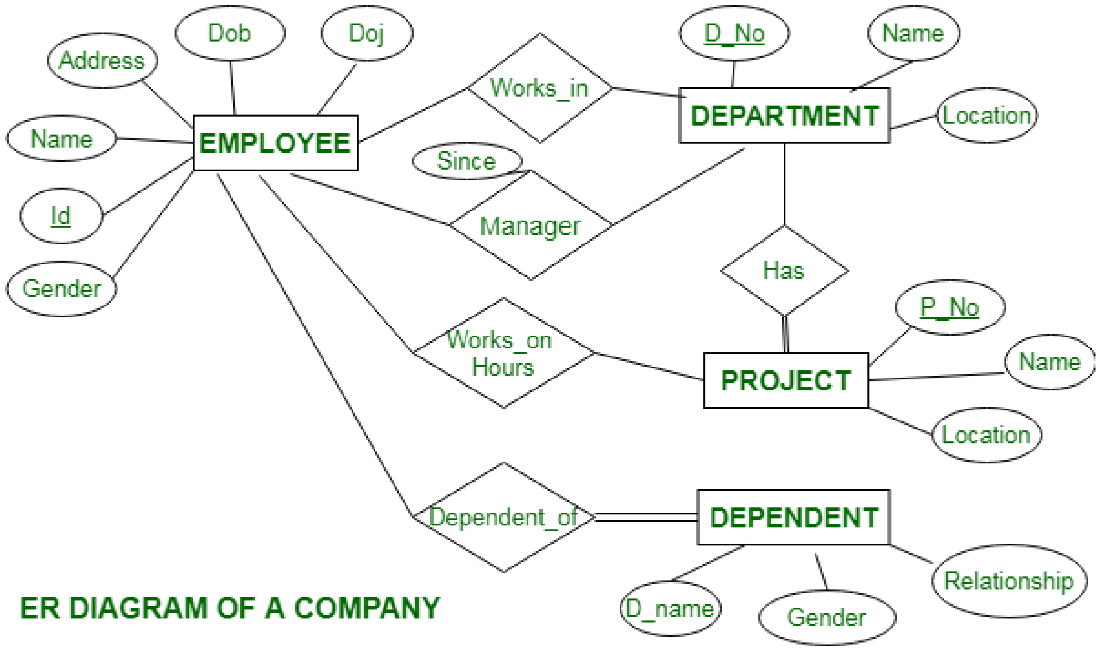
Day 8 & 9th Assignment

Prajwalhonashetti143@gmail.com

1. **Assignment 1: Analyse a given business scenario and create an ER diagram that includes entities, relationship, attributes, and cardinality. Ensure that the diagram reflects proper normalization up to third normal form.**

   ER diagram is known as Entity-Relationship Diagram, it is used to analyze to structure of the Database. It shows relationships between entities and their attributes. An ER Model provides a means of communication.



This Company ER diagram illustrates key information about Company, including entities such as employee, department, project and dependent. It allows to understand the relationships between entities.

Entities and their **Attributes** are

- **Employee Entity :** Attributes of Employee Entity are Name, Id, Address, Gender, Dob and Doj.
  Id is Primary Key for Employee Entity.
- **Department Entity :** Attributes of Department Entity are D_no, Name and Location.
  D_no is Primary Key for Department Entity.
- **Project Entity :** Attributes of Project Entity are P_No, Name and Location.
  P_No is Primary Key for Project Entity.
- **Dependent Entity :** Attributes of Dependent Entity are D_no, Gender and relationship.

Relationships are :

- **Employees works in Departments –**
  Many employee works in one Department but one employee can not work in many departments.
- **Manager controls a Department –**
  employee works under the manager of the Department and the manager records the date of joining of employee in the department.
- **Department has many Projects –**
  One department has many projects but one project can not come under many departments.
- **Employee works on project –**
  One employee works on several projects and the number of hours worked by the employee on a single project is recorded.
- **Employee has dependents –**
  Each Employee has dependents. Each dependent is dependent of only one employee.

Normalization
First Normal Form (1NF): Ensure that all columns contain atomic values, and each column contains only one type of value.
Second Normal Form (2NF): Ensure that the table is in 1NF and that all non-key attributes are fully functionally dependent on the primary key.
Third Normal Form (3NF): Ensure that the table is in 2NF and that all attributes are only dependent on the primary key.

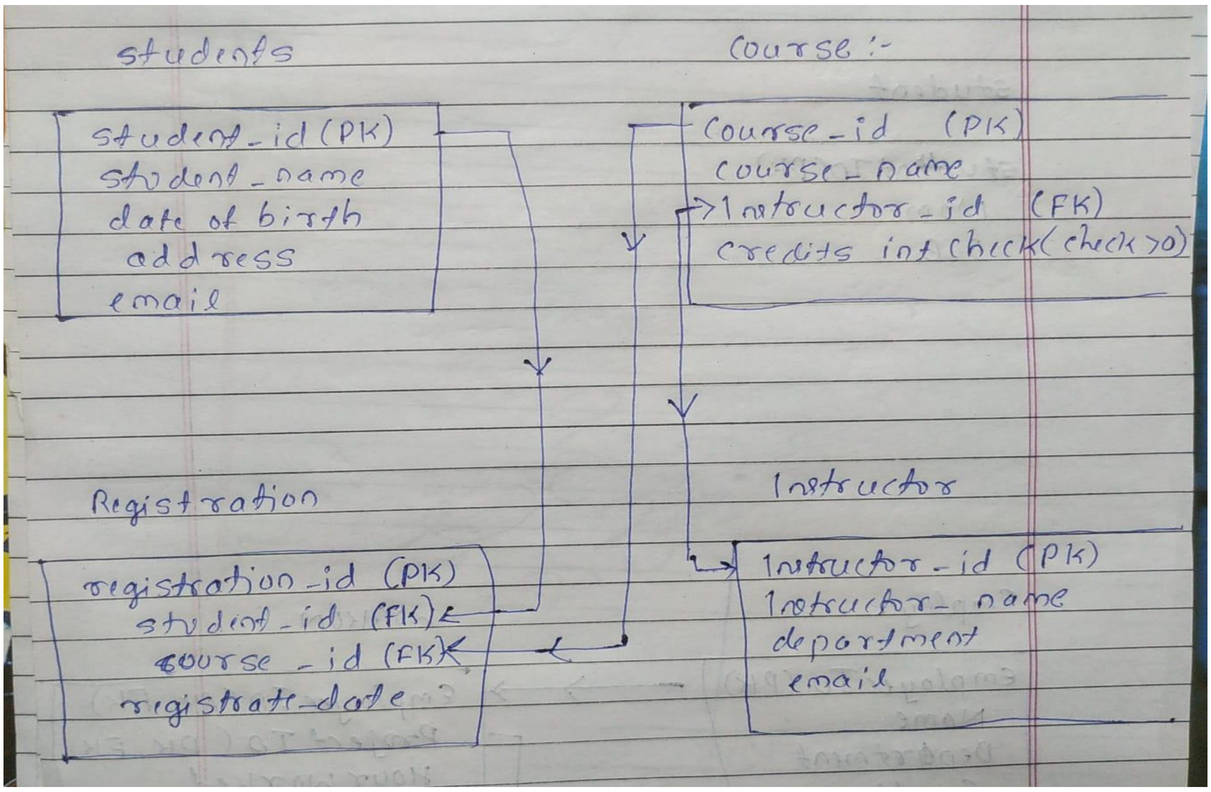
2. **Design a database schema for a library system. including tables ,fields, and constraints like NOT NULL, UNIQUE, and CHECK.**

Include primary and foreign keys to establish relationships between tables. let's design a schema for a university course registration system. This system will include tables for students, courses, registrations, and instructors.

Database Schema for University Course Registration System
Tables and Fields

Students:

```
CREATE TABLE students (
   student_id INT PRIMARY KEY AUTO_INCREMENT,
   student_name VARCHAR(100) NOT NULL,
   date_of_birth DATE,
   address VARCHAR(255),
   email VARCHAR(100) UNIQUE NOT NULL
);
```



Course:

```
CREATE TABLE courses (
    course_id INT PRIMARY KEY AUTO_INCREMENT,
    course_name VARCHAR(255) NOT NULL,
    instructor_id INT,
    credits INT CHECK (credits > 0),
    FOREIGN KEY (instructor_id) REFERENCES instructors(instructor_id)
);
```

Registration:

```
CREATE TABLE registrations (
    registration_id INT PRIMARY KEY AUTO_INCREMENT,
    student_id INT,
    course_id INT,
    registration_date DATE NOT NULL,
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
);
```

Instructor:

```
CREATE TABLE instructors (
   instructor_id INT PRIMARY KEY AUTO_INCREMENT,
   instructor_name VARCHAR(100) NOT NULL,
   department VARCHAR(100),
   email VARCHAR(100) UNIQUE NOT NULL
);
```

Explanation

Students Table: Stores information about students enrolled in the university. The email field is marked as UNIQUE to ensure each student has a unique email address.

Courses Table: Stores information about courses offered by the university. The credits field has a CHECK constraint to ensure it is greater than 0. The instructor_id field establishes a foreign key relationship with the instructors table.

Registrations Table: Records each registration made by a student for a course. It includes the student ID (student_id), course ID (course_id), and registration date (registration_date). Foreign keys are used to maintain referential integrity with the students and courses tables.

Instructors Table: Stores information about instructors teaching at the university. The email field is marked as UNIQUE to ensure each instructor has a unique email address.

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.**

ACID stands for Atomicity, Consistency, Isolation, and Durability. Here is a brief description of each of these properties:

Atomicity: Atomicity ensures that a transaction is treated as a single, indivisible unit of work. Either all the operations within the transaction are completed successfully, or none of them are.

Consistency: Consistency ensures that a transaction takes the database from one consistent state to another consistent state. The database is in a consistent state both before and after the transaction is executed. Constraints, such as unique keys and foreign keys, must be maintained to ensure data consistency.

Isolation: Isolation ensures that multiple transactions can execute concurrently without interfering with each other. Each transaction must be isolated from other transactions until it is completed

Durability: Durability ensures that once a transaction is committed, its changes are permanent and will survive any subsequent system failures. The transaction's changes are saved to the database permanently, and even if the system crashes, the changes remain intact and can be recovered.

SQL Statements and Isolation Levels
Scenario
Consider a simple scenario where we have **a bank_accounts** table with columns **account_id, balance,** and **last_updated.** We'll simulate a transfer of funds between two accounts using a transaction. We'll use locking to prevent concurrency issues, and we'll demonstrate different isolation levels

- **Create the bank_accounts table:**

```
CREATE TABLE bank_accounts (
   account_id INT PRIMARY KEY,
   balance DECIMAL(10, 2) NOT NULL,
   last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);

INSERT INTO bank_accounts (account_id, balance) VALUES
(1, 1000.00),
(2, 500.00);
```

- SQL Transaction to Transfer Funds

```
START TRANSACTION;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SELECT @balance1 := balance FROM bank_accounts WHERE account_id = 1;
SELECT @balance2 := balance FROM bank_accounts WHERE account_id = 2;

UPDATE bank_accounts SET balance = @balance1 - 200.00 WHERE account_id = 1;
UPDATE bank_accounts SET balance = @balance2 + 200.00 WHERE account_id = 2;

COMMIT;
```

- Demonstrate Different Isolation Levels
  Read Committed: Only committed data can be read.

  ```
  SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
  ```

START TRANSACTION;
SELECT @balance1 := balance FROM bank_accounts WHERE account_id = 1;

START TRANSACTION;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION;
SELECT @balance1 := balance FROM bank_accounts WHERE account_id = 1;

Repeatable Read: Read consistent snapshot of data.
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION;
SELECT @balance1 := balance FROM bank_accounts WHERE account_id = 1;

START TRANSACTION;
UPDATE bank_accounts SET balance = balance + 100.00 WHERE account_id = 1;
COMMIT;

Serializable: Prevents phantom reads and ensures serializable execution.

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
START TRANSACTION;
SELECT @balance1 := balance FROM bank_accounts WHERE account_id = 1;

START TRANSACTION;
UPDATE bank_accounts SET balance = balance + 100.00 WHERE account_id = 1;
COMMIT;

ASSIGNEMNT 4: write sql statements to CREATE a new databases 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.

Creating a New Database:

- **CREATE DATABASE Library;**

  USE Library;

Creating Tables

Assuming a basic library schema with tables for books, authors, and a junction table for book-author relationships:

- **Book table**

  ```
  CREATE TABLE Books (
     BookID INT PRIMARY KEY AUTO_INCREMENT,
     Title VARCHAR(255) NOT NULL,
     ISBN VARCHAR(20) NOT NULL,
     PublicationYear INT,
     Genre VARCHAR(50),
     Quantity INT NOT NULL DEFAULT 0
  );
  ```

- **Author table**

  ```
  CREATE TABLE Authors (
     AuthorID INT PRIMARY KEY AUTO_INCREMENT,
     Name VARCHAR(100) NOT NULL,
     Description TEXT
  );
  ```

- **Bookauthor junction table**

  ```
  CREATE TABLE BookAuthors (
     BookID INT,
     AuthorID INT,
     PRIMARY KEY (BookID, AuthorID),
     FOREIGN KEY (BookID) REFERENCES Books(BookID),
     FOREIGN KEY (AuthorID) REFERENCES Authors(AuthorID)
  );
  ```

- **Adding a new column to the Books table**

  ```
  ALTER TABLE Books
  ADD COLUMN Language VARCHAR(50);
  ```

- **Example: Dropping a table named RedundantTable**

  ```
  DROP TABLE RedundantTable;
  ```

ASSIGNMENT 5 : Demonstrate the creation of an table and discuss how it improves query performance. Use a DROP INDEX statement to remove the index and analyze the impact on query execution.

Step 1: Create a Table

First, let's create a Sales table and insert some sample data:

```
CREATE TABLE Sales (
   SaleID INT PRIMARY KEY AUTO_INCREMENT,
   CustomerID INT,
   SaleDate DATE,
   Amount DECIMAL(10, 2)
);

INSERT INTO Sales (CustomerID, SaleDate, Amount)
VALUES
(101, '2023-01-01', 150.00),
(102, '2023-01-02', 200.50),
(103, '2023-01-03', 75.25),
(101, '2023-01-04', 100.75),
(104, '2023-01-05', 300.00),
(102, '2023-01-06', 50.00),
(105, '2023-01-07', 125.50),
```

Step 2: Create an Index

Next, we create an index on the **CustomerID** column to improve the performance of queries that filter by customer ID.

```
CREATE INDEX idx_customerid ON Sales(CustomerID);
```

Step 3: Analyze Query Performance with Index

To analyze the performance, we can use the EXPLAIN statement to see how the query execution plan changes with the index.

EXPLAIN SELECT * FROM Sales WHERE CustomerID = 101;

Expected Result:

The E**XPLAIN** output should indicate that the **index idx_customerid** is being used, which reduces the number of rows MySQL has to scan.

Step 4: Drop the Index

Now, we drop the index and analyze the performance impact.

DROP INDEX idx_customerid ON Sales;

Step 5: Analyze Query Performance without Index

Run the same query again and use EXPLAIN to see the difference.

EXPLAIN SELECT * FROM Sales WHERE CustomerID = 101;

Expected Result:

Without the index, the EXPLAIN output will likely show a full table scan, meaning MySQL has to examine every row in the table to find the matching records, which is less efficient.

Summary of Performance Impact

With Index: The query execution plan uses the index **idx_customerid,** resulting in a much faster lookup as MySQL can quickly locate the rows that match the query condition.

Without Index: The query execution plan involves a full table scan, which is slower because MySQL must read each row to determine if it matches the query condition.

ASSIGNMENT 6: Create a new databases user with specific privileges using the CREATE USER and GRANT commands, Then, write a script to REVOKE certain privileges and DROP the user.

Create a New User and Grant Privileges

-- Create a new database user

CREATE USER 'library_user'@'localhost' IDENTIFIED BY 'secure_password';

-- Grant specific privileges to the new user

GRANT SELECT, INSERT, UPDATE, DELETE ON LibraryDatabase.* TO 'library_user'@'localhost';

-- Apply the changes

FLUSH PRIVILEGES;

Revoke Certain Privileges:

Let's say we want to revoke the UPDATE and DELETE privileges from library_user.

-- Revoke UPDATE and DELETE privileges from the user

REVOKE UPDATE, DELETE ON LibraryDatabase.* FROM 'library_user'@'localhost';

-- Apply the changes

FLUSH PRIVILEGES;

Drop the User
-- Drop the user from the database
DROP USER 'library_user'@'localhost';

- Replace **'library_user'@'localhost'** with the appropriate username and host if different.
- Replace **'secure_password'** with a strong, secure password.
- Adjust the privileges as per your specific requirements. The example grants **SELECT, INSERT, UPDATE,** and **DELETE** initially, and then revokes **UPDATE** and **DELETE**.

This script covers creating a new user, granting privileges, revoking some of those privileges, and then dropping the user, ensuring that all changes are properly applied by using **FLUSH PRIVILEGES** after each grant or revoke operation.

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.

Here is a series of SQL statements that demonstrate how to insert new records into the library tables, update existing records, delete records based on specific criteria, and perform bulk insert operations

- **Insert into Books Table**

  ```
  INSERT INTO Books (Title, ISBN, PublicationYear, Genre, Quantity, Language)
  VALUES ('The Great Gatsby', '9780743273565', 1925, 'Fiction', 3, 'English'),
         ('1984', '9780451524935', 1949, 'Dystopian', 5, 'English');
  ```

- **Insert into Authors Table**

  ```
  INSERT INTO Authors (Name, Description)
  VALUES ('F. Scott Fitzgerald', 'American novelist and short story writer.'),
         ('George Orwell', 'English novelist, essayist, journalist and critic.');
  ```

- **Insert into BookAuthors Table**

  ```
  INSERT INTO BookAuthors (BookID, AuthorID)
  VALUES (1, 1),
  VALUES (2, 2);
  ```

- **Update Quantity of a Book**

  ```
  UPDATE Books
  SET Quantity = 10
  WHERE ISBN = '9780743273565';
  ```

- **Update Author Description**

  ```
  UPDATE Authors
  SET Description = 'American novelist, essayist, screenwriter, and short story writer.'
  WHERE Name = 'F. Scott Fitzgerald';
  ```

- **Delete a Book by Title**
```
DELETE FROM Books
WHERE Title = '1984';
```

- **Delete an Author by Name**
```
DELETE FROM Authors
WHERE Name = 'George Orwell';
```

Bulk Insert Operations

To perform bulk insert operations from an external source (e.g., a CSV file), you can use the LOAD DATA INFILE statement in MySQL. Here is an example assuming you have a CSV file for books and authors:

- **Bulk Insert Books**
```
LOAD DATA INFILE '/path/to/books.csv'
INTO TABLE Books
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(Title, ISBN, PublicationYear, Genre, Quantity, Language);
```

- **Bulk Insert Authors**
```
LOAD DATA INFILE '/path/to/authors.csv'
INTO TABLE Authors
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(Name, Description);
```

Example CSV File Format

books.csv
```
Title,ISBN,PublicationYear,Genre,Quantity,Language
"The Great Gatsby","9780743273565",1925,"Fiction",3,"English"
"1984","9780451524935",1949,"Dystopian",5,"English"
```

authors.csv
```
Name,Description
"F. Scott Fitzgerald","American novelist and short story writer."
```

"George Orwell","English novelist, essayist, journalist and critic."

- Adjust file paths in LOAD DATA INFILE to match the actual locations of your CSV files.
- Make sure your MySQL server has permission to read the files and that the files are formatted correctly.
- Use appropriate file paths and ensure that your MySQL server is configured to allow the LOAD DATA INFILE operation (this may require setting the local_infile parameter).