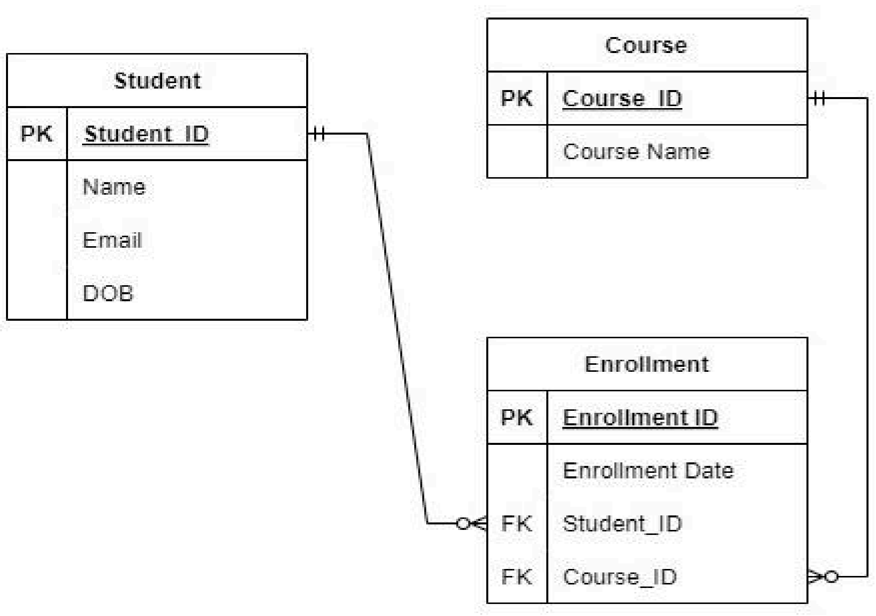
DBMS ASSIGNMENTS

DAY 1

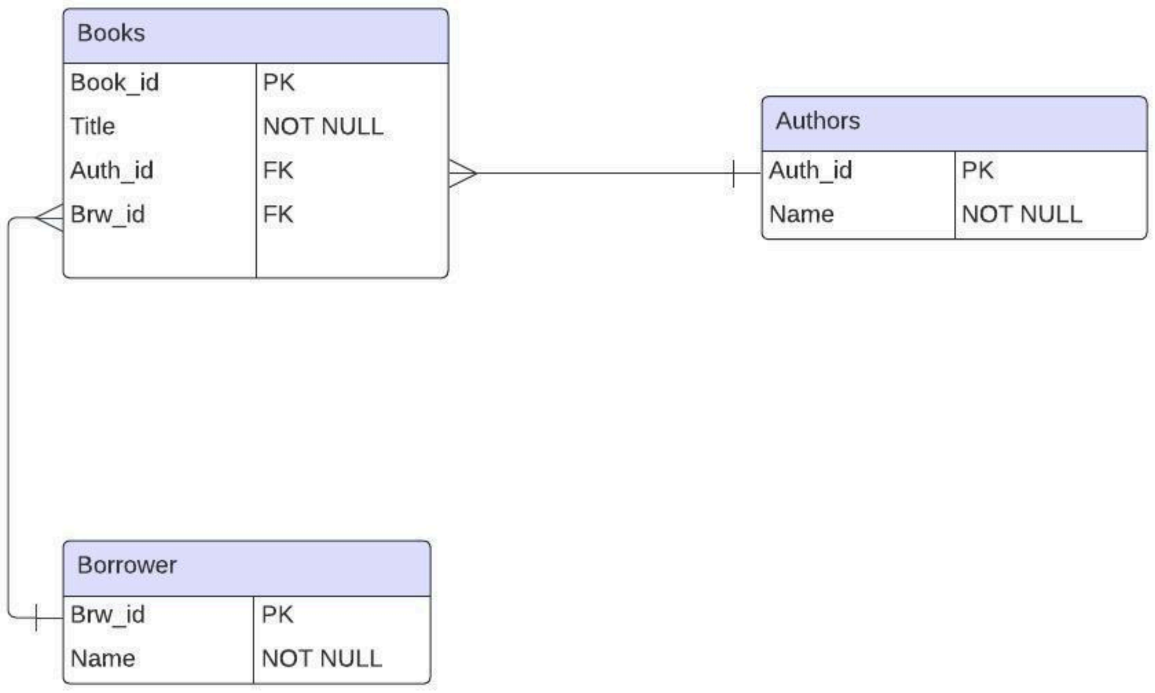
Assignment1:
Below is an ER Diagram of a University course enrollment system that tracks students,courses and enrollments:



In this scenario, all tables (Student, Course, and Enrollment) adhere to 3NF:

- Student: Student_ID is the primary key, and all attributes are dependent on Student ID.
- Course: Course_ID is the primary key, and all attributes are dependent on Course ID.
- Enrollment: Enrollment_ID is the primary key, and Enrollment Date, Student ID, and Course ID are dependent on Enrollment ID.

Assignment 2:
Database Schema for Library System:



SQL Query:
Author Table : CREATE TABLE Authors(Auth_id number PRIMARY KEY, Name VARCHAR(30)
NOT NULL)
Borrower Table : CREATE TABLE Borrower(Brw_id number PRIMARY KEY, Name
VARCHAR(20) NOT NULL)
Books Table : CREATE TABLE Books(Book_id number PRIMARY KEY, Title VARCHAR(20)
NOT NULL, Auth_id references Authors(Auth_id), Brw_id references
Borrower(Brw_id),CONSTRAINT c_brw CHECK(Brw_id IS NULL or Brw_id
IN(SELECT Brw_id FROM Borrower));

Assignment3:
The ACID properties of a transaction are:

- Atomicity: All parts of a transaction must be completed successfully, or none of them are applied. Kind of like an "all or nothing" rule.
- Consistency: Transactions must keep the database in a valid state, following all rules and constraints.
- Isolation: Transactions should not interfere with each other. Each transaction should happen independently.
- Durability: Once a transaction is committed, its changes are permanent, even if the system crashes.

Taking the above LibrarySystem database as an example to write SQL queries for ACID properties below::

```
START TRANSACTION;
SELECT * FROM Books WHERE book_id = 1 FOR UPDATE;
UPDATE Books SET borrower_id = 1 WHERE book_id = 1;
COMMIT;
```

1. Atomicity: The transaction ensures that both the UPDATE operations (changing the borrower_id) are either fully completed or none are applied. If any part fails, the entire transaction is rolled back.
2. Isolation: The SELECT ... FOR UPDATE statement locks the row to prevent other transactions from modifying it until the current transaction is complete.
3. Consistency: All these transactions ensure that the database remains in a valid state by following rules and constraints (e.g., foreign key constraints).
4. Durability: Once the COMMIT statement is executed, all changes made by the transaction are permanent and will survive any system failure.

Assignment 4:

SQL Query:

```
Create Database LibrarySystemDB;
Use LibrarySystemDB;
 CREATE TABLE Authors(Auth_id number PRIMARY KEY, Name VARCHAR(30) NOT NULL);
 CREATE TABLE Borrower(Brw_id number PRIMARY KEY, Name VARCHAR(20) NOT NULL);
 CREATE TABLE Books(Book_id number PRIMARY KEY, Title VARCHAR(20)  NOT NULL,
                    Auth_id  references Authors(Auth_id), Brw_id references
                    Borrower(Brw_id),CONSTRAINT c_brw CHECK(Brw_id IS NULL or Brw_id
                     IN(SELECT Brw_id FROM Borrower));
Alter Table Borrower add(Address Varchar(10))
Create table Redunt(id Number, name varchar(20)
DROP Table Redunt
```

Assignment 5:

Lets create an index on the Books table above:

Query: Create index ind on Books(auth_id)

By adding this index, finding books by author becomes quicker as the database knows exactly where to look in a big database, kind of like how a book index helps find specific topics. This helps the database to do less work and reduces the time to find the books and hence performance increases.

Lets drop the index:

Query: Drop index ind;

After removing the index, finding books by author might take longer since the database has to search through all the books again. But removing also meaning less work for the database in adding or removing books from the database.

In summary, while indexes can significantly improve query performance by facilitating faster data retrieval, they also come with overhead. It's essential to carefully evaluate the trade-offs and consider factors such as query patterns, data volume, and performance requirements when deciding whether to create or drop an index.

Assignment6:

SQL Query:

Create user ‘user’ identified by ‘password’;

Grant select,Insert,update on LibrarySystemDB.* To ‘user’;

Revoke update on LibrarySystemDB.* from ‘user’;

Drop user ‘user’;

Assignment7:

SQL Query:

```
INSERT - insert into Authors values(1,’JK’);
              Insert into Borrower(1,’DPG’);
              Insert into Books(‘1’,’Maths’,1,1);
UPDATE - update Authors set name = ‘Goswami’ Where auth_id = 1;
              update Borrower set name = ‘Prabal’ Where Brw_id = 1;
              update Books set title = ‘Science’ Where book_id = 1;
DELETE - Delete from Books Where book_id = 1;
              Delete from Authors Where auth_id = 1;
              Delete from Borrower Where brw_id = 1;
```

BULK INSERT

```
    Load Data Infile ‘/path/example.csv
    Into Table Books
    Fields Terminated By ‘,’
    Enclosed By ‘ “ ‘
    Lines terminated by ‘\n’
    Ignore 1 lines
```

DAY 2:

Assignment1:

Query to retrieve all columns: SELECT * FROM customers;

Query to retrieve specific columns :

```
       SELECT customer_name, email FROM customers  WHERE city = 'Guwahati’;
```

Assignment2:

Query:

```
    SELECT   c.customer_name,c.email,o.order_id, o.order_amount
    FROM customers c
    LEFT JOIN  orders o ON c.customer_id = o.customer_id
    WHERE c.region = 'Assam';
```

Assignment3:

Query:

```
  SELECT DISTINCT c.customer_id, c.customer_name, c.email
  FROM customers c JOIN orders o ON c.customer_id = o.customer_id
  WHERE o.order_amount > (  SELECT AVG(order_amount)   FROM orders)
  UNION
  SELECT customer_id, customer_name, email  FROM customers;
```

Assignment4:

Query:

```
  BEGIN TRANSACTION;
  INSERT INTO orders VALUES (4,104, 250.00);
  COMMIT;
  BEGIN TRANSACTION;
  UPDATE products SET stock_quantity = stock_quantity - 10 WHERE product_id = 9;
  ROLLBACK;
```

Assignment5:

Query:

```
 BEGIN TRANSACTION;
 INSERT INTO orders VALUES (5, 105,150.00);
 SAVEPOINT s1;
 INSERT INTO orders VALUES (6,106, 200.00);
 SAVEPOINT s2;
 INSERT INTO orders VALUES (7, 107, 500.00);
 SAVEPOINT s3;
 ROLLBACK TO SAVEPOINT s2;
 COMMIT;
```

Assignment6:

Report on the Importance of Transaction Logs for Data Recovery

Introduction:

Transaction logs are pivotal for maintaining data integrity and facilitating recovery in database systems. They serve as a comprehensive record of all database transactions, aiding in restoring data to a consistent state following unexpected shutdowns or failures.

Purpose of Transaction Logs:

Transaction logs act as a detailed account of changes made to the database. They document each transaction, including inserts, updates, or deletes, along with relevant data and timestamps. This information is indispensable for recovering databases to a consistent state after system disruptions.

How Transaction Logs Aid Data Recovery:

In the event of a database system failure, transaction logs are crucial for guiding the recovery process. By replaying the logged transactions, the system can reapply the changes that were underway at the time of the failure. This ensures data consistency and prevents loss or corruption of information.

Hypothetical Scenario:

Let's take an online retailer that manages a database of customer orders, storing transaction details, customer information, purchased items, and order statuses. During a routine day, the database server experiences an abrupt power outage due to a hardware malfunction.

Role of Transaction Logs in Recovery:

In this scenario, transaction logs play a pivotal role in database recovery. Upon system reboot, the database identifies the unexpected shutdown and consults the transaction logs. It identifies incomplete transactions and proceeds to replay them sequentially, restoring each order transaction to its pre-failure state.

Recovery Process:

Utilizing transaction logs, the system applies the recorded transactions to the database, ensuring the preservation of order data without loss or duplication. This meticulous approach restores the database to a consistent state, ensuring data integrity and system reliability.

Conclusion:

Transaction logs are indispensable for data recovery in database systems. Their role in maintaining a comprehensive record of transactions enables efficient recovery from unexpected failures while upholding data integrity. Implementing robust transaction logging mechanisms is imperative for ensuring the resilience and reliability of database systems.