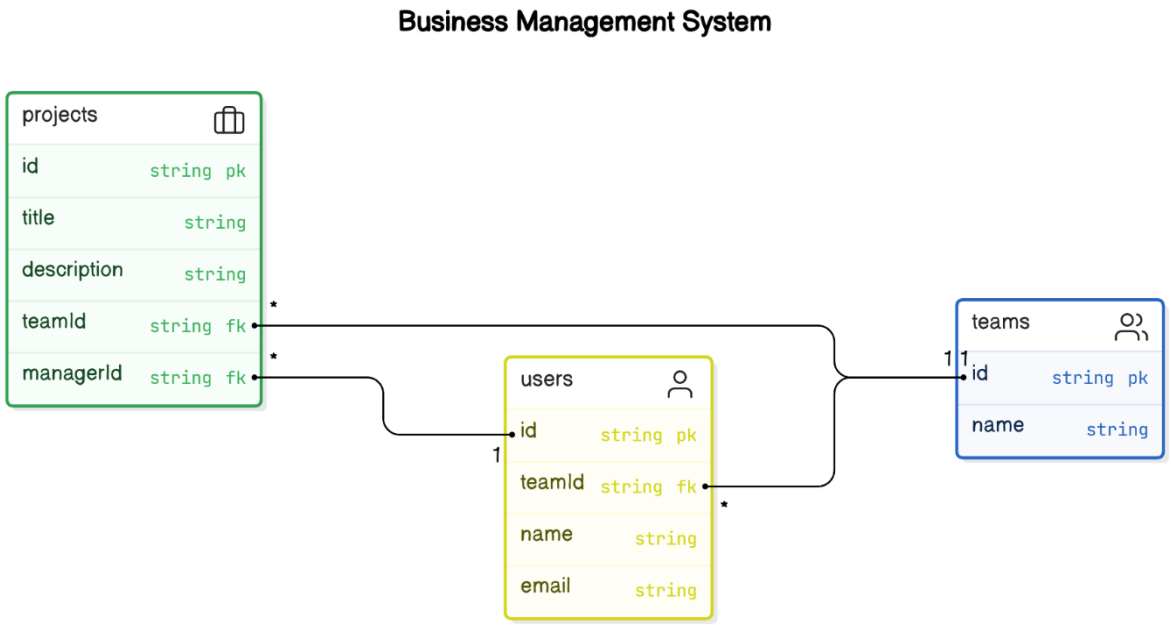
Digvijay Thakare

Day 8 Assignment

Assignment 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-Lets take one example of business management system and following is the Er diagram for that.



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 to establish relationships between tables.

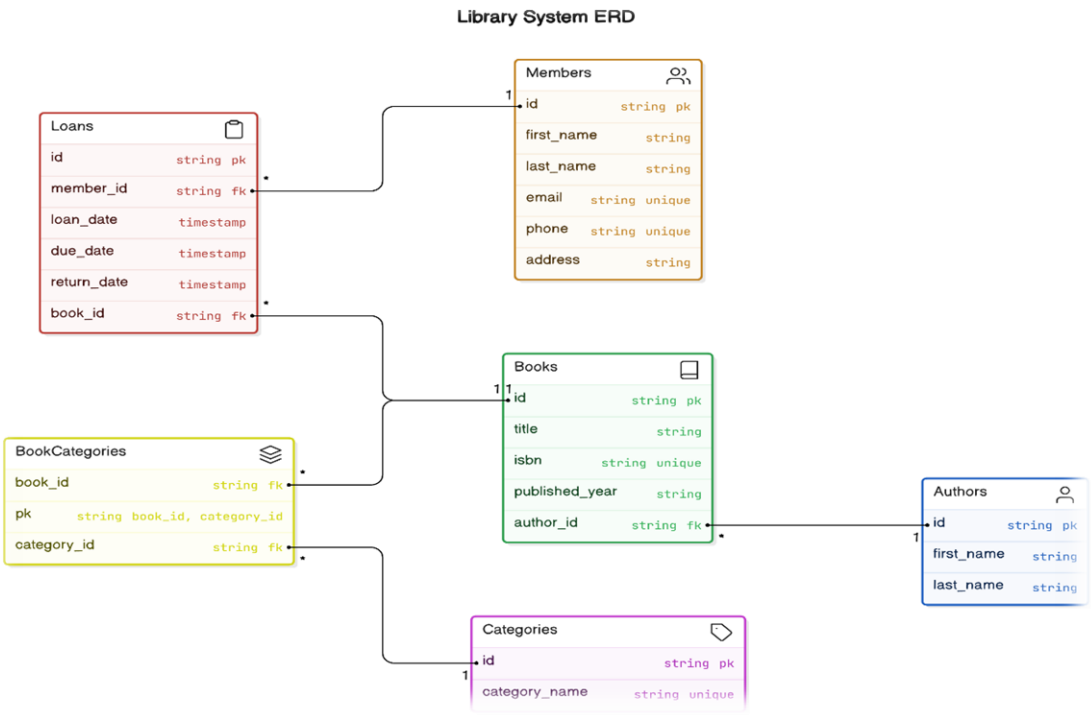
Solution-

Tables and Fields

1. **Authors**
    - **author_id**: a unique number for each author (Primary Key)
    - **first_name**: the author's first name
    - **last_name**: the author's last name
2. **Books**
    - **book_id**: a unique number for each book (Primary Key)
    - **title**: the title of the book
    - **isbn**: a unique code for each book (ISBN)
    - **published_year**: the year the book was published
    - **author_id**: the ID of the author who wrote the book (Foreign Key from Authors)
3. **Members**
    - **member_id**: a unique number for each member (Primary Key)
    - **first_name**: the member's first name
    - **last_name**: the member's last name
    - **email**: the member's email address (must be unique)
    - **phone**: the member's phone number (must be unique)
    - **address**: the member's address
4. **Loans**
    - **loan_id**: a unique number for each loan (Primary Key)
    - **member_id**: the ID of the member who borrowed the book (Foreign Key from Members)
    - **book_id**: the ID of the borrowed book (Foreign Key from Books)
    - **loan_date**: the date the book was borrowed
    - **due_date**: the date the book is due back

- **return_date**: the date the book was returned (if it has been returned)

5. **Categories**
    - **category_id**: a unique number for each category (Primary Key)
    - **category_name**: the name of the category (must be unique)
6. **BookCategories**
    - **book_id**: the ID of the book (Foreign Key from Books)
    - **category_id**: the ID of the category (Foreign Key from Categories)
    - **Primary Key**: combination of book_id and category_id (so a book can belong to multiple categories



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.

Solution-

1. **Atomicity:** This ensures that all operations within a transaction are treated as a single unit. Either all of them are executed or none at all. It's like saying, "Do everything or do nothing."
2. **Consistency:** Transactions must leave the database in a consistent state. If the database was consistent before the transaction, it must remain consistent after the transaction as well.
3. **Isolation**: This property ensures that transactions are properly isolated from each other. The changes of one transaction are not visible to other transactions until they are committed. This prevents 'dirty reads' and ensures that transactions appear to be executed in isolation, even if they run concurrently.
4. **Durability:** Once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors. This is often ensured by writing the transaction logs to non-volatile memory.

   Transaction with SQL statements that include locking:

```
START TRANSACTION;

SELECT * FROM accounts WHERE account_id = 1 FOR UPDATE;

UPDATE accounts SET balance = balance - 100 WHERE account_id = 1;

UPDATE accounts SET balance = balance + 100 WHERE account_id = 2;

COMMIT;
```

   In this transaction, we're transferring funds between two accounts. The FOR UPDATE clause locks the selected rows to prevent other transactions from modifying them until the transaction is complete

- **READ UNCOMMITTED**: This level allows a transaction to see uncommitted changes made by other transactions, which can lead to dirty reads.
- **READ COMMITTED**: This level ensures that a transaction can only read data that has been committed by other transactions, preventing dirty reads.
- **REPEATABLE READ**: This level guarantees that if a transaction reads a row, subsequent reads will see the same data, even if other transactions are modifying it.
- **SERIALIZABLE**: This is the strictest level, where transactions are completely isolated from one another, as if they were executed serially.

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.

Solution-

```
CREATE DATABASE IF NOT EXISTS LibraryDB;

USE LibraryDB;

CREATE TABLE Borrower (
    BorrowerID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Email VARCHAR(255)
);
```

```
mysql> desc borrower;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| BorrowerID     | int          | NO   | PRI | NULL    | auto_increment |
| FirstName      | varchar(50)  | YES  |     | NULL    |                |
| LastName       | varchar(50)  | YES  |     | NULL    |                |
| Email          | varchar(255) | YES  |     | NULL    |                |
| MembershipType | varchar(50)  | YES  |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

```
CREATE TABLE Librarian (
    LibrarianID INT PRIMARY KEY AUTO_INCREMENT
);
```

```
mysql> desc librarian;
+-------------+------+------+-----+---------+----------------+
| Field       | Type | Null | Key | Default | Extra          |
+-------------+------+------+-----+---------+----------------+
| LibrarianID | int  | NO   | PRI | NULL    | auto_increment |
+-------------+------+------+-----+---------+----------------+
1 row in set (0.00 sec)
```

```
CREATE TABLE Book (
    BookID INT PRIMARY KEY AUTO_INCREMENT,
    Title VARCHAR(255),
    Author VARCHAR(255),
    ISBN VARCHAR(13)
);
```

```
mysql> desc book;
+--------+-------------+------+-----+---------+----------------+
| Field  | Type        | Null | Key | Default | Extra          |
+--------+-------------+------+-----+---------+----------------+
| BookID | int         | NO   | PRI | NULL    | auto_increment |
| Title  | varchar(255) | YES  |     | NULL    |                |
| Author | varchar(255) | YES  |     | NULL    |                |
| ISBN   | varchar(13)  | YES  |     | NULL    |                |
+--------+-------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

```
CREATE TABLE Loan (
    LoanID INT PRIMARY KEY AUTO_INCREMENT,
    BorrowerID INT,
    BookID INT,
    LibrarianID INT,
    LoanDate DATE,
    ReturnDate DATE,
    FOREIGN KEY (BorrowerID) REFERENCES Borrower(BorrowerID),
    FOREIGN KEY (BookID) REFERENCES Book(BookID),
    FOREIGN KEY (LibrarianID) REFERENCES Librarian(LibrarianID)
);
```

```
mysql> desc loan;
+--------------+------+------+-----+---------+----------------+
| Field        | Type | Null | Key | Default | Extra          |
+--------------+------+------+-----+---------+----------------+
| LoanID       | int  | NO   | PRI | NULL    | auto_increment |
| BorrowerID   | int  | YES  | MUL | NULL    |                |
| BookID       | int  | YES  | MUL | NULL    |                |
| LibrarianID  | int  | YES  | MUL | NULL    |                |
| LoanDate     | date | YES  |     | NULL    |                |
| ReturnDate   | date | YES  |     | NULL    |                |
+--------------+------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

ALTER TABLE Borrower

ADD COLUMN MembershipType VARCHAR(50);

DROP TABLE IF EXISTS Publisher;

```
mysql> desc borrower;
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
+----------------+-------------+------+-----+---------+----------------+
| BorrowerID     | int         | NO   | PRI | NULL    | auto_increment |
| FirstName      | varchar(50) | YES  |     | NULL    |                |
| LastName       | varchar(50) | YES  |     | NULL    |                |
| Email          | varchar(255)| YES  |     | NULL    |                |
| MembershipType | varchar(50) | YES  |     | NULL    |                |
+----------------+-------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

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 analyze the impact on query execution.

Creating table Book

CREATE TABLE Book (

BookID INT PRIMARY KEY,

Title VARCHAR(255),

Author VARCHAR(255),

```
    ISBN VARCHAR(13)
);
```

Index Creation

```
CREATE INDEX idx_title ON Book (Title);
```

Drop Index

```
DROP INDEX IF EXISTS idx_title ON Book;
```

How Index improves query performance

- Faster Data Retrieval: When a query filters, sorts, or joins data based on the indexed column(s), the database engine can use the index to locate the relevant rows more efficiently. Instead of scanning the entire table, it can perform an index seek or scan, which is generally faster.
- Reduced Disk I/O: Indexes store a sorted copy of the indexed column(s), which reduces the amount of disk I/O required for query processing. This is particularly beneficial for large tables, as it minimizes the need to read data from disk.
- Optimized Sorting and Join Operations: Indexes can improve the performance of sorting and join operations by providing an ordered sequence of values. This can lead to fewer disk accesses and CPU cycles required to process the query.

Impact on query execution after DROP INDEX

After dropping the index, queries that relied on the index for efficient data retrieval may experience degraded performance. The database engine may need to resort to full table scans or other less efficient access methods, which can lead to slower query execution times, especially for queries involving filtering, sorting, or joining based on the "Title" column.

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.

Create a new database user with specific privileges using the CREATE USER and GRANT commands:

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

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

```
mysql> GRANT SELECT, INSERT, UPDATE ON testdb.* TO 'newuser'@'localhost';
Query OK, 0 rows affected (0.02 sec)

mysql>
```

script to REVOKE certain privileges:

REVOKE INSERT ON your_database.* FROM 'newuser'@'localhost';

```
mysql> REVOKE INSERT ON testdb.* FROM 'newuser'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

Drop the user:

DROP USER 'newuser'@'localhost';

```
mysql> DROP USER 'newuser'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

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.

Solution-

INSERT INTO Borrower (BorrowerID, FirstName, LastName, Email)

VALUES (1, 'John', 'Doe', 'john.doe@example.com');

INSERT INTO Book (BookID, Title, Author, ISBN)

VALUES (1, 'The Great Gatsby', 'F. Scott Fitzgerald', '9780743273565');

INSERT INTO Loan (LoanID, BorrowerID, BookID, LibrarianID, LoanDate, ReturnDate)

VALUES (1, 1, 1, 1, '2024-05-21', NULL);

UPDATE existing records with new information:

UPDATE Borrower

SET Email = 'johndoe@example.com'

WHERE BorrowerID = 1;

DELETE records based on specific criteria:

DELETE FROM Loan

WHERE ReturnDate IS NULL

BULK INSERT operations to load data from an external source: