOREIGN KEY(authorID) REFERENCES authors(authorID)
);

-- Insert sample data into the books table
INSERT INTO books (title, authorID) VALUES
("Making India Awesome", 3),
("One Indian Girl", 4),
("Naked Triangle", 5);

-- Delete book "Naked Triangle"
DELETE FROM books WHERE title='Naked Triangle';

-- Create the bookExternal table
CREATE TABLE bookExternal (
    title VARCHAR(300),
    author VARCHAR(300),
    published_year INT -- Consider using VARCHAR if published year may contain
non-numeric characters
);

-- Load data from CSV file into the bookExternal table
LOAD DATA INFILE '/path/to/book2.csv' INTO TABLE bookExternal
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'

```

```
IGNORE 1 LINES; -- Make sure to use forward slashes in the file path and
adjust the file path as necessary
```

```
=====
=====
```

Day 2:

Assignment1: Write a SELECT query to retrieve all columns from a 'customers' table, and modify it to return only the customer's name and email address for customers in a specific city.

```
USE library;

CREATE TABLE customers(
  customerID INT PRIMARY KEY AUTO_INCREMENT,
  customerName VARCHAR(100),
  email VARCHAR(100) NOT NULL,
  city VARCHAR(50)
);

INSERT INTO customers (customerName, email, city) VALUES ("Sai_Chandana",
"saichandana@gmail.com", 'Hyderabad');
INSERT INTO customers (customerName, email, city) VALUES ("preethi",
"preethi@gmail.com", 'Bengaluru');
INSERT INTO customers (customerName, email, city) VALUES ("shiva",
"shiva@gmail.com", 'chennai');

-- Retrieve all columns from the 'customers' table
SELECT * FROM customers;

-- Modify the query to return only the customer's name and email address for
customers in a specific city
SELECT customerName, email FROM customers WHERE city = 'chennai';
```

```
+-----+-----+-----+-----+
| customerID | customerName | email | city |
+-----+-----+-----+-----+
| 1 | Sai_Chandana | saichandana@gmail.com | Hyderabad |
| 2 | preethi | preethi@gmail.com | Bengaluru |
| 3 | shiva | shiva@gmail.com | chennai |
+-----+-----+-----+-----+
```

```

+-----+-----+
| customerName | email |
+-----+-----+
| shiva | shiva@gmail.com |
+-----+-----+

```

Assignment2: 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.

```

-- Create the newdatabase if it does not exist
CREATE DATABASE IF NOT EXISTS newdatabase;

-- Use the newdatabase
USE newdatabase;

-- Create the customers table
CREATE TABLE customers (
  customerID INT PRIMARY KEY AUTO_INCREMENT,
  customerName VARCHAR(100),
  email VARCHAR(100) NOT NULL,
  city VARCHAR(50)
);

-- Insert sample data into the customers table
INSERT INTO customers (customerName, email, city) VALUES ("SaiChandana",
"saichandana@gmail.com", 'Hyderabad');
INSERT INTO customers (customerName, email, city) VALUES ("preethi",
"preethi@gmail.com", 'Bengaluru');
INSERT INTO customers (customerName, email, city) VALUES ("shiva",
"shiva@gmail.com", 'Chennai');

-- Create the orders table
CREATE TABLE orders (
  orderID INT PRIMARY KEY AUTO_INCREMENT,
  customerID INT,
  orderDate DATE,
  totalAmount DECIMAL(10,2),
  FOREIGN KEY (customerID) REFERENCES customers (customerID)
);

-- Insert sample data into the orders table

```

```

INSERT INTO orders (customerID, orderDate, totalAmount) VALUES (1, '2024-05-16', 200.00);
INSERT INTO orders (customerID, orderDate, totalAmount) VALUES (2, '2024-05-10', 1000.56);

-- Query 1: INNER JOIN to combine 'orders' and 'customers' tables for customers in a specified region (Hyderabad)
SELECT customers.customerID, customers.customerName, customers.email, orders.totalAmount
FROM customers
INNER JOIN orders ON customers.customerID = orders.customerID
WHERE customers.city = 'Hyderabad'; -- Filter customers by city

-- Query 2: LEFT JOIN to display all customers including those without orders
SELECT customers.customerID, customers.customerName, customers.email, orders.totalAmount, orders.orderDate
FROM customers
LEFT JOIN orders ON customers.customerID = orders.customerID;

```

For the first query (INNER JOIN to combine 'orders' and 'customers' tables for customers in the city of Hyderabad):

```

+-----+-----+-----+-----+
| customerID | customerName | email | totalAmount |
+-----+-----+-----+-----+
| 1 | SaiChandana | saichandana@gmail.com | 200.00 |
+-----+-----+-----+-----+

```

second query (LEFT JOIN to display all customers including those without orders):

```

+-----+-----+-----+-----+-----+
----+
| customerID | customerName | email | totalAmount |
orderDate |
+-----+-----+-----+-----+-----+
----+
| 1 | SaiChandana | saichandana@gmail.com | 200.00 | 2024-05-16
|
| 2 | preethi | preethi@gmail.com | 1000.56 | 2024-05-10
|
| 3 | shiva | shiva@gmail.com | NULL | NULL
|
+-----+-----+-----+-----+-----+
----+

```

Assignment3: 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.

```
-- Your existing code for creating tables and performing UNION query
USE newdatabase;
```

```
CREATE TABLE customers (
  customerID INT PRIMARY KEY AUTO_INCREMENT,
  customerName VARCHAR(100),
  email VARCHAR(100) NOT NULL,
  city VARCHAR(50)
);
```

```
CREATE TABLE orders (
  orderID INT PRIMARY KEY AUTO_INCREMENT,
  customerID INT,
  orderDate DATE,
  totalAmount DECIMAL(10,2),
  FOREIGN KEY (customerID) REFERENCES customers (customerID)
);
```

```
-- Your CREATE TABLE statements remain unchanged
```

```
WITH AvgOrder AS (
  SELECT AVG(totalAmount) AS avgTotalAmount
  FROM orders
)
SELECT orderID, customerID, orderDate, totalAmount
FROM orders
WHERE totalAmount > (SELECT avgTotalAmount FROM AvgOrder)
UNION
SELECT orderID, customerID, orderDate, totalAmount
FROM orders
WHERE totalAmount <= (SELECT avgTotalAmount FROM AvgOrder);
```

```
-- Insert values into the customers table
INSERT INTO customers (customerName, email, city) VALUES
("Sai_Chandana", "sai@gmail.com", "Hyderabad"),
("preethi", "preethi@gmail.com", "karnataka"), -- Comma added here
("Shiva", "Shiva@gmail.com", "Noida");
```

```
-- Insert values into the orders table
INSERT INTO orders (customerID, orderDate, totalAmount) VALUES
```

```
(1, '2024-05-01', 150.00),
(2, '2024-05-02', 200.50),
(3, '2024-05-03', 300.75);
```

OUTPUT:

```
+-----+-----+-----+-----+
| customerID | customerName | email                | city        |
+-----+-----+-----+-----+
| 1          | Sai_Chandana | sai@gmail.com        | Hyderabad  |
| 2          | preethi      | preethi@gmail.com    | karnataka  |
| 3          | Shiva        | Shiva@gmail.com      | Noida      |
+-----+-----+-----+-----+
```

```
+-----+-----+-----+-----+
| orderID | customerID | orderDate | totalAmount |
+-----+-----+-----+-----+
| 1       | 1          | 2024-05-01 | 150.00      |
| 2       | 2          | 2024-05-02 | 200.50      |
| 3       | 3          | 2024-05-03 | 300.75      |
+-----+-----+-----+-----+
```

```
+-----+-----+-----+-----+
| orderID | customerID | orderDate | totalAmount |
+-----+-----+-----+-----+
| 2       | 2          | 2024-05-02 | 200.50      |
| 3       | 3          | 2024-05-03 | 300.75      |
+-----+-----+-----+-----+
```

Assignment4: 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 the newdatabase
USE newdatabase;

-- Begin a transaction
BEGIN;

-- Create the products table if not exists
CREATE TABLE IF NOT EXISTS products (
    productID INT PRIMARY KEY AUTO_INCREMENT,
    productName VARCHAR(100) NOT NULL,
    quantity INT NOT NULL DEFAULT 0
```

```

);

-- Insert values into the orders table within the transaction
INSERT INTO orders (customerID, orderDate, totalAmount) VALUES
(4, '2024-05-15', 100.00),
(5, '2024-05-11', 900.56);

-- Update the products table within the transaction
UPDATE products SET quantity = quantity + 1 WHERE productID = 1;

-- Commit the transaction
COMMIT;

-- Rollback the transaction (for demonstration purposes)
ROLLBACK;

```

Assignment5: 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.

```

USE newdatabase;

-- Begin a transaction
START TRANSACTION;

-- Create the products table
CREATE TABLE IF NOT EXISTS products (
    productID INT PRIMARY KEY AUTO_INCREMENT,
    productName VARCHAR(100) NOT NULL,
    quantity INT NOT NULL DEFAULT 0
);

-- INSERT INTO customers specifying columns
INSERT INTO customers (customerID, customerName, email, city) VALUES
(6, "bhavani", "bhavani@gmail.com", 'Varanasi'),
(7, "chintu", "chintu@gmail.com", 'Chennai');

-- INSERT INTO orders specifying columns
INSERT INTO orders (customerID, orderDate, totalAmount) VALUES
(6, '2024-05-15', 100.00);
SAVEPOINT save1;

-- INSERT INTO orders specifying columns
INSERT INTO orders (customerID, orderDate, totalAmount) VALUES
(7, '2024-05-11', 1900.56);
SAVEPOINT save2;

```



```
-- Rollback to the second savepoint
ROLLBACK TO SAVEPOINT save2;

-- Commit the transaction
COMMIT;
```

Assignment6: 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.

Introduction:

Transaction logs are crucial components of database management systems that record all changes made to a database. These logs serve as a reliable source of information for recovering data in the event of system failures or unexpected shutdowns.

Importance of Transaction Logs:

- 1. Data Integrity:** Transaction logs ensure data integrity by recording every transaction before it is committed to the database. This allows for rollbacks or recovery to a specific point in time.
- 2. Recovery Point:** They provide a recovery point in case of system failures, allowing databases to be restored to a consistent state prior to the failure.
- 3. Performance Monitoring:** Transaction logs also aid in performance monitoring and troubleshooting, as they track changes and can identify potential issues.

Hypothetical Scenario:

Imagine a scenario where a large e-commerce company experiences an unexpected server shutdown during a peak shopping period, resulting in potential data loss and customer disruption. However, due to the implementation of transaction logs, the company's database administrator can initiate a successful data recovery process.

Scenario Details:

- 1. Unexpected Shutdown:** The e-commerce platform experiences a sudden server shutdown due to a power outage.
- 2. Data Loss Concerns:** Concerns arise about potential data loss, including ongoing transactions and customer orders that were being processed.
- 3. Transaction Logs Utilization:** The database administrator leverages transaction logs to restore the database to its state just before the shutdown.
- 4. Recovery Process:** By analysing the transaction logs, the administrator identifies the last committed transactions before the shutdown.

5. Database Restoration: Using this information, the administrator restores the database to the point just before the unexpected shutdown, ensuring minimal data loss and maintaining data consistency.

6. Customer Impact Mitigation: The quick recovery minimizes disruption for customers, allowing them to resume their transactions seamlessly.

Conclusion:

Transaction logs play a vital role in data recovery, especially in scenarios of unexpected shutdowns or system failures. By maintaining a record of all database transactions, transaction logs enable organizations to restore data integrity and minimize downtime, ultimately ensuring business continuity and customer satisfaction.