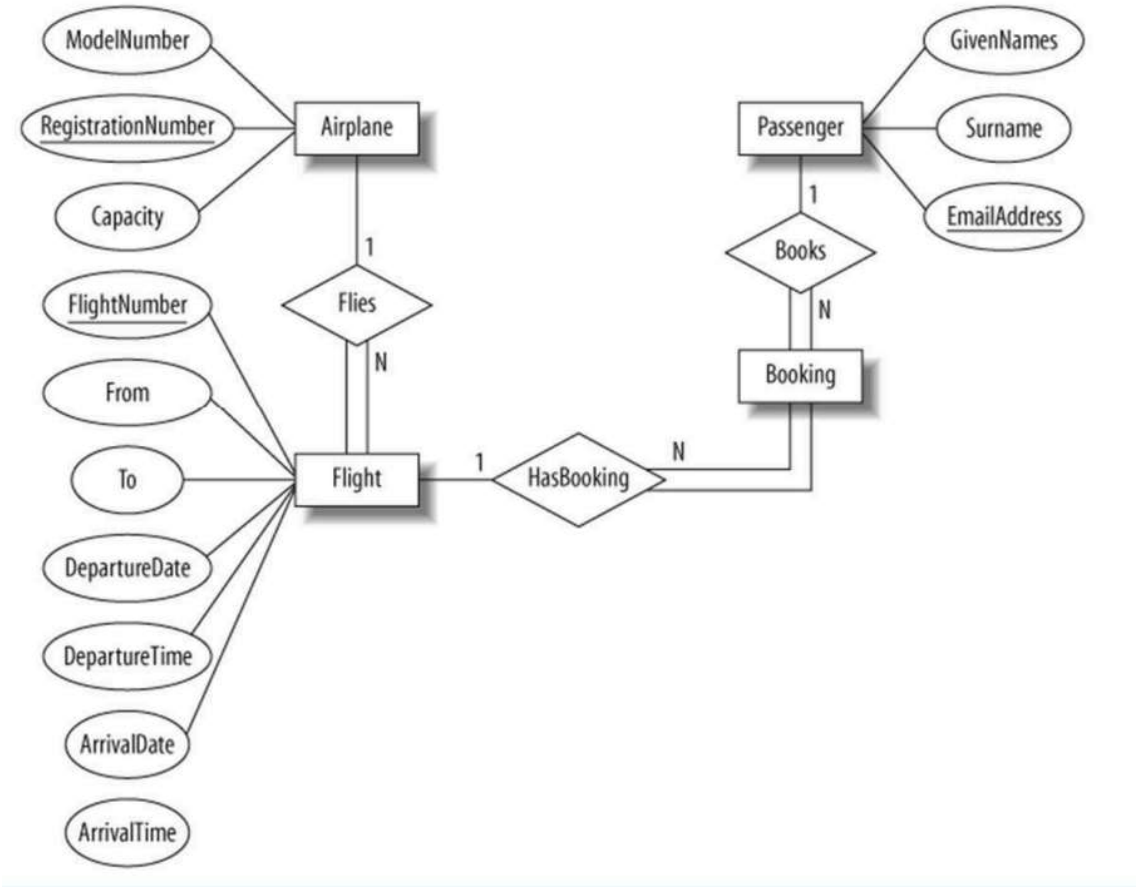
Assignment 1: Analyse a given business scenario and create an ER diagram that includes entities, relationships, attributes, and cardinality. Ensure that the diagram reflects proper normalization up to the third normal form.

Answer:



Flight Database

Assignment 2: QUES 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.

```
create database libsystem;

use libsystem;
CREATE TABLE Users (
    UserID INT NOT NULL AUTO_INCREMENT,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Email VARCHAR(100) NOT NULL UNIQUE,
    PhoneNumber VARCHAR(15),
    MembershipDate DATE NOT NULL ,
    PRIMARY KEY (UserID)
);
use libsystem;
CREATE TABLE Authors (
    AuthorID INT NOT NULL AUTO_INCREMENT,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    BirthDate DATE,
    Nationality VARCHAR(50),
    PRIMARY KEY (AuthorID)
);
use libsystem;
CREATE TABLE Books (
    BookID INT NOT NULL AUTO_INCREMENT,
    Title VARCHAR(255) NOT NULL,
    ISBN VARCHAR(13) NOT NULL UNIQUE,
    PublicationYear YEAR NOT NULL,
```

```
    Genre VARCHAR(50) NOT NULL,
    CopiesAvailable INT NOT NULL CHECK (CopiesAvailable >= 0),
    PRIMARY KEY (BookID)
);
use libsystem;
CREATE TABLE BookAuthors (
    BookID INT NOT NULL,
    AuthorID INT NOT NULL,
    PRIMARY KEY (BookID, AuthorID),
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (AuthorID) REFERENCES Authors(AuthorID)
);
```

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.

Answer: ACID stands for Atomicity, Consistency, Isolation, and Durability, which are essential properties for ensuring data integrity in database transactions.

Atomicity: Imagine a transaction as a single unit of work. Either all the operations within the transaction succeed, or none of them do. This prevents partial updates or inconsistencies in the data. It's like flipping a coin - it lands heads or tails, but it can't be both at the same time.
Consistency: A transaction transforms the database from one valid state (following defined rules and constraints) to another. It ensures the data remains consistent before and after the transaction completes. Think of this as moving money between accounts - the total amount in the system stays the same, even though it's distributed differently.
Isolation: Concurrent transactions executing at the same time are isolated from each other. This prevents conflicts and guarantees predictable outcomes, even if multiple users are modifying the same data. Imagine two people editing the same document - locking ensures only one person can make changes at a time, preventing conflicts.
Durability: Once a transaction commits (completes successfully), the changes are permanently persisted in the database storage. Even in case of system failures, the committed data remains intact. Think of writing on a piece of paper - even if the pen runs out or you make a mistake, the information written is permanent.

```
create database bank;
use bank;
CREATE TABLE account (
  account_number INT PRIMARY KEY,
```

```
  balance INT NOT NULL
);
USE BANK;
START TRANSACTION;  -- Begin the transaction

-- Lock the accounts to prevent conflicts (pessimistic locking)
SELECT * FROM account WHERE account_number = 620459495493 FOR
UPDATE;
SELECT * FROM account WHERE account_number = 988012345215 FOR
UPDATE;

-- Update balances (assuming sufficient funds)
UPDATE account SET balance = balance - 5000 WHERE account_number =
620459495493;
UPDATE account SET balance = balance + 5000 WHERE account_number =
988012345215;

COMMIT;  -- Commit the transaction if successful
```

Isolation Levels:

MySQL uses transaction isolation levels defined in the SET TRANSACTION ISOLATION LEVEL statement. Here's how different levels affect our scenario:

- **READ UNCOMMITTED (RU):** Transactions can see uncommitted changes from other transactions. This can lead to "dirty reads" where a transaction reads data that might be rolled back later.

Scenario: Transaction A starts reading FROM _ACCOUNT_NUMBER while Transaction B is updating it (but hasn't committed yet). A might read an inconsistent balance before B commits.

- **READ COMMITTED (RC):** Transactions can only see committed changes from other transactions. This ensures consistency but might introduce a slight delay.

Scenario: Transaction A waits until Transaction B commits before reading the updated balance in FROM_ACCOUNT_NUMBER, ensuring consistent data.

- **REPEATABLE READ (RR):** Transactions can see a consistent snapshot of the database at the start of the transaction, preventing "dirty reads" and "non-repeatable reads" (where the same data is read twice with different results due to concurrent modifications).

Scenario: Transaction A reads FROM_ACCOUNT_NUMBER, and even if another transaction modifies it later, A will always see the initial value it read, ensuring consistent results.

- **SERIALIZABLE (SERIAL):** Transactions are executed one at a time, ensuring the highest level of consistency but sacrificing concurrency.

Scenario: Transactions are queued, and only one executes at a time. This avoids all potential conflicts but can significantly impact performance.

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.

```
CREATE DATABASE librarydb;

USE librarydb;

CREATE TABLE Users (

    UserID INT NOT NULL AUTO_INCREMENT,

    FirstName VARCHAR(50) NOT NULL,

    LastName VARCHAR(50) NOT NULL,

    Email VARCHAR(100) NOT NULL UNIQUE,

    PhoneNumber VARCHAR(15),

    MembershipDate DATE NOT NULL ,

    PRIMARY KEY (UserID)

);

use librarydb;

CREATE TABLE Authors (

    AuthorID INT NOT NULL AUTO_INCREMENT,

    FirstName VARCHAR(50) NOT NULL,

    LastName VARCHAR(50) NOT NULL,
```

```
    BirthDate DATE,
    Nationality VARCHAR(50),
    PRIMARY KEY (AuthorID)
);
use librarydb;
CREATE TABLE Books (
    BookID INT NOT NULL AUTO_INCREMENT,
    Title VARCHAR(255) NOT NULL,
    ISBN VARCHAR(13) NOT NULL UNIQUE,
    PublicationYear YEAR NOT NULL,
    CopiesAvailable INT NOT NULL CHECK (CopiesAvailable >= 0),
    PRIMARY KEY (BookID)
);
use librarydb;
CREATE TABLE BookAuthors (
    BookID INT NOT NULL,
    AuthorID INT NOT NULL,
    PRIMARY KEY (BookID, AuthorID),
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (AuthorID) REFERENCES Authors(AuthorID)
);
use librarydb;
CREATE TABLE Loans (
    LoanID INT NOT NULL AUTO_INCREMENT,
    UserID INT NOT NULL,
```

```
    BookID INT NOT NULL,
    LoanDate DATE NOT NULL,
    DueDate DATE NOT NULL,
    ReturnDate DATE,
    PRIMARY KEY (LoanID),
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    CONSTRAINT chk_due_date CHECK (DueDate > LoanDate)
);
```

-- modifying table structure

```
ALTER TABLE users
ADD address varchar(50);

ALTER TABLE Books
ADD COLUMN genre VARCHAR(255);
```

-- Removing a Redundant Table(Assuming it it no longer necessary)

```
DROP TABLE bookauthors;
```

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 impact on query execution.

Answer: Creating table customers if not exists

```
CREATE TABLE IF NOT EXISTS customers (
  customer_id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL UNIQUE,
  city VARCHAR(50) NOT NULL
);
```

-- Create an Index on :city

```
CREATE INDEX idx_city ON customers(city);
```

Indexes significantly improve query performance when searching or filtering data based on the indexed column(s). They speed up retrieval by providing a faster way for the database to locate relevant rows. However, indexes come with some overhead in terms of storage space and maintenance (updating the index when the table data changes).

-- Analyse query performance(Without Index)

```
EXPLAIN SELECT * FROM customers WHERE city = 'New York';
```

Running this query will display the execution plan, including details like the table access method used. If no index exists, a full table scan might be used, meaning the database engine needs to examine all rows in the customers table to find those with city = ‘New York’.

--Analyse Query Performance (With Index)

```
EXPLAIN SELECT * FROM customers WHERE city = 'New York';
```

After running the same EXPLAIN statement after creating the index. We should see a more efficient access method used, potentially an index scan on idx_city.This suggests the database can leverage the index to locate rows with city = ‘New York’ much faster than a full table scan.

-- Drop the Index

```
DROP INDEX idx_city ON customers;
```

The impact on query execution.

Dropping the index essentially removes the optimized access path that the database engine could use to find data based on the city column.This leads to slower query execution times and potentially impacts the performance of other queries that might benefit from the index indirectly (e.g., joins involving the city 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.

Answer: Create a new database user

```
CREATE USER 'new_user'@'localhost' IDENTIFIED BY 'password';
```

Grant specific privileges to the user on a specific database

```
GRANT SELECT, INSERT, UPDATE ON librarydb.* TO 'new_user'@'localhost';
```

Revoke certain privileges from the user

```
REVOKE INSERT ON librarydb.* FROM 'new_user'@'localhost';
```

Drop the user

```
DROP USER 'new_user'@'localhost';
```

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."

CREATE DATABASE librarydb;

```
USE librarydb;

CREATE TABLE Users (
    UserID INT NOT NULL AUTO_INCREMENT,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
```

```
    Email VARCHAR(100) NOT NULL UNIQUE,
    PhoneNumber VARCHAR(15),
    MembershipDate DATE NOT NULL ,
    PRIMARY KEY (UserID)
);
use librarydb;
CREATE TABLE Authors (
    AuthorID INT NOT NULL AUTO_INCREMENT,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    BirthDate DATE,
    Nationality VARCHAR(50),
    PRIMARY KEY (AuthorID)
);
use librarydb;
CREATE TABLE Books (
    BookID INT NOT NULL AUTO_INCREMENT,
    Title VARCHAR(255) NOT NULL,
    ISBN VARCHAR(13) NOT NULL UNIQUE,
    PublicationYear YEAR NOT NULL,
    CopiesAvailable INT NOT NULL CHECK (CopiesAvailable >= 0),
    PRIMARY KEY (BookID)
);
use librarydb;
CREATE TABLE BookAuthors (
```

```
    BookID INT NOT NULL,
    AuthorID INT NOT NULL,
    PRIMARY KEY (BookID, AuthorID),
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (AuthorID) REFERENCES Authors(AuthorID)
);
use librarydb;
CREATE TABLE Loans (
    LoanID INT NOT NULL AUTO_INCREMENT,
    UserID INT NOT NULL,
    BookID INT NOT NULL,
    LoanDate DATE NOT NULL,
    DueDate DATE NOT NULL,
    ReturnDate DATE,
    PRIMARY KEY (LoanID),
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    CONSTRAINT chk_due_date CHECK (DueDate > LoanDate)
);
```

-- modifying table structure

```
ALTER TABLE users
ADD address varchar(50);
```

```
ALTER TABLE Books

ADD COLUMN genre VARCHAR(255);
```

-- Removing a Redundant Table(Assuming it it no longer necessary)

```
DROP TABLE bookauthors;
```

-- Inserting a new user

```
INSERT INTO Users (FirstName, LastName, Email, PhoneNumber,
MembershipDate)

VALUES ('John', 'Doe', 'johndoe@example.com', '+123456789', '2024-05-22');
```

-- Inserting a new book

```
INSERT INTO Books (Title, ISBN, PublicationYear, CopiesAvailable)

VALUES ('The Great Gatsby', '9780743273565', 1925, 5);
```

-- Update user's phone number

```
UPDATE Users

SET PhoneNumber = '+987654321'

WHERE UserID = 1;
```

-- Update book's publication year

```
UPDATE Books
```

```
SET PublicationYear = 1926
WHERE Title = 'The Great Gatsby';

set SQL_SAFE_UPDATES =0;

-- Delete a user based on email
DELETE FROM Users
WHERE Email = 'kumarisonal@example.com';

-- Delete a book based on ISBN
DELETE FROM Books
WHERE ISBN = '9780743273565';

-- Create a temporary table to hold the bulk data
CREATE TEMPORARY TABLE temp_books (
    Title VARCHAR(255),
    ISBN VARCHAR(13),
    PublicationYear YEAR,
    CopiesAvailable INT
);

-- Bulk insert data from CSV file into the temporary table

LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\'
INTO TABLE Books
```

```
FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 LINES

(Title, ISBN, PublicationYear, CopiesAvailable);


SHOW VARIABLES LIKE 'secure_file_priv';
```

Here , i am using a csv file contains the record

```
-- Insert data from the temporary table into the main table

INSERT INTO Books (Title, ISBN, PublicationYear, CopiesAvailable)

SELECT Title, ISBN, PublicationYear, CopiesAvailable

FROM temp_books;


-- Drop the temporary table

DROP TABLE temp_books;
```