**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.

**Sol:**

Entities:

1. Customer
2. Product
3. Order
4. Category
5. Delivery

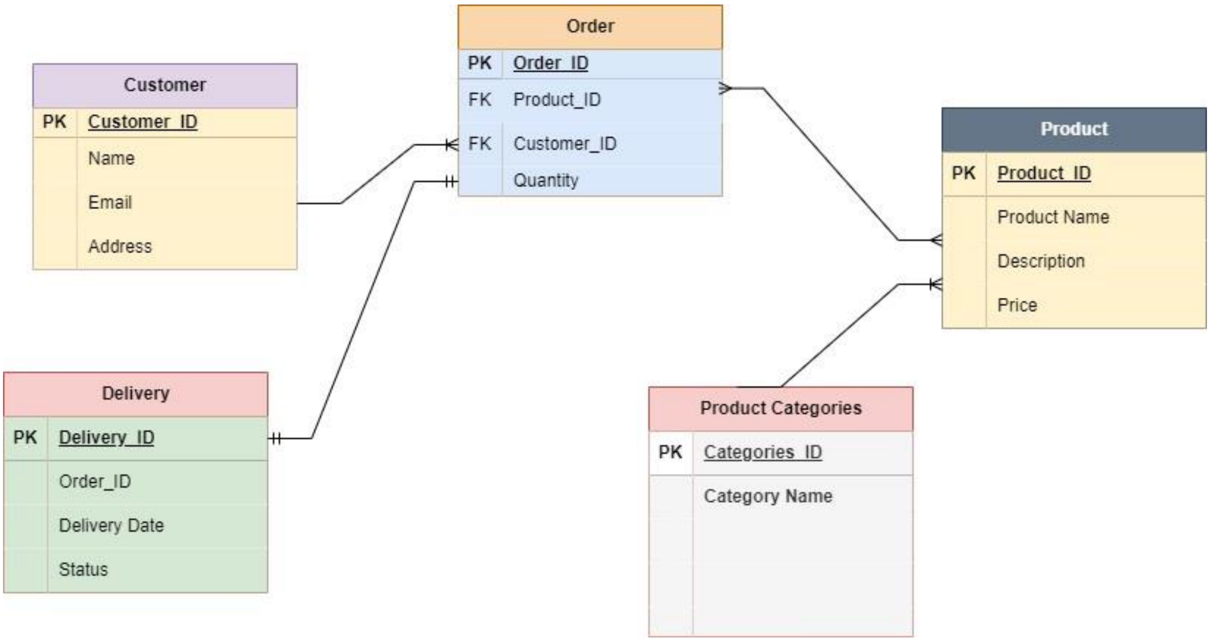
Relationships:

1. Customer-Order: One customer place many orders (One-to-many).
2. Product-Order: One product can be in many orders and one order can have multiple products (Many-to-Many).
3. Categories-Product: One category can have many products. One product belongs to one category (one-to-many)
4. Order-Delivery: One order can have one delivery and one delivery is associated with one order.

Attributes:

1. Customer: Customer_ID, Name, Email, Address.
2. Product: Product_ID, Product Name, Description, Price.
3. Product Categories: Category_ID, Category Name.
4. Order: Order_ID, Product_ID, Customer_ID, Quantity.
5. Delivery: Delivery_ID, Order_ID, Delivery Date, Status.

Entity Relationship Diagram Schema:




**Assignment 2:** Design a database schema for a library system, including tables, fields, and constraints like NOT NULL, UNIQUE, and CHECK. Include primary and foreign keys da to establish relationships between tables

Sol:

CREATE DATABASE LibrarySystem;

USE LibrarySystem;

CREATE TABLE Student (

Student_id INT AUTO_INCREMENT PRIMARY KEY,

Name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

DOB DATE NOT NULL,

City VARCHAR(100)

);

INSERT INTO Student (Name, email, DOB, City)

VALUES

("Rajesh", "rajesh123@gmail.com", "2000-11-21", "Delhi"),

("Radha", "radha1234@gmail.com", "2002-01-22", "Kanpur"),

("Anuradha", "anuradha123@gmail.com", "2003-01-22", "Lucknow"),

```
("RaviTeja", "ravi1523@gmail.com", "2001-11-21", "Delhi"),
("Krishna", "krishna1234@gmail.com", "2002-01-22", "Mathura"),
("Sapna", "sapna123@gmail.com", "2003-01-22", "Lucknow"),
("Ravi", "ravi153@gmail.com", "2001-11-21", "Delhi"),
("Viraaj", "viraaj1234@gmail.com", "2006-01-22", "Kanpur"),
("Raunak", "raunak123@gmail.com", "2003-01-22", "Lucknow"),
("Abhay", "abhay1523@gmail.com", "2001-11-21", "Delhi");
select * from Student;
CREATE TABLE Book (
   Book_id INT AUTO_INCREMENT PRIMARY KEY,
   Book_Name VARCHAR(100) NOT NULL,
   Author VARCHAR(100) NOT NULL,
   Publication_Year YEAR CHECK (Publication_Year >= 1990)
);

INSERT INTO Book (Book_Name, Author, Publication_Year)
VALUES
("Mathematics", "B.K. Gupta", 2000),
("Physics", "Newton", 2001),
("Chemistry", "R.K. Gupta", 2010),
("History", "Bipan Chandra", 2011),
("Biology", "B.P. Pandey", 2010);
Select * from book;
CREATE TABLE Book_Issue (
   Book_Issue_ID INT AUTO_INCREMENT PRIMARY KEY,
   Book_id INT NOT NULL,
   student_id INT NOT NULL,
```

Issue_Date DATE NOT NULL,

Return_Date DATE NOT NULL,

Status VARCHAR(20),

FOREIGN KEY (Book_id) REFERENCES Book (Book_id),

FOREIGN KEY (student_id) REFERENCES Student (Student_id)

);

desc Book_Issue;

INSERT INTO Book_Issue (Book_id, student_id, Issue_Date, Return_Date, Status)

VALUES

(1, 1, '2024-05-10', '2024-05-15', 'Returned');

select * from book_issue;

**Both tables are linked through foreign key.**

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| Book_Issue_ID | int | NO | PRI | NULL | auto_increment |
| Book_id | int | NO | MUL | NULL | |
| student_id | int | NO | MUL | NULL | |
| Issue_Date | date | NO | | NULL | |
| Return_Date | date | NO | | NULL | |
| Status | varchar(20) | YES | | NULL | |

**Student Table:**

| Student_id | Name | email | DOB | City |
|---|---|---|---|---|
| 1 | Rajesh | rajesh123@gmail.com | 2000-11-21 | Delhi |
| 2 | Radha | radha1234@gmail.com | 2002-01-22 | Kanpur |
| 3 | Anuradha | anuradha123@gmail.com | 2003-01-22 | Lucknow |
| 4 | RaviTeja | ravi1523@gmail.com | 2001-11-21 | Delhi |
| 5 | Krishna | krishna1234@gmail.com | 2002-01-22 | Mathura |
| 6 | Sapna | sapna123@gmail.com | 2003-01-22 | Lucknow |
| 7 | Ravi | ravi153@gmail.com | 2001-11-21 | Delhi |
| 8 | Viraaj | viraaj1234@gmail.com | 2006-01-22 | Kanpur |
| 9 | Raunak | raunak123@gmail.com | 2003-01-22 | Lucknow |
| 10 | Abhay | abhay1523@gmail.com | 2001-11-21 | Delhi |
| NULL | NULL | NULL | NULL | NULL |

**Books Table:**

| Book_id | Book_Name | Author | Publication_Year |
|---|---|---|---|
| 1 | Mathematics | B.K.Gupta | 2000 |
| 2 | Physics | Newton | 2001 |
| 3 | Chemistry | R.K.Gupta | 2010 |
| 4 | History | Bipan Chandra | 2011 |
| 5 | Biology | B.P.Pandey | 2010 |
| NULL | NULL | NULL | NULL |

**Book_Issue table:**

| Book_Issue_ID | Book_id | student_id | Issue_Date | Return_Date | Status |
|---|---|---|---|---|---|
| 5 | 1 | 1 | 2024-05-10 | 2024-05-15 | Returned |
| NULL | NULL | NULL | NULL | NULL | NULL |

**Assignment 3:** 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.

**Sol:**

**ACID Properties:**

ACID stands for Atomicity, Consistency, Isolation, and Durability, which are the four key properties that ensure reliability and integrity of transactions in a database system.

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 DATABASE BankDB;

Use BankDB;

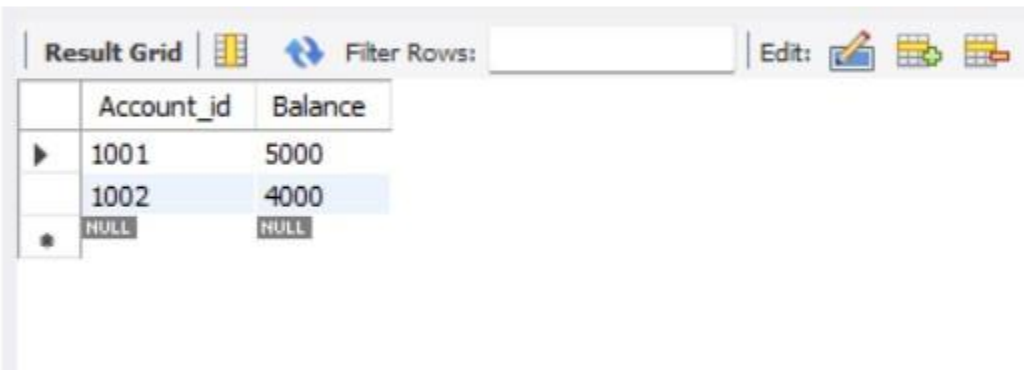
CREATE TABLE bank_accounts (Account_id INT PRIMARY KEY, Balance DECIMAL);

INSERT INTO bank_accounts (Account_id, Balance) VALUES

(1001, 5000.00),

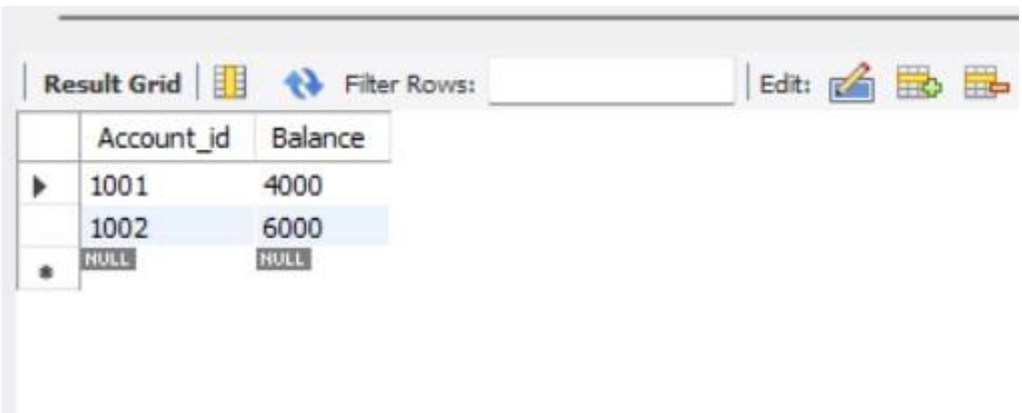
(1002, 4000.00);

SELECT * FROM bank_accounts;




| Account_id | Balance |
|---|---|
| 1001 | 5000 |
| 1002 | 4000 |
| NULL | NULL |

```
-- BEGIN TRANSACTION;

-- Withdrawal operation

UPDATE bank_accounts

SET Balance = Balance - 1000.00

WHERE Account_id = 1001;

UPDATE bank_accounts

SET Balance = Balance + 1000.00

WHERE account_id = 1002;

Commit;

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

SELECT balance FROM bank_accounts WHERE account_id = 1001;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

SELECT balance FROM bank_accounts WHERE account_id = 1001;

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;
```



**Assignment 4:** 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.

**Sol:**

```
CREATE DATABASE LibrarySystem;

USE LibrarySystem;

CREATE TABLE Student (

   Student_id INT AUTO_INCREMENT PRIMARY KEY,

   Name VARCHAR(100) NOT NULL,

   email VARCHAR(100) UNIQUE NOT NULL,

   DOB DATE NOT NULL,

   City VARCHAR(100)

);

INSERT INTO Student (Name, email, DOB, City)

VALUES

("Rajesh", "rajesh123@gmail.com", "2000-11-21", "Delhi"),

("Radha", "radha1234@gmail.com", "2002-01-22", "Kanpur"),

("Anuradha", "anuradha123@gmail.com", "2003-01-22", "Lucknow"),

("RaviTeja", "ravi1523@gmail.com", "2001-11-21", "Delhi"),

("Krishna", "krishna1234@gmail.com", "2002-01-22", "Mathura"),

("Sapna", "sapna123@gmail.com", "2003-01-22", "Lucknow"),

("Ravi", "ravi153@gmail.com", "2001-11-21", "Delhi"),

("Viraaj", "viraaj1234@gmail.com", "2006-01-22", "Kanpur"),

("Raunak", "raunak123@gmail.com", "2003-01-22", "Lucknow"),

("Abhay", "abhay1523@gmail.com", "2001-11-21", "Delhi");

select * from Student;

CREATE TABLE Book (

   Book_id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    Book_Name VARCHAR(100) NOT NULL,
    Author VARCHAR(100) NOT NULL,
    Publication_Year YEAR CHECK (Publication_Year >= 1990)
);

INSERT INTO Book (Book_Name, Author, Publication_Year)
VALUES
("Mathematics", "B.K. Gupta", 2000),
("Physics", "Newton", 2001),
("Chemistry", "R.K. Gupta", 2010),
("History", "Bipan Chandra", 2011),
("Biology", "B.P. Pandey", 2010);
Select * from book;
CREATE TABLE Book_Issue (
    Book_Issue_ID INT AUTO_INCREMENT PRIMARY KEY,
    Book_id INT NOT NULL,
    student_id INT NOT NULL,
    Issue_Date DATE NOT NULL,
    Return_Date DATE NOT NULL,
    Status VARCHAR (20),
    FOREIGN KEY (Book_id) REFERENCES Book (Book_id),
    FOREIGN KEY (student_id) REFERENCES Student (Student_id)
);

CREATE TABLE Old_Book (
    Book_id INT AUTO_INCREMENT PRIMARY KEY,
    Book_Name VARCHAR(100) NOT NULL,
```

```
    Author VARCHAR (100) NOT NULL,

    Publication_Year YEAR CHECK (Publication_Year >= 1990)

);


desc Book_Issue;

INSERT INTO Book_Issue (Book_id, student_id, Issue_Date, Return_Date, Status)

VALUES

(1, 1, '2024-05-10', '2024-05-15', 'Returned');

select * from book_issue;


Alter table book add Publisher varchar (50);

Drop table if exists old_book;

select * from book;
```

**Before Altering Table:**

| Book_id | Book_Name | Author | Publication_Year |
|---|---|---|---|
| 1 | Mathematics | B.K.Gupta | 2000 |
| 2 | Physics | Newton | 2001 |
| 3 | Chemistry | R.K.Gupta | 2010 |
| 4 | History | Bipan Chandra | 2011 |
| 5 | Biology | B.P.Pandey | 2010 |
| NULL | NULL | NULL | NULL |

**After Altering Table:**

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

| Book_id | Book_Name | Author | Publication_Year | Publisher |
|---|---|---|---|---|
| 1 | Mathematics | B.K.Gupta | 2000 | NULL |
| 2 | Physics | Newton | 2001 | NULL |
| 3 | Chemistry | R.K.Gupta | 2010 | NULL |
| 4 | History | Bipan Chandra | 2011 | NULL |
| 5 | Biology | B.P.Pandey | 2010 | NULL |
| NULL | NULL | NULL | NULL | NULL |

**Dropped Table: (Drop table if exists old_book;)**


**Assignment 5:** 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 in impact on query execution.

**Sol:**

CREATE DATABASE EmpDB;

USE EmpDB;

CREATE TABLE employees (

employee_id INT PRIMARY KEY,

first_name VARCHAR(50),

last_name VARCHAR(50),

department_id INT

);

INSERT into employees (employee_id, first_name, Last_name, department_id)

Values

(100, 'Ramesh', 'Gupta', 111),

(101, 'Rakesh', 'Gupta', 112),

(102, 'Ritesh', 'Gupta', 113);

Desc employees;

select * from employees;

-- Create an index on the department_id column

CREATE INDEX idx_department_id ON employees (department_id);

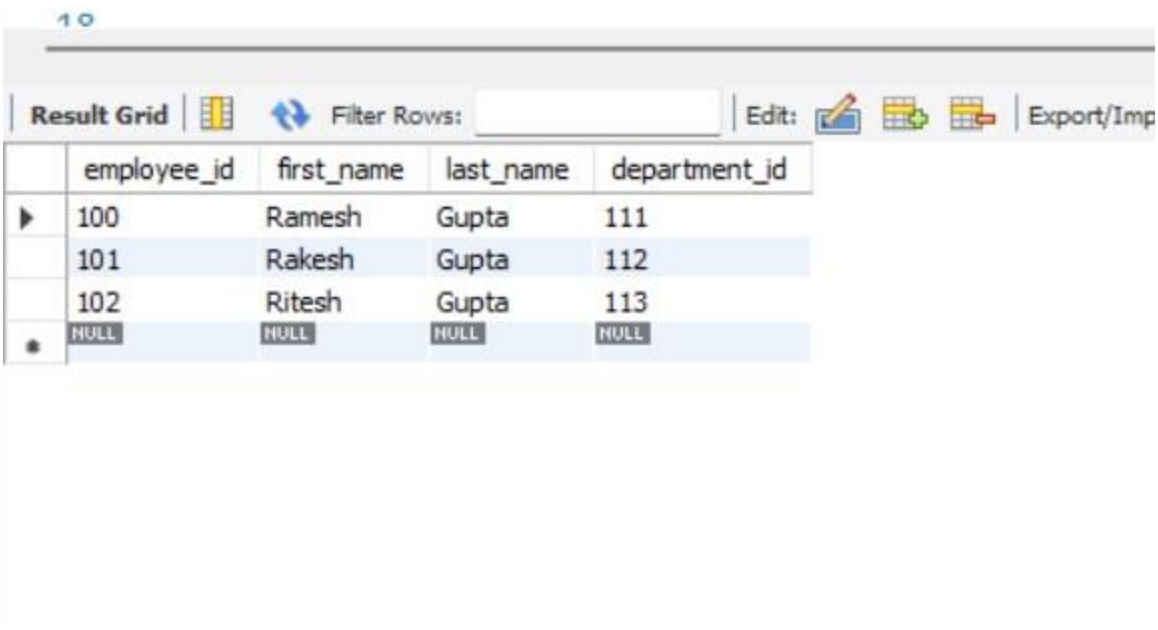
-- Query to find employees in a specific department

SELECT * FROM employees WHERE department_id = 111;

-- Drop the index on the department_id column

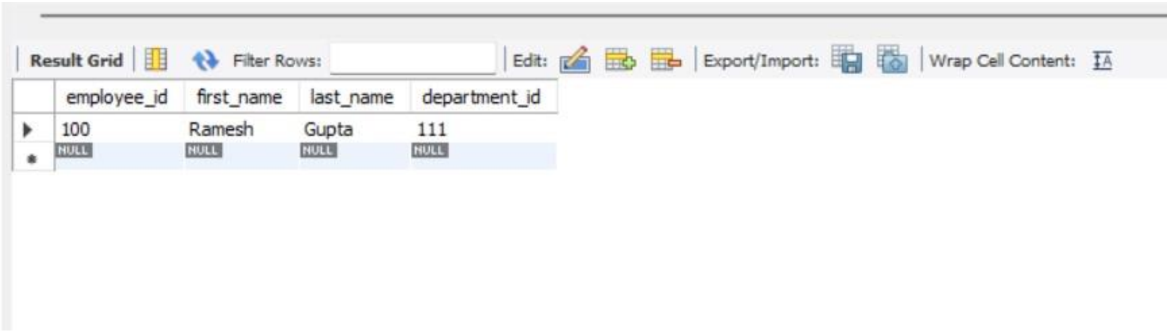
DROP INDEX idx_department_id ON employees;

**Table Employee list Screenshot:**

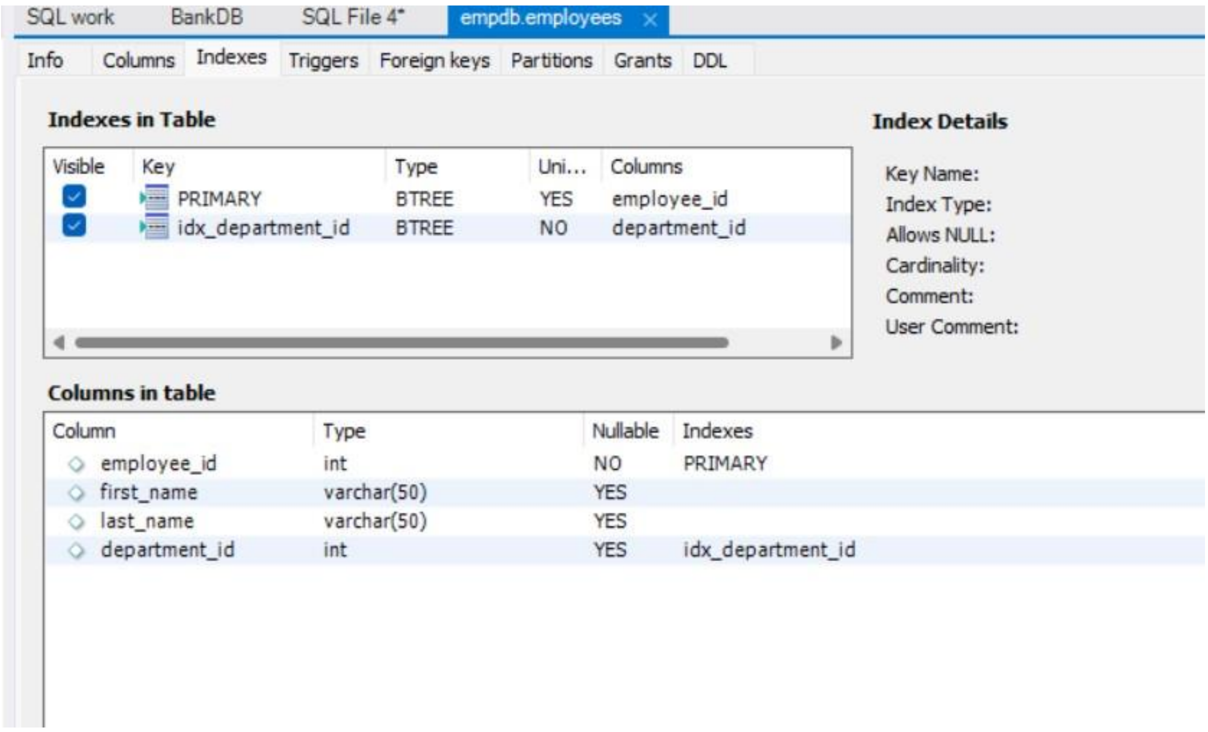


| employee_id | first_name | last_name | department_id |
|---|---|---|---|
| 100 | Ramesh | Gupta | 111 |
| 101 | Rakesh | Gupta | 112 |
| 102 | Ritesh | Gupta | 113 |
| NULL | NULL | NULL | NULL |

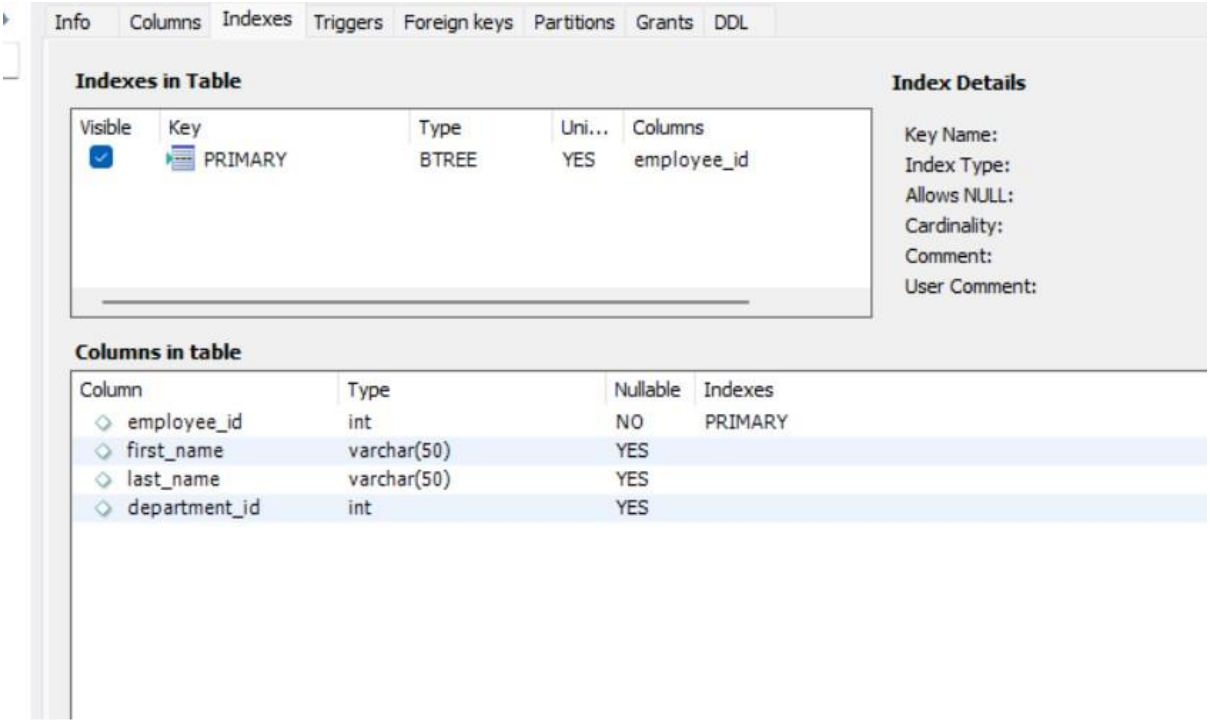
**Filtering Records where Department ID = 111.**




**Creating an Index :**




**Deleting above Index:**




**Assignment 6**: 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.

**Sol**:

CREATE USER 'new_user'@'SuneelVerma' IDENTIFIED BY 'password';

-- Grant privileges to the new user

GRANT SELECT, INSERT, UPDATE, DELETE ON my_database.* TO 'new_user'@'SuneelVerma';

-- Revoke certain privileges from the user

REVOKE DELETE ON my_database.* FROM 'new_user'@'SuneelVerma';

-- Drop the user

DROP USER 'new_user'@'SuneelVerma';

**Assignment 7:** 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.

**Sol:**

```
USE LibrarySystem;
CREATE TABLE Student (
   Student_id INT AUTO_INCREMENT PRIMARY KEY,
   Name VARCHAR(100) NOT NULL,
   email VARCHAR(100) UNIQUE NOT NULL,
   DOB DATE NOT NULL,
   City VARCHAR(100)
);
INSERT INTO Student (Name, email, DOB, City)
VALUES
("Rajesh", "rajesh123@gmail.com", "2000-11-21", "Delhi"),
("Radha", "radha1234@gmail.com", "2002-01-22", "Kanpur"),
("Anuradha", "anuradha123@gmail.com", "2003-01-22", "Lucknow"),
("RaviTeja", "ravi1523@gmail.com", "2001-11-21", "Delhi"),
("Krishna", "krishna1234@gmail.com", "2002-01-22", "Mathura"),
("Sapna", "sapna123@gmail.com", "2003-01-22", "Lucknow"),
("Ravi", "ravi153@gmail.com", "2001-11-21", "Delhi"),
("Viraaj", "viraaj1234@gmail.com", "2006-01-22", "Kanpur"),
("Raunak", "raunak123@gmail.com", "2003-01-22", "Lucknow"),
("Abhay", "abhay1523@gmail.com", "2001-11-21", "Delhi");
select * from Student;
CREATE TABLE Book (
   Book_id INT AUTO_INCREMENT PRIMARY KEY,
   Book_Name VARCHAR(100) NOT NULL,
   Author VARCHAR(100) NOT NULL,
   Publication_Year YEAR CHECK (Publication_Year >= 1990)
);

INSERT INTO Book (Book_Name, Author, Publication_Year)
VALUES
("Mathematics", "B.K.Gupta", 2000),
("Physics", "Newton", 2001),
("Chemistry", "R.K.Gupta", 2010),
("History", "Bipan Chandra", 2011),
("Biology", "B.P.Pandey", 2010);
Select * from book;
CREATE TABLE Book_Issue (
   Book_Issue_ID INT AUTO_INCREMENT PRIMARY KEY,
   Book_id INT NOT NULL,
   student_id INT NOT NULL,
   Issue_Date DATE NOT NULL,
   Return_Date DATE NOT NULL,
```

```sql
    Status VARCHAR(20),
    FOREIGN KEY (Book_id) REFERENCES Book (Book_id),
    FOREIGN KEY (student_id) REFERENCES Student (Student_id)
);

CREATE TABLE Old_Book (
    Book_id INT AUTO_INCREMENT PRIMARY KEY,
    Book_Name VARCHAR(100) NOT NULL,
    Author VARCHAR(100) NOT NULL,
    Publication_Year YEAR CHECK (Publication_Year >= 1990)
);
Select * from Old_book;

desc Book_Issue;
INSERT INTO Book_Issue (Book_id, student_id, Issue_Date, Return_Date, Status)
VALUES
(1, 1, '2024-05-10', '2024-05-15', 'Returned');
select * from book_issue;

Alter table book add Publisher varchar(50);
Drop table if exists old_book;
select * from book;
```

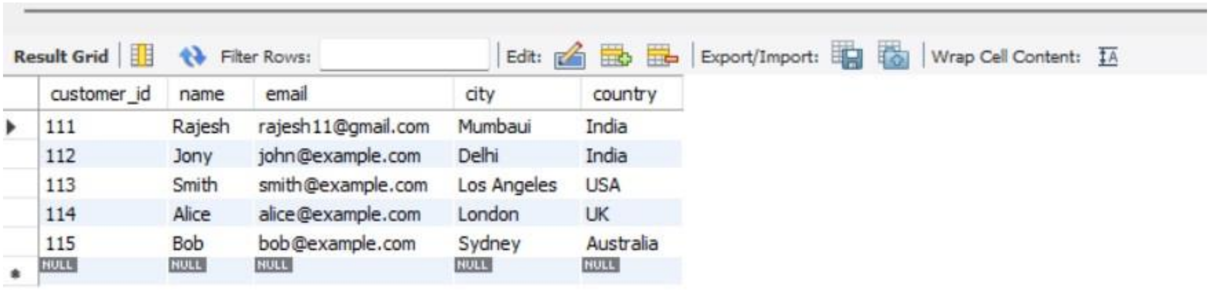
## -------------------------------------Day -2- Assignments-----------------------------------

**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.

**Sol:**

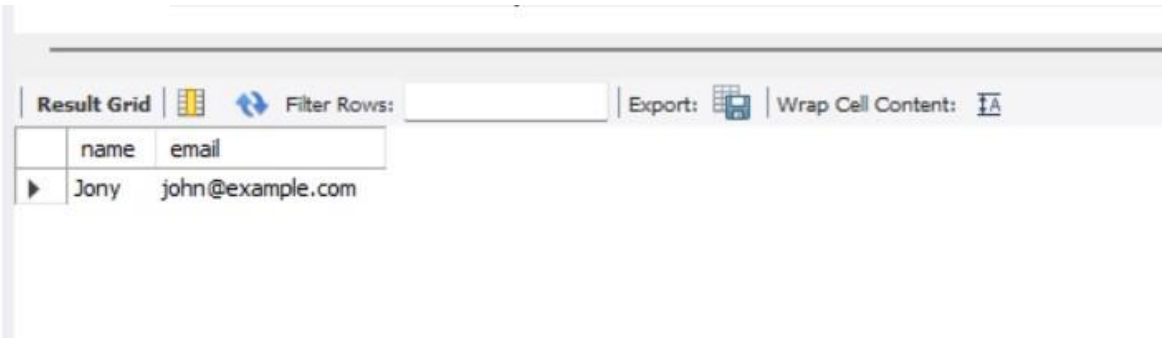
```
Create Database CustomerDB;

Use CustomerDB;

CREATE TABLE customers (

   customer_id INT PRIMARY KEY AUTO_INCREMENT,

   name VARCHAR(100),

   email VARCHAR(255),

   city VARCHAR(100),

   country VARCHAR(100)

);

INSERT INTO customers values (111, 'Rajesh','rajesh11@gmail.com', 'Mumbaui', 'India');

INSERT INTO customers (name, email, city, country)

VALUES

   ('Jony', 'john@example.com', 'Delhi', 'India'),

   ('Smith', 'smith@example.com', 'Los Angeles', 'USA'),

   ('Alice', 'alice@example.com', 'London', 'UK'),

   ('Bob', 'bob@example.com', 'Sydney', 'Australia');

SELECT * FROM customers;

SELECT name, email

FROM customers

WHERE city = 'Delhi';
```

**Employee Table:**




| customer_id | name | email | city | country |
|---|---|---|---|---|
| 111 | Rajesh | rajesh11@gmail.com | Mumbaui | India |
| 112 | Jony | john@example.com | Delhi | India |
| 113 | Smith | smith@example.com | Los Angeles | USA |
| 114 | Alice | alice@example.com | London | UK |
| 115 | Bob | bob@example.com | Sydney | Australia |
| NULL | NULL | NULL | NULL | NULL |

**After filtering Records:** (SELECT name, email FROM customers WHERE city = 'Delhi')




| name | email |
|---|---|
| Jony | john@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.

**Sol:**

CREATE DATABASE IF NOT EXISTS jointMethodsWorks;

USE jointMethodsWorks;

CREATE TABLE Products (

Product_ID INT AUTO_INCREMENT PRIMARY KEY,

Product_Name VARCHAR (100) NOT NULL,

Category_ID INT NOT NULL,

```
    Price INT
);
INSERT INTO Products (Product_Name, Category_ID, Price)
VALUES
('Amla Oil', 1, 250),
('Lux Soap', 2, 25),
('Dove Shampoo', 3, 450),
('Head & Shoulders', 4, 445),
('Dhoop Agarbatti', 5, 20),
('Colgate maxfresh', 6, 50);


CREATE TABLE Categories (
    Category_ID INT PRIMARY KEY,
    Category_Name VARCHAR(100) NOT NULL,
    Cat_Description VARCHAR (200)
);
INSERT INTO Categories (Category_ID, Category_Name, Cat_Description)
VALUES
(1, 'Oil', 'Amla oil is good for hair'),
(2, 'Soap', 'Lux soap is good.'),
(3, 'Shampoo', 'Dove shampoo is good for hair'),
(4, 'Shampoo', 'Head & Shoulder shampoo is good shampoo'),
(5, 'Agarbatti', 'Dhoop agarbatti purifies the air nicely'),
(6, 'Colgate', 'Colgate Maxfresh provides extra freshness');
```

**-- Inner Join**

```
SELECT P. Product_ID, P. Product_Name, P. Price, C. Cat_Description
FROM Products P
```

INNER JOIN Categories C ON P. Category_ID = C. Category_ID;

**-- Left Join**

SELECT P. Product_ID, P. Product_Name, C. Category_Name

FROM Products P

LEFT JOIN Categories C ON P. Category_ID = C. Category_ID;

**Inner join :**

```
-- Inner Join
SELECT P.Product_ID, P.Product_Name, P.Price, C.Cat_Description
FROM Products P
INNER JOIN Categories C ON P.Category_ID = C.Category_ID;
```

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

| Product_ID | Product_Name | Price | Cat_Description |
|---|---|---|---|
| 1 | Amla Oil | 250 | Amla oil is good for hair |
| 2 | Lux Soap | 25 | Lux soap is good. |
| 3 | Dove Shampoo | 450 | Dove shampoo is good for hair |
| 4 | Head & Shoulders | 445 | Head & Shoulder shampoo is good shampoo |
| 5 | Dhoop Agarbatti | 20 | Dhoop agarbatti purifies the air nicely |
| 6 | Colgate maxfresh | 50 | Colgate Maxfresh provides extra freshness |

**Left join:**

```sql
-- Left Join
SELECT P.Product_ID, P.Product_Name, C.Category_Name
FROM Products P
LEFT JOIN Categories C ON P.Category_ID = C.Category_ID;
```

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

| Product_ID | Product_Name | Category_Name |
|---|---|---|
| 1 | Amla Oil | Oil |
| 2 | Lux Soap | Soap |
| 3 | Dove Shampoo | Shampoo |
| 4 | Head & Shoulders | Shampoo |
| 5 | Dhoop Agarbatti | Agarbatti |
| 6 | Colgate maxfresh | Colgate |

**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.

**Sol:**

Create Database CustomerDB;

Use CustomerDB;

CREATE TABLE customers (

customer_id INT PRIMARY KEY AUTO_INCREMENT,

name VARCHAR(100),

email VARCHAR (255),

city VARCHAR (100),

country VARCHAR (100)

);

INSERT INTO customers values (111, 'Rajesh','rajesh11@gmail.com', 'Mumbaui', 'India');

INSERT INTO customers (name, email, city, country)

VALUES

('Jony', 'john@example.com', 'Delhi', 'India'),

('Smith', 'smith@example.com', 'Los Angeles', 'USA'),

```
    ('Alice', 'alice@example.com', 'London', 'UK'),
    ('Bob', 'bob@example.com', 'Sydney', 'Australia');
SELECT * FROM customers;
CREATE TABLE Orders (
    Order_ID INT AUTO_INCREMENT PRIMARY KEY,
    Customer_ID INT NOT NULL,
    Order_Date DATE NOT NULL,
    Order_Value DECIMAL (10, 2) NOT NULL,
    FOREIGN KEY (Customer_ID) REFERENCES Customers (Customer_ID)
);
INSERT INTO Orders (Customer_ID, Order_Date, Order_Value)
VALUES
(111, '2024-05-01', 100.00),
(112, '2024-05-02', 150.00),
(113, '2024-05-03', 200.00),
(111, '2024-05-04', 120.00),
(112, '2024-05-05', 180.00);
select * from orders;
SELECT *
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
    )
);
```

**After filtering records:**

| customer_id | name | email | city | country |
|---|---|---|---|---|
| 112 | Jony | john@example.com | Delhi | India |
| 113 | Smith | smith@example.com | Los Angeles | USA |
| NULL | NULL | NULL | NULL | NULL |

**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.

**Sol**:

```
Create database ProdutOrderDB;

use ProdutOrderDB;

-- Create the 'orders' table

CREATE TABLE orders (

   order_id INT PRIMARY KEY,

   customer_id INT,

   order_date DATE,

   total_amount DECIMAL(10, 2)

);


-- Inserting sample data into the 'orders' table

INSERT INTO orders (order_id, customer_id, order_date, total_amount)

VALUES

   (101, 1, '2024-05-19', 250.00),

   (102, 2, '2024-05-20', 150.00),

   (103, 3, '2024-05-21', 350.00);
```

```sql
    select * from orders;


-- Create the 'products' table
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    quantity INT
);
-- Inserting sample data into the 'products' table
INSERT INTO products (product_id, product_name, quantity)
VALUES
    (1, 'Product A', 100),
    (2, 'Product B', 150),
    (3, 'Product C', 200);
select * from products;
    -- Begin the transaction
-- BEGIN TRANSACTION;


-- Insert a new record into the 'orders' table
INSERT INTO orders (order_id, customer_id, order_date, total_amount)
VALUES (104, 1, '2024-05-19', 250.00);


-- Commit the transaction
COMMIT;


-- Update the 'products' table
UPDATE products
```

```
SET quantity = quantity - 1

WHERE product_id = 1;


-- Rollback the transaction

ROLLBACK;
```

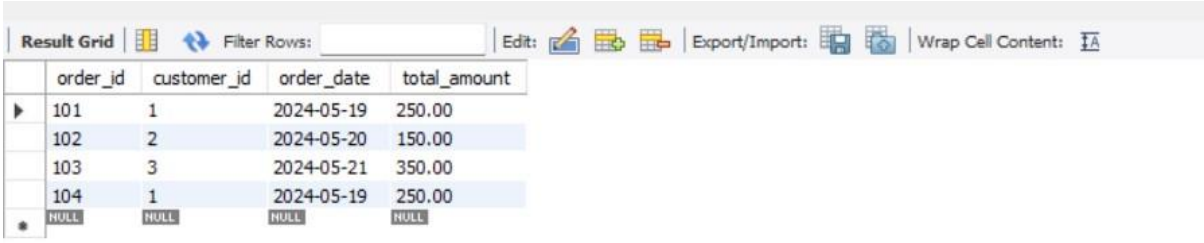
**Product Table:**

| product_id | product_name | quantity |
|---|---|---|
| 1 | Product A | 100 |
| 2 | Product B | 150 |
| 3 | Product C | 200 |
| NULL | NULL | NULL |

**Order Table:**

| order_id | customer_id | order_date | total_amount |
|---|---|---|---|
| 101 | 1 | 2024-05-19 | 250.00 |
| 102 | 2 | 2024-05-20 | 150.00 |
| 103 | 3 | 2024-05-21 | 350.00 |
| NULL | NULL | NULL | NULL |

**Updated Order table**




| order_id | customer_id | order_date | total_amount |
|---|---|---|---|
| 101 | 1 | 2024-05-19 | 250.00 |
| 102 | 2 | 2024-05-20 | 150.00 |
| 103 | 3 | 2024-05-21 | 350.00 |
| 104 | 1 | 2024-05-19 | 250.00 |
| NULL | NULL | NULL | NULL |

**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.

**Sol**

BEGIN TRANSACTION;

-- Perform the first INSERT into 'orders'

INSERT INTO orders (order_id, customer_id, order_date, total_amount)

VALUES (105, 1, '2024-05-19', 250.00);

-- Set the first SAVEPOINT

SAVEPOINT savepoint1;

-- Perform the second INSERT into 'orders'

INSERT INTO orders (order_id, customer_id, order_date, total_amount)

VALUES (106, 6, '2024-05-20', 150.00);

-- Set the second SAVEPOINT

SAVEPOINT savepoint2;

-- Perform the third INSERT into 'orders'

INSERT INTO orders (order_id, customer_id, order_date, total_amount)

VALUES (107, 7, '2024-05-21', 350.00);

-- Rollback to the second SAVEPOINT

ROLLBACK TO SAVEPOINT savepoint2;

-- Commit the overall transaction

COMMIT;

**Initial table:**

Result Grid | Filter Rows: | Edit: | Export/Import:

| order_id | customer_id | order_date | total_amount |
|---|---|---|---|
| 101 | 1 | 2024-05-19 | 250.00 |
| 102 | 2 | 2024-05-20 | 150.00 |
| 103 | 3 | 2024-05-21 | 350.00 |
| 104 | 1 | 2024-05-19 | 250.00 |
| NULL | NULL | NULL | NULL |

**SavePoint1 (Select * from orders;)**

| order_id | customer_id | order_date | total_amount |
|---|---|---|---|
| 101 | 1 | 2024-05-19 | 250.00 |
| 102 | 2 | 2024-05-20 | 150.00 |
| 103 | 3 | 2024-05-21 | 350.00 |
| 104 | 1 | 2024-05-19 | 250.00 |
| 105 | 1 | 2024-05-19 | 250.00 |
| 106 | 6 | 2024-05-20 | 150.00 |
| NULL | NULL | NULL | NULL |

**SavePoint2 (Select * from orders;)**

| order_id | customer_id | order_date | total_amount |
|---|---|---|---|
| 101 | 1 | 2024-05-19 | 250.00 |
| 102 | 2 | 2024-05-20 | 150.00 |
| 103 | 3 | 2024-05-21 | 350.00 |
| 104 | 1 | 2024-05-19 | 250.00 |
| 105 | 1 | 2024-05-19 | 250.00 |
| 106 | 6 | 2024-05-20 | 150.00 |
| 107 | 7 | 2024-05-21 | 350.00 |
| NULL | NULL | NULL | NULL |

**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.

**Sol:**

**Introduction:**

Transaction logs are a fundamental component of database management systems (DBMS) that play a crucial role in ensuring data integrity and facilitating recovery in the event of system failures or unexpected shutdowns. These logs record all modifications made to the database during transactions, providing a detailed record of changes that can be used for recovery purposes.

**Importance of Transaction Logs:**

**Data Integrity:** Transaction logs preserve the integrity of the database by logging all committed transactions. They provide a chronological record of changes, allowing for the reconstruction of data in the event of failures.

**Point-in-Time Recovery:** Transaction logs enable point-in-time recovery, allowing database administrators to restore the database to a specific moment before the failure occurred. This feature is particularly useful for recovering from human errors or logical corruption.

**Disaster Recovery:** Transaction logs serve as a critical component of disaster recovery strategies, ensuring that data remains accessible even in the face of catastrophic events such as hardware failures, natural disasters, or cyber-attacks.

**Reduced Downtime:** With transaction logs, recovery processes can be automated, reducing downtime and minimizing the impact on business operations. This helps organizations maintain high availability and meet service-level agreements (SLAs) with customers.

**Hypothetical Scenario:**

Imagine a financial institution that relies heavily on its database system to process transactions in real-time. One day, the database server experiences a sudden power outage due to a hardware failure, leading to an unexpected shutdown of the database system. As a result, critical financial data becomes inaccessible, posing a significant risk to the organization's operations and reputation.

**In this scenario, transaction logs prove to be instrumental in data recovery:**

**1.Identification of Last Consistent State:** Upon restarting the database system, administrators analyze the transaction logs to identify the last consistent state of the database before the shutdown occurred.

**2.Transaction Rollback:** The database system rolls back any incomplete or uncommitted transactions recorded in the transaction logs, ensuring that partial changes are not applied to the database.

**3.Redo Operations:** Following rollback, the system replays committed transactions from the transaction logs, applying changes to the database and restoring it to a consistent state.

**4.Point-in-Time Recovery:** Database administrators utilize transaction logs to perform point-in-time recovery, restoring the database to the state immediately before the unexpected shutdown occurred. This enables the institution to resume normal operations with minimal data loss and downtime.

**Conclusion:**

Transaction logs are a critical component of data management and recovery strategies in modern database systems. By capturing all changes made to the database, transaction logs enable organizations to recover from failures and ensure the integrity and availability of their data. Implementing robust transaction logging practices is essential for safeguarding against data loss and maintaining business continuity in the face of unexpected events.