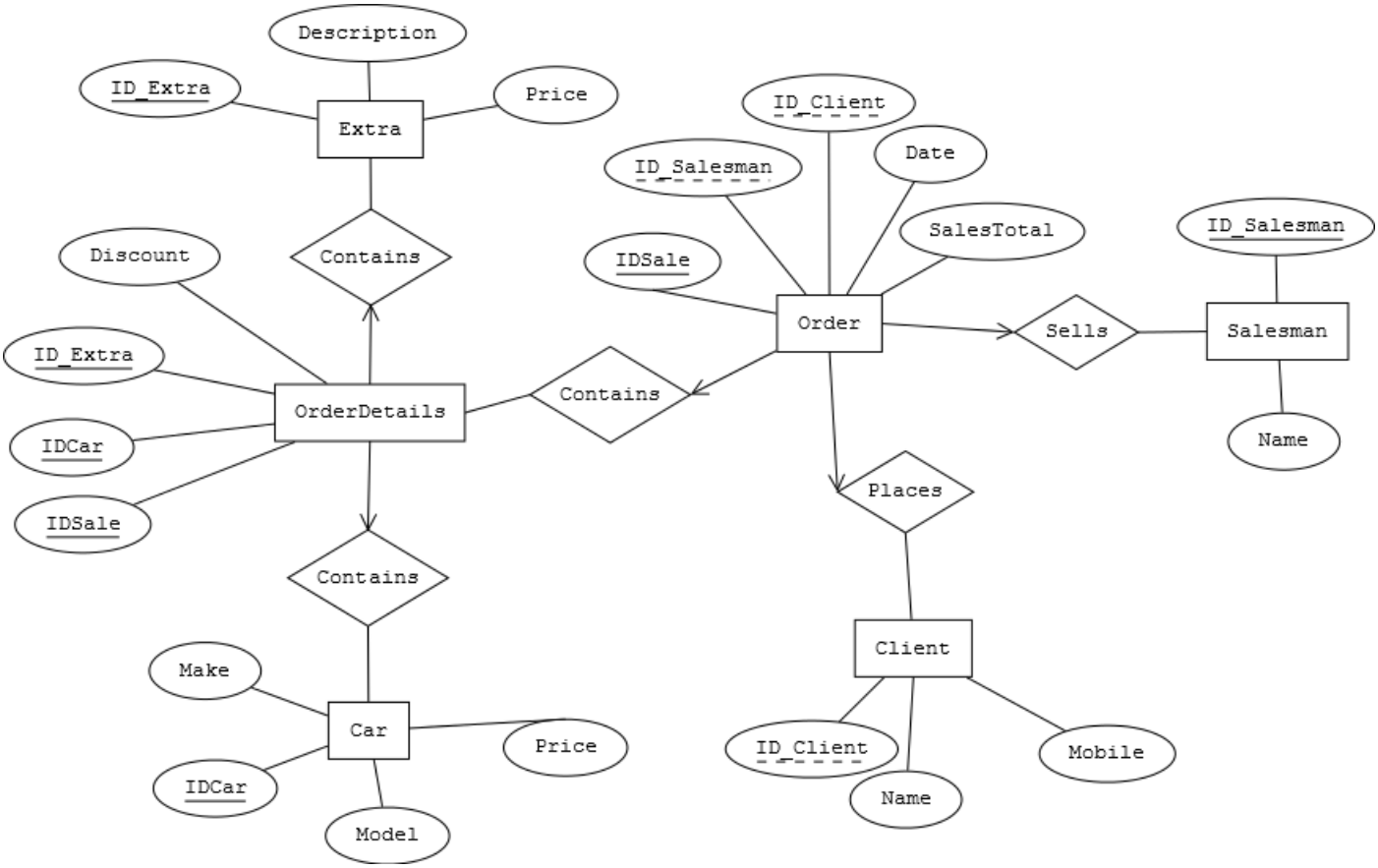
hariharane01@gmail.com

SQL-Assignment 01

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.

Answer:



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.

Answer:

Authors:

```
CREATE TABLE Authors (
    author_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);
```

Books:

```
CREATE TABLE Books (
    book_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    author_id INT,
    isbn VARCHAR(13) UNIQUE NOT NULL,
    FOREIGN KEY (author_id) REFERENCES Authors(author_id)
);
```

Members:

```
CREATE TABLE Members (
    member_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL
);
```

Loans:

```
CREATE TABLE Loans (
    loan_id INT AUTO_INCREMENT PRIMARY KEY,
    book_id INT,
    member_id INT,
    loan_date DATE NOT NULL,
    due_date DATE NOT NULL,
    FOREIGN KEY (book_id) REFERENCES Books(book_id),
    FOREIGN KEY (member_id) REFERENCES Members(member_id)
);
```

Explanation of Constraints

- **Primary Key (PRIMARY KEY):** Uniquely identifies each record in the table.
- **Foreign Key (FOREIGN KEY**): Establishes relationships between tables.
- **NOT NULL:** Ensures that a field cannot be left empty.
- **UNIQUE:** Ensures all values in a column are distinct.
- **CHECK:** Adds a condition that each row must satisfy.

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:

Atomicity: This property ensures that a transaction is treated as a single unit of work. It means that either all the operations within the transaction are successfully completed and committed to the database, or none of them are. There's no halfway point; it's all or nothing.

Consistency: This property ensures that the database remains in a consistent state before and after the transaction. If the database is in a valid state before the transaction begins, it should remain in a valid state after the transaction is completed, regardless of any failures or errors that may occur during the transaction.

Isolation: This property ensures that the execution of transactions concurrently does not result in any interference or inconsistency. Each transaction should operate independently of other transactions, as if it is the only transaction running on the database. Isolation prevents transactions from seeing each other's intermediate states, thus maintaining data integrity.

Durability: This property ensures that once a transaction is committed, the changes it has made to the database persist even in the event of system failures or crashes. Committed data should be stored permanently and should not be lost, ensuring that the database remains reliable.

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 a 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;

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;
```

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.

Answer:

- **Create a new database as LibraryManagement:**

```
CREATE DATABASE LibraryManagement;
USE LibraryManagement;
```

- **Create tables according to the library schema:**

❖ **Create Authors table**

```
CREATE TABLE Authors (
    AuthorID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(255) NOT NULL,
    LastName VARCHAR(255) NOT NULL,
    BirthYear INT
);
```

-

❖ **Create Publishers table**

```
CREATE TABLE Publishers (
    PublisherID INT PRIMARY KEY AUTO_INCREMENT,
    PublisherName VARCHAR(255) NOT NULL,
    Country VARCHAR(100)
);
```

❖ **Create Books table**

```
CREATE TABLE Books (
    BookID INT PRIMARY KEY AUTO_INCREMENT,
    Title VARCHAR(255) NOT NULL,
    AuthorID INT,
    PublisherID INT,
    YearPublished INT,
    Genre VARCHAR(100),
    FOREIGN KEY (AuthorID) REFERENCES Authors(AuthorID),
    FOREIGN KEY (PublisherID) REFERENCES Publishers(PublisherID)
);
```

❖ **Create Members table**

```
CREATE TABLE Members (
    MemberID INT PRIMARY KEY AUTO_INCREMENT,
```

```
    FullName VARCHAR(255) NOT NULL,
    MembershipDate DATE
);
```

- Create Loans table

```
CREATE TABLE Loans (
    LoanID INT PRIMARY KEY AUTO_INCREMENT,
    BookID INT,
    MemberID INT,
    LoanDate DATE,
    ReturnDate DATE,
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID)
);
```

- Adding Email as new column in member table

```
ALTER TABLE Members
ADD COLUMN Email VARCHAR(255);

-- Add ISBN column to Books table
ALTER TABLE Books
ADD COLUMN ISBN VARCHAR(13);
```

- Drop redundant table OldPublishers

```
DROP TABLE OldPublishers;
```

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.

Answer:

- Create table as student:

```
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    name VARCHAR(100),
    age INT,
    grade VARCHAR(2)
);

INSERT INTO students (student_id, name, age, grade) VALUES
(1, 'John Doe', 18, 'A'),
(2, 'Jane Smith', 17, 'B'),
```

(3, 'Michael Johnson', 19, 'C'),
(4, 'Emily Davis', 18, 'A'),
(5, 'Chris Wilson', 17, 'B'),
(6, 'Sarah Brown', 19, 'A'),
(7, 'Alex Lee', 18, 'C'),
(8, 'Emma Taylor', 17, 'B'),
(9, 'Ryan Moore', 19, 'A'),
(10, 'Olivia Anderson', 18, 'B');

- Create an index
  We create an index on the grade column to improve the performance of the queries

  CREATE INDEX idx_grade ON students (grade);

- **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 students WHERE grade = 'A';

- **Drop the index:**
  Now, we drop the index and analyze the performance impact.

  DROP INDEX idx_grade ON students;
- **Analyze Query Performance with Index:**
  Run the same query again and use EXPLAIN to see the difference.

  EXPLAIN SELECT * FROM students IGNORE INDEX (idx_grade) WHERE grade = 'A';

Summary of Performance Impact

With Index: The query execution plan utilizes the index idx_grade, enabling the database to efficiently locate rows matching the query condition based on the 'grade' column. This results in faster query execution, especially for large datasets, as the database can avoid a full table scan.

Without Index: The query execution plan involves a full table scan, where the database must examine every row in the 'students' table to identify those with a grade of 'A'. This approach is slower compared to using an index, as it requires reading each row individually to determine if it meets the query condition.

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 new user:**

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

- **Revoke certain privileges:**

```
REVOKE INSERT, UPDATE ON database_name.* FROM
'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.

Answer:

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 new records into the library tables:**

Inserting a new book record

```
INSERT INTO Books (Title, Author, Publication_Year, ISBN)
VALUES ('To Kill a Mockingbird', 'Harper Lee', 1960, '9780061120084');
```

Inserting a new member record

```
INSERT INTO Members (Name, Email, Phone)
VALUES ('John Smith', 'john@example.com', '123-456-7890');
```

- **UPDATE existing records with new information:**

Updating book information

```
UPDATE Books
SET Publication_Year = 1962
```

```
WHERE Title = 'To Kill a Mockingbird';
```

Updating member information

```
UPDATE Members
SET Phone = '987-654-3210'
WHERE Name = 'John Smith';
```

- **DELETE records based on specific criteria:**

Deleting a book record

```
DELETE FROM Books
WHERE Title = 'To Kill a Mockingbird';
```

Deleting a member record

```
DELETE FROM Members
WHERE Name = 'John Smith';
```

To perform bulk insert operations from an external source (e.g., a CSV file), you can use the LOAD DATA INFILE statement in MySQL.

- **Bulk Insert Book**

```
BULK INSERT Books
FROM 'C:\path\to\books.csv'
WITH (
   FIELDTERMINATOR = ',',
   ROWTERMINATOR = '\n',
   FIRSTROW = 2 -- Skip header row if present
);
```

- **Example file format**

```
Title,Author,Publication_Year,ISBN
To Kill a Mockingbird,Harper Lee,1960,9780061120084
1984,George Orwell,1949,9780451524935
Pride and Prejudice,Jane Austen,1813,9780141439518
The Great Gatsby,F. Scott Fitzgerald,1925,9780743273565
```