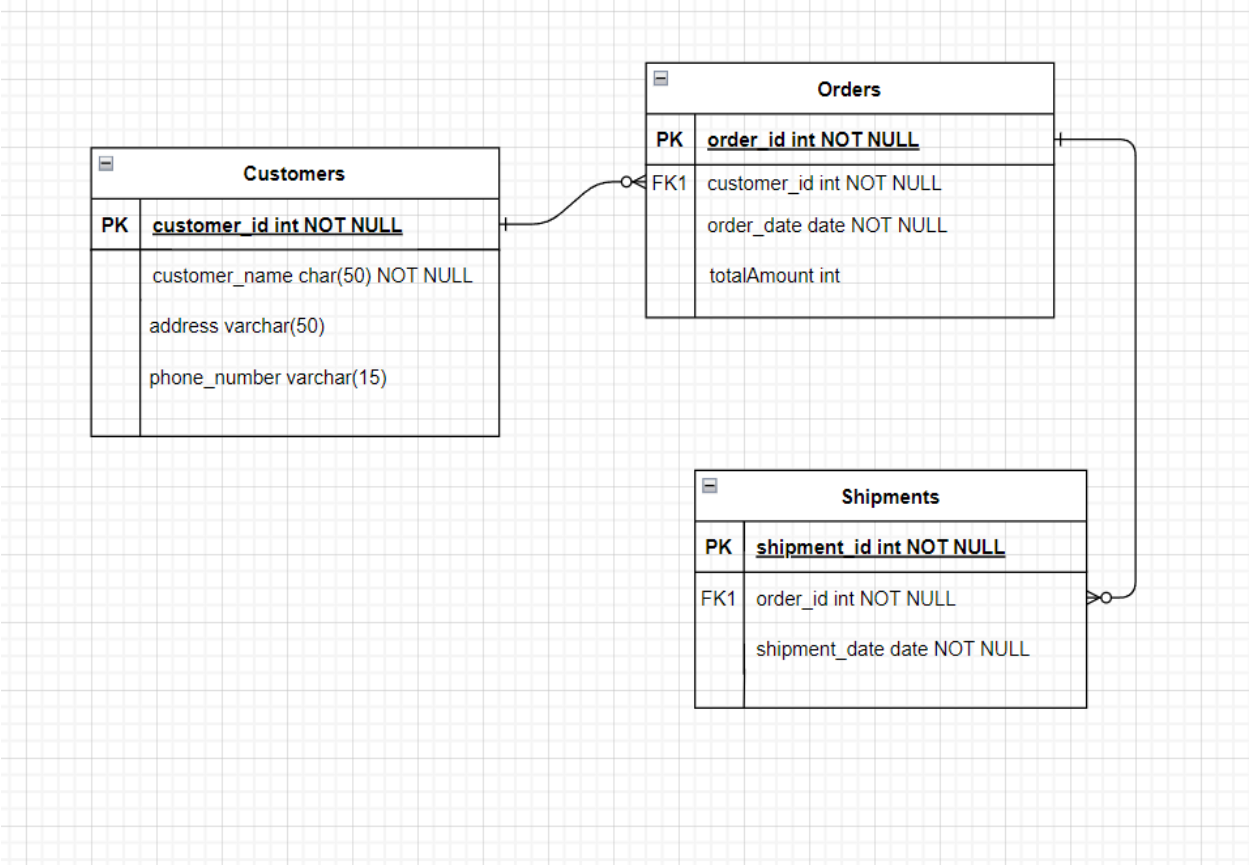
ASSIGNMENT - RDBMS and SQL

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).

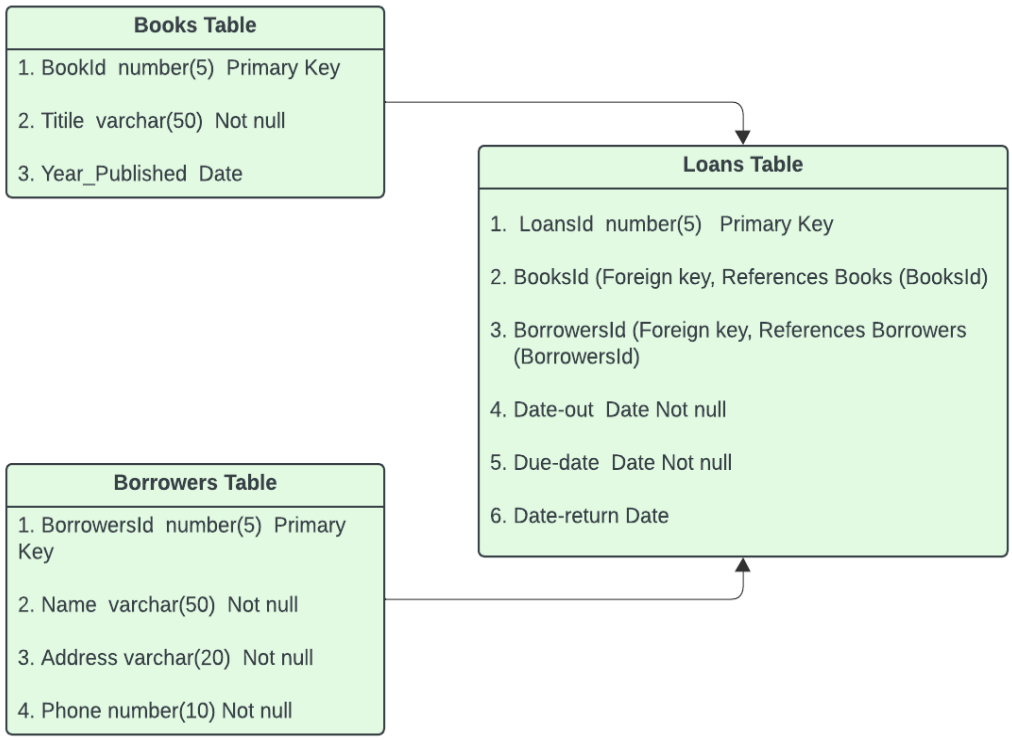
Order to Shipments: One-to-Many (One order can have many shipments, but each shipment is placed by one order).

ASSIGNMENT - RDBMS and SQL

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.



Constraints

The constraints in this schema include NOT NULL and UNIQUE. The NOT NULL constraint ensures that a field must always have a value, which prevents null values from being entered. The UNIQUE constraint ensures that all values in a column are different.

ASSIGNMENT - RDBMS and SQL

Keys

The **primary keys** are BookID, BorrowerID, and LoanID for the Books, Borrowers, and Loans tables respectively. These keys uniquely identify each record in their respective tables.
The **foreign keys** are BookID and BorrowerID in the Loans table. These keys establish a link between the data in two tables. They ensure that the relationship between the Books and Borrowers tables and the Loans table is maintained.

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.

Atomicity: This property ensures that a transaction is treated as a single unit of work. Either all the operations within the transaction are successfully completed, or none of them are. There's no partial execution. If any part of the transaction fails, the entire transaction is rolled back to its initial state.

Consistency: This property 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. The integrity of the data is maintained throughout the transaction.

Isolation: This property 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. Isolation prevents transactions from seeing each other's intermediate states, ensuring that the outcome of each transaction is consistent and predictable.

Durability: This property 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. Even if the system crashes immediately after a transaction commits, the changes made by that transaction should still be present when the system recovers.

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);
```

ASSIGNMENT - RDBMS and SQL

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 - RDBMS and SQL

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.

SQL Query:

Create Database LibrarySystemDB;
Use LibrarySystemDB;

Create table Books(BookId number(5) PRIMARY KEY, Title VARCHAR(30) NOT NULL, Year_Published Date);
Create table Borrowers(BorrowersId Number(5) PRIMARY KEY, Name VARCHAR(30) NOT NULL, Address VARCHAR(50) NOT NULL, PhoneNo NUMBER(10) CHECK(PHONE(LENGTH=10)));
Create table Loans(LoansIId number(5) PRIMARY KEY, Bookid FOREIGN KEY references Books(BooksId), BorrowersId FOREIGN KEY references Borrowers(BorrowersId), Date-out DATE NOT NULL, Date-due DATE NOT NULL, Date-return DATE NOT NULL);

ALTER TABLE Books
ADD COLUMN Genre VARCHAR(50);

ALTER TABLE Borrowers
ADD COLUMN Email VARCHAR(50);

DROP TABLE Books;

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.

- Create a table named as employee with columns ‘employee_id’,’first_name’,’last_name’ and ‘department_id’.we’ll create an index on the ‘department_id’.

- CREATE INDEX department_index ON employees (department_id);

ASSIGNMENT - RDBMS and SQL

This statement creates an index named ‘department_index’ on the ‘department_id’ column of the ‘employee’ table.

This index improves query performance by the following ways :

1. **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.
2. **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.
3. **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.

Now, let's see the impact of removing the index. We'll use the ‘DROP_INDEX’ statement.

DROP INDEX department_index ON employees;
This statement removes the ‘department_index’ index from the ‘employee’ table.

After removing index :

- Slower Query Performance
- Increased Disk I/O

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 USER 'resh’ @ 'localhost ' IDENTIFIED BY '12345';
GRANT SELECT INSERT UPDATE ON USER.* TO 'resh’ @ 'localhost';

FLUSH PRIVILEGES;

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

DROP USER 'resh’ @ ' localhost';

ASSIGNMENT - RDBMS and SQL

Explanation of the SQL Commands

1. CREATE USER: This command is used to create a new user in the SQL database. In this case, the new user is named 'newuser' and the password is set as 'password'.
2. GRANT: This command is used to give privileges to a user. In this case, 'newuser' is granted all privileges.
3. FLUSH PRIVILEGES: This command is used to reload the grant tables in the master database, enabling the changes to take effect immediately.
4. REVOKE: This command is used to remove certain privileges from a user. In this case, the INSERT and UPDATE privileges are being revoked from 'newuser'.
5. DROP USER: This command is used to delete a user from the database. In this case, 'newuser' is being deleted.

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 to **insert** into library tables:

```
insert into books(bookId , bookName , genre) values (121 , 'Ikigai' , 'Self-Help');

insert into category(categoryId , categoryName) values (2 , 'Romance');

insert into author (authorId , authorName , birthYear) values (101, 'Hector Garcia' ,Null);
```

OUTPUT:

BOOIKID	BOOKNAME	GENRE
121	Ikigai	Self-Help

CATEGORYID	CATEGORYNAME
2	Romance

AUTHORID	AUTHORNAME	BIRTHYEAR
101	Hector Garcia	

ASSIGNMENT - RDBMS and SQL

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

```
update books
set authorName='The Secret'
where bookId=121;

update category
set categoryName='Action'
where categoryId = 2;

update author
set birthYear=1914
where authorId=101;
```

OUTPUT:

BOOKID	AUTHORNAME	GENRE
121	The Hobbit	Self-Help

CATEGORYID	CATEGORYNAME
2	Action

AUTHORID	AUTHORNAME	BIRTHYEAR
101	Hector Garcia	1914

SQL Statements to **delete** records based on specific criteria:

```
delete  from books
where bookId=121;
```

BULK INSERT Load Data

```
Infile '/path/example.csv
Into Table Books
Fields Terminated By ','
Enclosed By ' " '
 Lines terminated by '\n'
```