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.

Scenario: Online Store

An online store sells various products to customers. Customers can place orders that contain multiple products. Each product is supplied by a single supplier. The store keeps track of customer information, product details, supplier information, and order history.

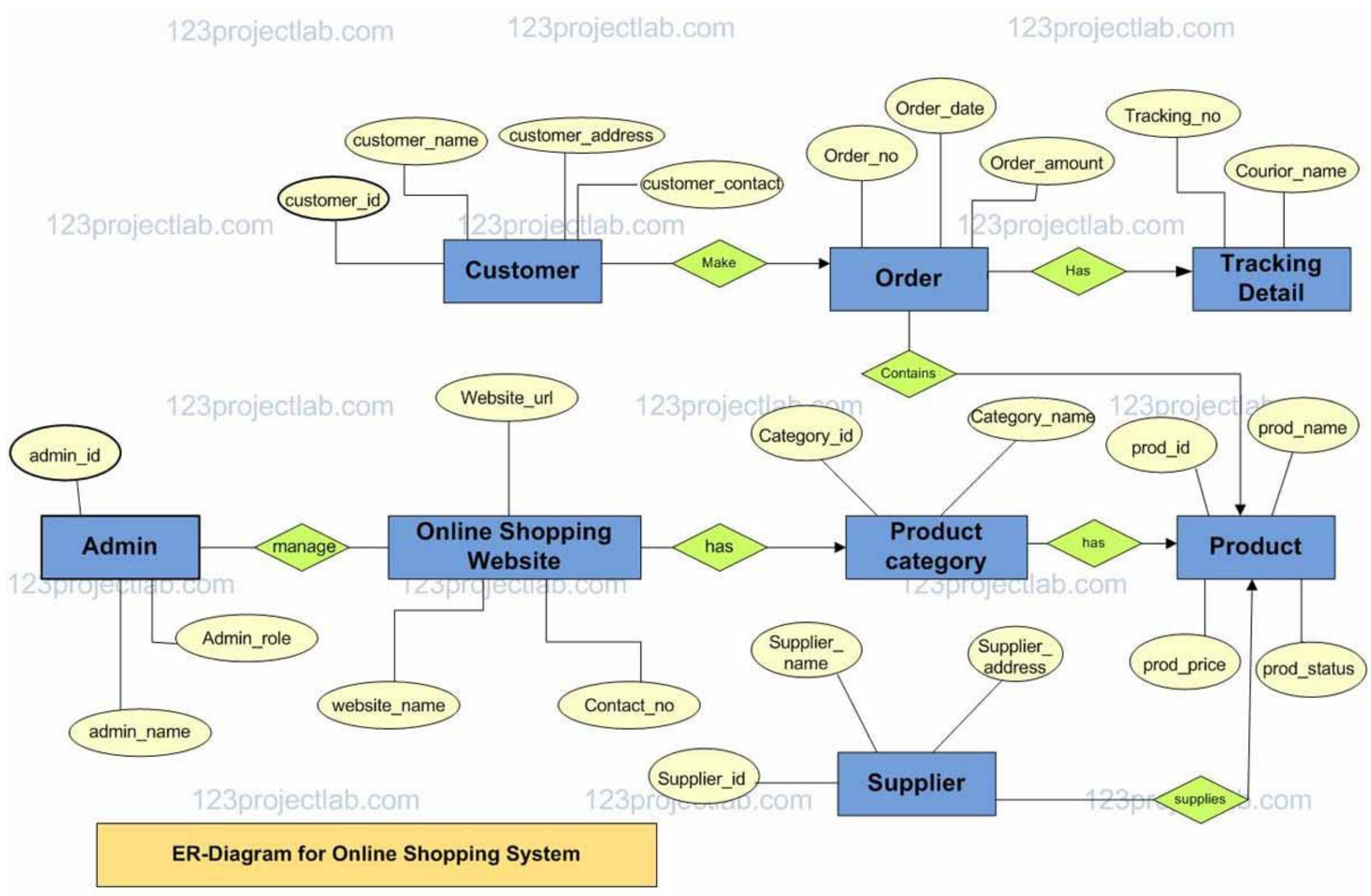
Entities and Attributes

- Customer:
  - CustomerID (Primary Key)
  - Name
  - Email
  - Address
  - Phone
- Product:
  - ProductID (Primary Key)
  - Name
  - Description
  - Price
  - SupplierID (Foreign Key)
- Supplier:
  - SupplierID (Primary Key)
  - Name
  - ContactName
  - Email
  - Phone
- Order:
  - OrderID (Primary Key)
  - CustomerID (Foreign Key)
  - OrderDate
  - TotalAmount
- OrderItem:
  - OrderID (Foreign Key)
  - ProductID (Foreign Key)
  - Quantity
  - UnitPrice

Relationships and Cardinality

- Customer and Order: One-to-Many (A customer can place many orders, but each order belongs to only one customer)
- Product and Supplier: Many-to-One (A product is supplied by one supplier, but a supplier can supply many products)
- Order and OrderItem: One-to-Many (An order can contain many order items, but each order item belongs to only one order)
- Product and OrderItem: Many-to-Many (A product can be in many orders, and an order can contain many products - resolved using OrderItem as an associative entity)

ER Diagram



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.

Tables:

Books:

BookID (Primary Key)

Title

Author

Publisher

PublishYear

ISBN (Unique)

```
CREATE TABLE Books (
    BookID SERIAL PRIMARY KEY, -- Auto-incrementing unique identifier
    Title VARCHAR (255) NOT NULL,
    Author VARCHAR (255) NOT NULL,
    Publisher VARCHAR (100),
    PublishYear INTEGER CHECK (PublishYear > 0), -- Ensure year is positive
    ISBN VARCHAR (13) UNIQUE NOT NULL
);
```

Members:

MemberID (Primary Key)

Name

Email

Phone

```
CREATE TABLE Members (
    MemberID SERIAL PRIMARY KEY, -- Auto-incrementing unique identifier
    Name VARCHAR (255) NOT NULL,
    Email VARCHAR (100) UNIQUE,
    Phone VARCHAR (20)
);
```

Borrowings:

BorrowingID (Primary Key)

BookID (Foreign Key referencing Books)

MemberID (Foreign Key referencing Members)

BorrowDate

ReturnDate

Status (e.g., 'Borrowed', 'Returned')

```
CREATE TABLE Borrowings (
    BorrowingID SERIAL PRIMARY KEY, -- Auto-incrementing unique identifier
    BookID INTEGER REFERENCES Books (BookID) NOT NULL, -- Foreign key reference to Books
    MemberID INTEGER REFERENCES Members (MemberID) NOT NULL, -- Foreign key reference to
Members
    BorrowDate DATE NOT NULL,
    ReturnDate DATE,
    Status VARCHAR (10) CHECK (Status IN ('Borrowed', 'Returned')) NOT NULL
);
```

Constraints:

NOT NULL: Ensure that certain fields cannot be empty.

UNIQUE: Ensure that certain fields have unique values.

CHECK: Apply conditions to ensure data integrity.

Field Constraints:

In the Books table:

BookID: NOT NULL

Title: NOT NULL

ISBN: UNIQUE

In the Members table:

MemberID: NOT NULL

Name: NOT NULL

In the Borrowings table:

BorrowingID: NOT NULL

BookID: NOT NULL

MemberID: NOT NULL

BorrowDate: NOT NULL

Status: CHECK (Status must be either 'Borrowed' or 'Returned')

Primary and Foreign Keys:

Primary Keys: Unique identifiers for each record in a table.

BookID in Books table

MemberID in Members table

BorrowingID in Borrowings table

Foreign Keys: Establish relationships between tables.

BookID in Borrowings table references BookID in Books table

MemberID in Borrowings table references MemberID in Members table

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.

Atomicity: Atomicity ensures that a transaction is treated as a single unit of work. This means that either all of the operations within the transaction are completed successfully, or none of them are. If any part of the transaction fails, the entire transaction is rolled back to its initial state

Consistency: Consistency ensures that a transaction brings the database from one valid state to another valid state. In other words, integrity constraints, business rules, and other constraints are maintained before and after the transaction.

Isolation: Isolation ensures that concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially, i.e., one after the other. Isolation prevents transactions from interfering with each other by ensuring that intermediate transaction states are not visible to other transactions until the transaction is committed.

Durability: Durability guarantees that once a transaction is committed, its effects are permanent and survive system failures. Even in the event of a system crash or power loss, the changes made by committed transactions are preserved in the database.

```
-- Create a table for demonstration
CREATE TABLE Account (
    AccountID INT PRIMARY KEY,
    Balance DECIMAL(10,2)
);

-- Insert some sample data
INSERT INTO Account (AccountID, Balance) VALUES (1, 1000), (2, 2000);

-- Begin a transaction
BEGIN TRANSACTION;

-- Simulate a money transfer between accounts
UPDATE Account SET Balance = Balance - 500 WHERE AccountID = 1;
UPDATE Account SET Balance = Balance + 500 WHERE AccountID = 2;

-- Commit the transaction
COMMIT;

-- Display the updated balances
SELECT * FROM Account;
```

-- Set isolation level to Read Uncommitted

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

xSET TRANSACTION ISOLATION LEVEL READ COMMITTED;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Isolation Level Explanation

READ UNCOMMITTED: The lowest level, allows a transaction to read uncommitted changes made by other transactions. This can lead to "dirty reads." In our scenario, one member might see the book as available even though another transaction is in the process of checking it out.

READ COMMITTED: Prevents dirty reads by only allowing a transaction to read committed changes. However, it can lead to "non-repeatable reads" where a transaction reads the same data twice and gets different results.

REPEATABLE READ: Fixes non-repeatable reads by ensuring that if a transaction reads a row, it will see the same values throughout the transaction. However, it can still experience "phantom reads" where new rows appear in the result set.

SERIALIZABLE: The most restrictive level. It acts as if transactions were executed one after the other, completely eliminating concurrency issues. This is the safest but also the least performant option.

Important Notes:

The FOR-UPDATE clause in the SELECT statement is crucial for locking the book record and preventing race conditions.

Isolation levels are a trade-off between consistency and performance. Choose the appropriate level based on your application's requirements.

Most databases use READ COMMITTED as the default isolation level.

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 a new database

CREATE DATABASE LibraryDB;

```
-- Use the newly created database
USE LibraryDB;

-- Create the Books table
CREATE TABLE Books (
    BookID INT PRIMARY KEY,
    Title VARCHAR(255) NOT NULL,
    Author VARCHAR(255) NOT NULL,
    Publisher VARCHAR(255),
    PublishYear INT,
    ISBN VARCHAR(20) UNIQUE
);

-- Create the Members table
CREATE TABLE Members (
    MemberID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Email VARCHAR(100),
    Phone VARCHAR(20)
);

-- Create the Borrowings table
CREATE TABLE Borrowings (
    BorrowingID INT PRIMARY KEY,
    BookID INT,
    MemberID INT,
    BorrowDate DATE NOT NULL,
    ReturnDate DATE,
    Status ENUM('Borrowed', 'Returned'),
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID)
);
ALTER TABLE Books ADD COLUMN Genre VARCHAR(50);
```

```
-- Drop the redundant Watchlist table

DROP TABLE IF EXISTS Watchlist;
```

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.

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

SELECT * FROM Books WHERE Title = 'Harry Potter';

SELECT * FROM Books WHERE Title = 'Harry Potter';

DROP INDEX idx_title ON Books;

SELECT * FROM Books WHERE Title = 'Harry Potter';
```

In summary, creating an index on a table column can improve query performance by allowing the database to quickly locate rows that match the specified criteria without having to scan the entire table. Dropping the index may result in slower query performance, especially for queries that rely on the indexed column for filtering or sorting. However, indexes also have associated costs such as increased storage space and potential overhead on data modification operations.

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

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

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

REVOKE INSERT ON database_name.table_name FROM 'new_user'@'localhost';
```

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

In this script:

Replace 'new_user' with the username you want to create.

Replace 'password' with the password for the new user.

Replace 'database_name' with the name of the database you want to grant privileges on.

Replace 'table_name' with the name of the table from which you want to revoke privileges.

This script creates a new user, grants SELECT, INSERT, and UPDATE privileges on a specific database to the user, revokes the INSERT privilege on a specific table, and finally drops the user.

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.

INSERT INTO statement to add new records:

```sql
-- Insert a new book record
INSERT INTO Books (BookID, Title, Author, Publisher, PublishYear, ISBN)
VALUES (1, 'To Kill a Mockingbird', 'Harper Lee', 'J.B. Lippincott & Co.', 1960, '9780061120084');

-- Insert a new member record
INSERT INTO Members (MemberID, Name, Email, Phone)
VALUES (1, 'John Doe', 'john@example.com', '123-456-7890');
```

UPDATE statement to modify existing records:

```sql
-- Update the Publisher of a book with a specific BookID
UPDATE Books
SET Publisher = 'Harper Perennial Modern Classics'
WHERE BookID = 1;
```

DELETE FROM statement to remove records based on specific criteria:

```
-- Delete a book record with a specific BookID
DELETE FROM Books
WHERE BookID = 1;
```

BULK INSERT operation to load data from an external source (e.g., a CSV file):

```
-- Bulk insert data from a CSV file into the Books table
BULK INSERT Books
FROM 'C:\path\to\books.csv'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2 -- Skip header row
);
```

In these SQL statements:

INSERT INTO is used to add new records to the Books and Members tables.

UPDATE modifies the Publisher of a specific book.

DELETE FROM removes a book record based on a specific BookID.

BULK INSERT loads data from an external CSV file into the Books table, specifying the field and row terminators and skipping the header row.