Assignment no 7

1] Analyze 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.

Solution:-
Let's consider a scenario for a university system:

Business Scenario:

A university wants to create a database system to manage its student enrollment, courses, and faculty information. The university offers multiple courses, each taught by one or more faculty members. Students can enroll in multiple courses, and each course can have multiple enrolled students. Each faculty member can teach multiple courses.

Entities:

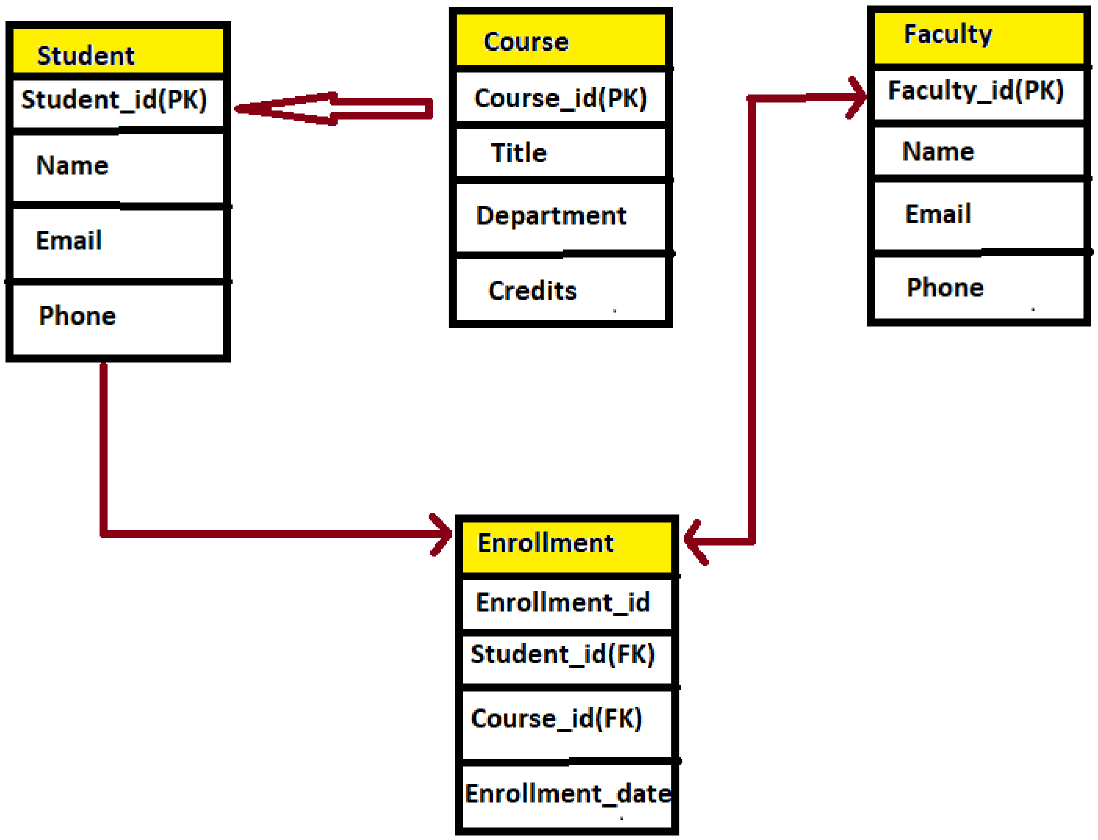
1. Student:
    - Attributes: Student_ID (Primary Key), Name, Email, Phone
2. Course:
    - Attributes: Course_ID (Primary Key), Title, Department, Credits
3. Faculty:
    - Attributes: Faculty_ID (Primary Key), Name, Email, Phone
4. Enrollment:
    - Attributes: Enrollment_ID (Primary Key), Student_ID (Foreign Key), Course_ID (Foreign Key), Enrollment_Date

Relationships:

1. Teaches:
    - Faculty teaches Course
    - Cardinality: One Faculty can teach Many Courses (1
)
2. Enrolls:
    - Student enrolls in Course

- Cardinality: One Student can enroll in Many Courses, and One Course can have Many Students (M
)

ER Diagram:



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 to establish relationships between tables.

Solution:-

```
-- Table for storing information about books
CREATE TABLE Books (
    BookID INT PRIMARY KEY,
    Title VARCHAR(100) NOT NULL,
    Author VARCHAR(100) NOT NULL,
    PublicationYear INT,
    ISBN VARCHAR(20) UNIQUE
);

-- Table for storing information about library members
CREATE TABLE Members (
    MemberID INT PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Email VARCHAR(100) UNIQUE
);

-- Table for storing information about borrowing transactions
CREATE TABLE Transactions (
    TransactionID INT PRIMARY KEY,
    BookID INT,
    MemberID INT,
    BorrowDate DATE NOT NULL,
    ReturnDate DATE,
    CONSTRAINT FK_Books FOREIGN KEY (BookID) REFERENCES
Books(BookID),
    CONSTRAINT FK_Members FOREIGN KEY (MemberID) REFERENCES
Members(MemberID)
);
```

In this schema:

1. **Books:** Contains information about books, including a unique BookID, title, author, publication year, and ISBN. BookID serves as the primary key, ensuring each book entry is unique. ISBN is marked as UNIQUE to ensure there are no duplicate ISBNs.
2. **Members:** Stores information about library members, including a unique MemberID, first name, last name, and email address. MemberID serves as the primary key, and email is marked as UNIQUE to ensure each member has a unique email address.

3. **Transactions:** Records borrowing transactions, with each transaction having a unique TransactionID. It includes foreign keys BookID and MemberID, establishing relationships with the Books and Members tables, respectively. BorrowDate indicates when the book was borrowed, while ReturnDate records when it was returned. BookID and MemberID constraints ensure that the values in these columns must exist in the corresponding tables.

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.

Solution:-

ACID Properties are:-

1] Atomicity: This property ensures that a transaction is treated as a single unit of work. It means that either all operations within the transaction are completed successfully, or none of them are. There's no partial execution; it's all or nothing.

2] Consistency: This property ensures that the database remains in a consistent state before and after the transaction. It means that any data modifications made by the transaction must adhere to the predefined rules and constraints of the database.

3] Isolation: Isolation ensures that the intermediate state of a transaction is invisible to other transactions until it's committed. It prevents interference between concurrent transactions, ensuring that they do not affect each other's results.

4] Durability: Once a transaction is committed, the changes made by that transaction are permanent and will not be lost, even in the event of a system failure. Durability guarantees that the committed changes are stored safely and persistently.

Now, let's demonstrate different isolation levels in SQL using a simple example of transferring funds between two bank accounts. We'll use SQL statements to simulate a transaction that includes locking:

– Enable explicit transaction mode

```
SET autocommit = 0;

– Start a transaction

BEGIN TRANSACTION;

– Update the balance of account A

UPDATE Accounts

SET Balance = Balance - 100

WHERE AccountID = 'A';

– Add funds to account B

UPDATE Accounts

SET Balance = Balance + 100

WHERE AccountID = 'B';

– Commit the transaction

COMMIT;
```

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.

Solution:-

SQL statements to create a new database, tables, and make modifications to the table structures for the library schema:

```
-- Create a new database
CREATE DATABASE LibraryDB;

-- Use the newly created database
USE LibraryDB;

-- Create table for storing information about books
CREATE TABLE Books (
  BookID INT PRIMARY KEY,
  Title VARCHAR(100) NOT NULL,
  Author VARCHAR(100) NOT NULL,
  PublicationYear INT,
  ISBN VARCHAR(20) UNIQUE
);

-- Create table for storing information about library members
CREATE TABLE Members (
  MemberID INT PRIMARY KEY,
  FirstName VARCHAR(50) NOT NULL,
  LastName VARCHAR(50) NOT NULL,
  Email VARCHAR(100) UNIQUE
);

-- Create table for storing information about borrowing transactions
CREATE TABLE Transactions (
  TransactionID INT PRIMARY KEY,
  BookID INT,
  MemberID INT,
  BorrowDate DATE NOT NULL,
  ReturnDate DATE,
  CONSTRAINT FK_Books FOREIGN KEY (BookID) REFERENCES Books(BookID),
    CONSTRAINT FK_Members FOREIGN KEY (MemberID) REFERENCES Members(MemberID)
);

-- Modify the structure of the Books table to add a new column 'Genre'
ALTER TABLE Books
ADD Genre VARCHAR(50);

-- Remove the redundant 'Transactions' table and replace it with a new 'Borrowings' table
DROP TABLE Transactions;

-- Create a new table for storing borrowing transactions
CREATE TABLE Borrowings (
    BorrowingID INT PRIMARY KEY,
    BookID INT,
    MemberID INT,
    BorrowDate DATE NOT NULL,
    ReturnDate DATE,
    CONSTRAINT FK_Books FOREIGN KEY (BookID) REFERENCES Books(BookID),
    CONSTRAINT FK_Members FOREIGN KEY (MemberID) REFERENCES Members(MemberID)
);
```

- We first create a new database called `LibraryDB`.
- Inside the database, we create three tables: `Books`, `Members`, and `Transactions` (which will later be replaced with `Borrowings`).
- We then use the `ALTER TABLE` statement to modify the structure of the `Books` table by adding a new column called `Genre`.
- Next, we use the `DROP TABLE` statement to remove the redundant `Transactions` table.

- Finally, we create a new table called `Borrowings` to replace the `Transactions` table, which stores borrowing transactions with a similar structure but with a different name.

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 analyze the impact on query execution.

Solution:-

let's first demonstrate creating an index on the `Title` column of the `Books` table in our library database, and then discuss its impact on query performance:

-- Create an index on the 'Title' column of the 'Books' table

CREATE INDEX idx_title ON Books(Title);

Creating an index on the `Title` column will improve the performance of queries that involve searching or sorting by the book title. When a query filters or sorts data based on the indexed column (`Title` in this case), the database engine can use the index to quickly locate the relevant rows without having to scan the entire table sequentially. This significantly reduces the time required to execute such queries, especially on large datasets.

Now, let's analyze the impact on query execution:

-- Run a SELECT query without the index

SELECT * FROM Books WHERE Title = 'Harry Potter';

This query will still work, but without the index, the database engine may need to perform a full table scan to find all rows where the title matches 'Harry Potter'. This can be inefficient, especially if the table is large.

```
– Run a SELECT query with the index

SELECT * FROM Books WHERE Title = 'Harry Potter';
```

With the index in place, the database engine can quickly locate the rows with the specified title by using the index. This leads to much faster query execution compared to when the index is not present.

Now, let's remove the index and observe the impact:

```
– Drop the index

DROP INDEX idx_title ON Books;
```

After dropping the index, queries that rely on searching or sorting by the book title may experience degraded performance, as the database engine no longer has the optimized index to quickly locate the relevant rows. Instead, it may resort to full table scans, leading to slower query execution, especially on large datasets.

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.

Solution:-

Here's how you can create a new database user, grant specific privileges, revoke certain privileges, and finally drop the user:

```
– Create a new user

CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';

– Grant privileges to the user
```

GRANT SELECT, INSERT, UPDATE ON LibraryDB.* TO 'newuser'@'localhost';

-- Revoke UPDATE privilege from the user

REVOKE UPDATE ON LibraryDB.* FROM 'newuser'@'localhost';

-- Drop the user

DROP USER 'newuser'@'localhost';

1. We create a new user named `newuser` with the password `'password'`. Replace `'localhost'` with the appropriate host if the user is accessing the database from a remote location.
2. We grant the SELECT, INSERT, and UPDATE privileges on all tables in the `LibraryDB` database to the user.
3. We then revoke the UPDATE privilege from the user. Now, the user will have SELECT and INSERT privileges only.
4. Finally, we drop the user from the database. This removes the user and all associated privileges.

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.

Solution:-

Here's a series of SQL statements to INSERT, UPDATE, and DELETE records in the library tables, including BULK INSERT operations to load data from an external source:

-- Insert new records into the Books table

INSERT INTO Books (BookID, Title, Author, PublicationYear, ISBN, Genre)

VALUES

```
    (1, 'To Kill a Mockingbird', 'Harper Lee', 1960, '9780061120084', 'Fiction'),

    (2, '1984', 'George Orwell', 1949, '9780451524935', 'Science Fiction'),

    (3, 'The Great Gatsby', 'F. Scott Fitzgerald', 1925, '9780743273565', 'Classic');

-- Update existing record in the Members table

UPDATE Members

SET Email = 'newemail@example.com'

WHERE MemberID = 1;

-- Delete records from the Transactions table based on specific criteria

DELETE FROM Transactions

WHERE MemberID = 1;

-- Bulk insert data from an external source into the Books table

BULK INSERT Books

FROM 'C:\data\books_data.csv'  -- Path to the CSV file

WITH (

    FIELDTERMINATOR = ',',  -- Specify the field terminator

    ROWTERMINATOR = '\n',    -- Specify the row terminator

    FIRSTROW = 2             -- Specify the first row to start reading data
);
```

1. We use the `INSERT INTO` statement to insert new records into the `Books` table.
2. We use the `UPDATE` statement to update the email address of a member in the `Members` table.
3. We use the `DELETE FROM` statement to delete records from the `Transactions` table based on specific criteria, in this case, where the `MemberID` is 1.
4. We use the `BULK INSERT` statement to load data from an external CSV file (`books_data.csv`) into the `Books` table. The `FIELDTERMINATOR` and `ROWTERMINATOR` options specify how the fields and rows are terminated in the CSV file. `FIRSTROW` specifies the row number from which to start reading data. Adjust these options according to your CSV file format.