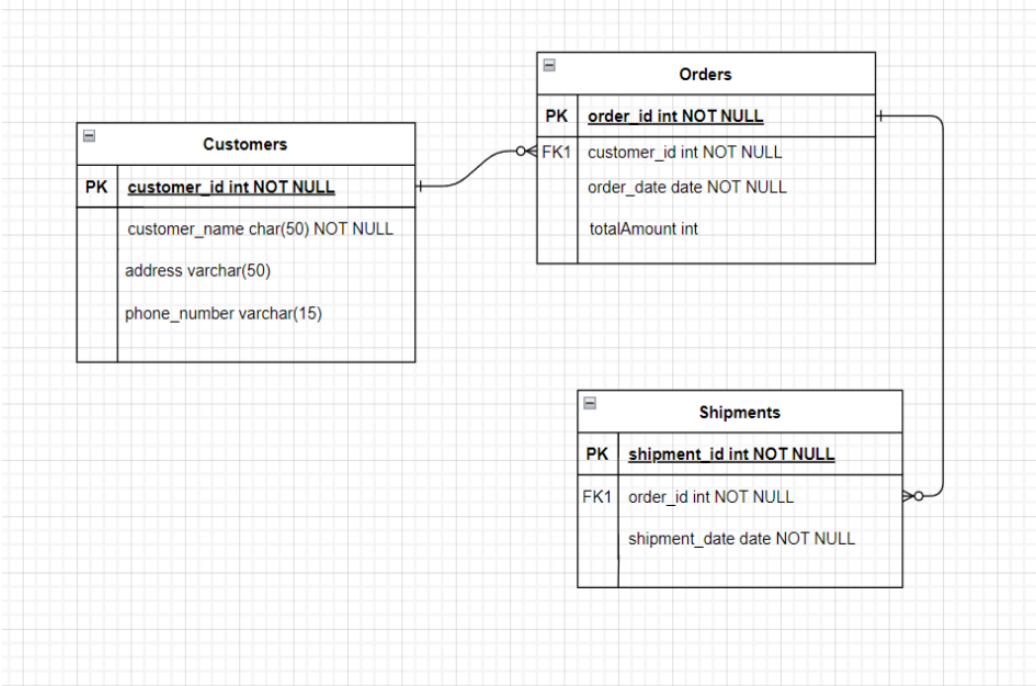
DBMS And SQL

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.

Business Scenario:

An online store desires to create a database system to manage customer information, orders made by customers, and the shipping of those orders. The system needs to store information about customers, the orders they make and the shipping of each order. A product can be in multiple orders and an order can have multiple products delivered by a shipment.



Relationships:

Customers to Orders: One-to-Many (One customer can place many orders, but each order is placed by one customer).

Order to Shipments: One-to-Many (One order can have many shipments, but each shipment is placed by one order).

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.

Database Schema design for a Library System

The database schema for a library system includes three main entities: Books, Borrowers, and Loans. Each entity is represented by a table in the database, with specific fields, constraints, and keys.

DBMS And SQL

Tables:

1. Books

- **BookID (Primary Key, NOT NULL, UNIQUE):** Unique identifier for each book.
- **Title:** Book title.
- **AuthorID (Foreign Key, references Authors.AuthorID):** The author of the book**.**
- **PublicationDate**: Date of publication**.**
- **ISBN:** International Standard Book Number**.**

2. Borrowers

- **BorrowerID (Primary Key, NOT NULL, UNIQUE):** Unique identifier for each borrower.
- **Name:** Borrower's name.
- **Email:** Borrower's email.
- **Phone:** Borrower's phone number.

3. Loans

- **LoanID (Primary Key, NOT NULL, UNIQUE)**: Unique identifier for each loan.
- **BookID (Foreign Key, references Books.BookID):** The book being borrowed.
- **BorrowerID (Foreign Key, references Borrowers.BorrowerID)**: The borrower.
- **LoanDate:** Date the book was borrowed.
- **ReturnDate:** Date the book is due to be returned.
- **Returned:** A boolean indicating whether the book has been returned.

Constraints:

1. Books

- **BookID** is a NOT NULL and UNIQUE constraint to ensure each book has a unique identifier.
- **AuthorID** is a foreign key referencing Authors.AuthorID to establish the relationship between authors and books.
- **PublicationDate** is a CHECK constraint to ensure the date is in the format YYYY-MM-DD.
- **ISBN** is a CHECK constraint to ensure the ISBN is in the format XXXX-XXXX-XXXX-XXXX.

2. Borrowers

- **BorrowerID** is a NOT NULL and UNIQUE constraint to ensure each borrower has a unique identifier.
- **Name, Email** and **Phone** are NOT NULL constraints to ensure these fields are always filled.

3. Loans

- **LoanID** is a NOT NULL and UNIQUE constraint to ensure each loan has a unique identifier.
- **BookID** is a foreign key referencing Books.BookID to establish the relationship between books and loans.
- **BorrowerID** is a foreign key referencing Borrowers.BorrowerID to establish the relationship between borrowers and loans.
- **LoanDate and ReturnDate** are CHECK constraints to ensure the dates are in the format YYYY-MM-DD.
- **Returned** is a CHECK constraint to ensure the value is either TRUE or FALSE.

This schema establishes relationships between authors, books, borrowers, loans, and fines. It ensures data integrity by enforcing constraints on each table.

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.

The ACID properties are fundamental principles that ensure the reliability and integrity of transactions within a database system.

ACID Properties

1. **Atomicity:** Ensures that a transaction is treated as a single unit of work. If any part of the transaction fails, the entire transaction is rolled back to its initial state.

2. **Consistency:** Ensures that the database remains in a consistent state before and after the transaction. It means that any data modifications performed by a transaction must abide by all the constraints, rules, and relationships defined in the database schema.

3. **Isolation:** Ensures that the execution of multiple transactions concurrently does not result in interference or inconsistency. Each transaction should operate independently of other transactions, as if it were the only transaction executing on the database.

4. **Durability:** Ensures that once a transaction is committed, its changes are permanently saved in the database and survive system failures such as crashes or power outages.

Now, for demonstrating different isolation levels and concurrency control in SQL, We'll use SQL statements to showcase different isolation levels.

CREATE TABLE Account (account_id INT PRIMARY KEY, balance DECIMAL(10, 2));

INSERT INTO Account (account_id, balance) VALUES (1, 1000.00);

Transaction1:

START TRANSACTION:

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

SELECT SLEEP(5);

UPDATE Account SET balance = balance + 100 WHERE account_id = 1;

COMMIT;

Transaction 2:

START TRANSACTION:

UPDATE Account SET balance = balance - 50 WHERE account_id = 1;

SELECT SLEEP(5);

UPDATE Account SET balance = balance + 50 WHERE account_id = 1;

COMMIT;

We can set different isolation levels for each transaction using the SET TRANSACTION

ISOLATION LEVEL statement before starting the transaction. For example:

1. READ COMMITTED: Transaction 1 can read and modify committed data, but it will not see uncommitted changes made by Transaction

2. REPEATABLE READ: Transaction 1 will not see changes made by Transaction 2 until it commits.

3. SERIALIZABLE: Transactions are completely isolated from each other. Neither Transaction 1 nor Transaction 2 can see changes made by the other until both transactions 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.

Solution:

SQL statements to create a new database and tables for the library schema:

CREATE DATABASE LibrarySystem;

DBMS And SQL

→Switch to the new database

```
USE LibrarySystem;
```

→ Create the Books table

```
CREATE TABLE Books (
    BookID INT PRIMARY KEY NOT NULL UNIQUE,
    Title VARCHAR(255) NOT NULL,
    AuthorID INT,
    PublicationDate DATE CHECK (PublicationDate >= '1900-01-01'),
    ISBN VARCHAR(20) CHECK (ISBN REGEXP '^[0-9]{4}-[0-9]{4}-[0-9]{4}-[0-9]{4}$'),
    FOREIGN KEY (AuthorID) REFERENCES Authors(AuthorID)
);
```

→ Create the Borrowers table

```
CREATE TABLE Borrowers (
    BorrowerID INT PRIMARY KEY NOT NULL UNIQUE,
    Name VARCHAR(255) NOT NULL,
    Email VARCHAR(255) NOT NULL,
    Phone VARCHAR(20) NOT NULL
);
```

→Create the Loans table

```
CREATE TABLE Loans (
    LoanID INT PRIMARY KEY NOT NULL UNIQUE,
    BookID INT,
    BorrowerID INT,
    LoanDate DATE CHECK (LoanDate >= '1900-01-01'),
    ReturnDate DATE CHECK (ReturnDate >= '1900-01-01'),
    Returned BOOLEAN CHECK (Returned IN (TRUE, FALSE)),
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (BorrowerID) REFERENCES Borrowers(BorrowerID)
);
```

→ Drop the Fines table (redundant)

DROP TABLE Fines;

→Modify the Loans table to include a CHECK constraint for Returned

ALTER TABLE Loans

ADD CHECK (Returned IN (TRUE, FALSE));

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 an Index on the 'department_id' Column

To create an index on the 'department_id' column of the 'employee' table, you can use the following SQL statement:

```
CREATE INDEX department_index ON employees (department_id);
```

Impact of the Index on Query Performance

The index improves query performance in several ways:

Faster Data Retrieval: When you execute a query that involves filtering, sorting, or joining based on the 'department_id' column, the index allows the database to quickly locate the relevant rows. Without the index, the database would have to scan through the entire table to find the matching rows, which can be slower, especially for large tables.

Reduced Disk I/O: With the index in place, the database engine can access the necessary data by reading the index structure rather than the entire table. This reduces the amount of disk I/O required, leading to faster query execution.

Improved Query Plan: The database optimizer can use the index to generate more efficient query execution plans. It can choose index scans or index seeks instead of table scans, which generally require fewer resources and execute faster.

Removing the Index

To remove the index, you can use the following SQL statement:

```
DROP INDEX department_index ON employees;
```

Impact of Removing the Index

Slower Query Performance: Without the index, the database has to scan through the entire table to find the matching rows, which can lead to slower query execution times.

Increased Disk I/O: The database engine now has to read the entire table to access the necessary data, leading to increased disk I/O and slower query execution.

creating an index on the 'department_id' column improves query performance by allowing for faster data retrieval, reduced disk I/O, and improved query plans. Removing the index leads to slower query performance and increased disk I/O due to the need to scan the entire table.

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:-

SQL commands:

```
CREATE USER 'admin’ @ 'localhost ' IDENTIFIED BY '12345';

GRANT SELECT INSERT UPDATE ON USER.* TO 'admin’ @ 'localhost';

FLUSH PRIVILEGES;

REVOKE INSERT ON user.* FROM 'admin’ @ 'localhost ';

DROP USER 'admin’ @ ' localhost'
```

Explanation of the Script

1. The script starts by creating a new user named 'admin' with a password of '12345' on the 'localhost' server.
2. It then grants the user SELECT, INSERT, and UPDATE privileges on all tables in the database.
3. The FLUSH PRIVILEGES command is used to reload the grant tables and enable the changes to take effect immediately.
4. The script then revokes the INSERT privilege from the user.
5. Finally, the script drops the user from the database.

This script demonstrates the creation of a new user, granting privileges, revoking privileges, and dropping the user.

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.

SQL Statements for INSERT, UPDATE, and DELETE Operations

SQL Statements to **insert** into library tables:

```
INSERT INTO Books (BookID, BookName, Genre) VALUES (125, 'The Little Prince', 'Fiction');
```

INSERT INTO Category (CategoryId, CategoryName) VALUES (3, 'Science Fiction');

insert into author (authorId , authorName , birthYear) values (102, 'Antoine de Saint-Exupéry', null);

OUTPUT:

BOOIKID	BOOKNAME	GENRE
125	The Little Prince	Fiction

CATEGORYID	CATEGORYNAME
3	Science Fiction

AUTHORID	AUTHORNAME	BIRTHYEAR
102	Antoine de Saint-Exupéry	

SQL statements to **update** existing records with new information:

update books

set authorName='Robert'

where bookId=125;

update category

set categoryName='Action'

where categoryId = 3;

update author

set birthYear=1914

where authorId=102;

OUTPUT:

BOOIKID	BOOKNAME	GENRE
125	Robert	Fiction

CATEGORYID	CATEGORYNAME
3	Action

AUTHORID	AUTHORNAME	BIRTHYEAR
102	Antoine de Saint-Exupéry	1914

SQL Statements to delete records based on specific criteria:

delete from books where bookId=125;

BULK INSERT Operation:

```
BULK INSERT Books
FROM 'C:\Path\To\Data.csv'
WITH
   (FIRSTROW = 2,
    FIELDS TERMINATED BY ',',
    ENCLOSED BY '\"',
    DATEFORMAT = 'yyyy-mm-dd',
    IGNOREBLANKLINES = 1);
```

The BULK INSERT statement loads data from an external source into the database.