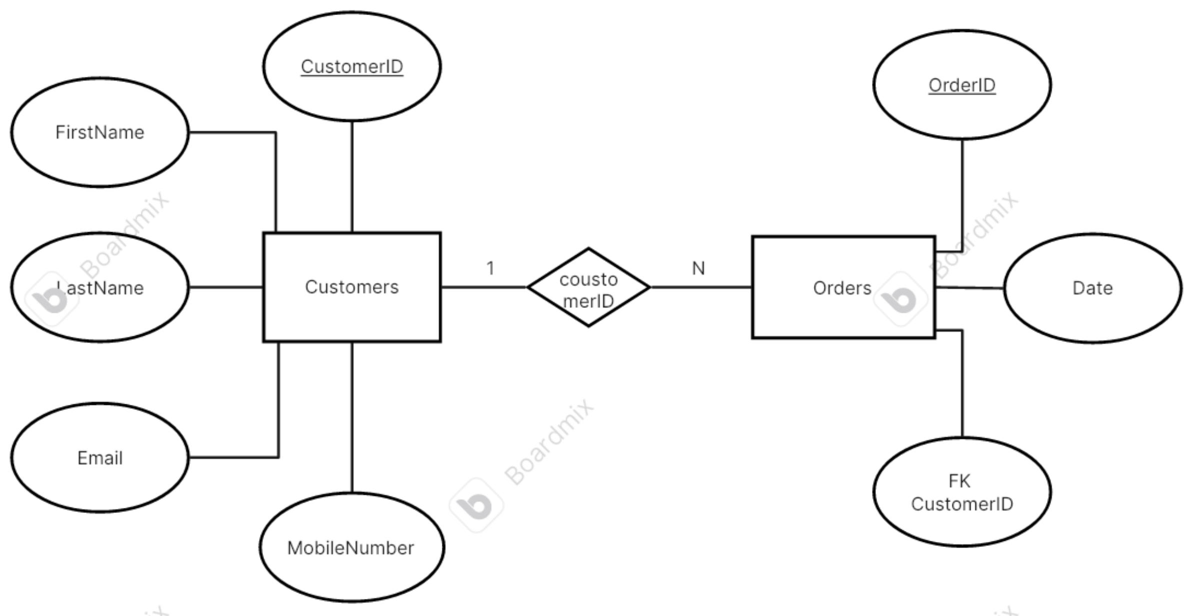
Assignment 1: Analyze a given business scenario and create an ER diagram that includes entities, relationships, attributes, and cardinality. Ensure that the diagram reflects proper normalization up to the third normal form.



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.

Create Authors table

```
CREATE TABLE Authors (
    AuthorID NUMBER PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL
);
```

Create Categories table

```
CREATE TABLE Categories (
    CategoryID NUMBER PRIMARY KEY,
    CategoryName VARCHAR(100) UNIQUE NOT NULL
);
```

Create Publishers table

```
CREATE TABLE Publishers (
    PublisherID NUMBER PRIMARY KEY,
    PublisherName VARCHAR(100) UNIQUE NOT NULL
);
```

Create Books table

```
CREATE TABLE Books (
    BookID NUMBER PRIMARY KEY,
    Title VARCHAR(255) NOT NULL,
    PublisherID NUMBER NOT NULL,
    PublicationYear NUMBER CHECK (PublicationYear > 0),
    FOREIGN KEY (PublisherID) REFERENCES Publishers(PublisherID)
);
```

Create Members table

```
CREATE TABLE Members (
    MemberID NUMBER PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL
);
```

Create Loans table

```
CREATE TABLE Loans (
    LoanID NUMBER PRIMARY KEY,
    BookID NUMBER NOT NULL,
    MemberID NUMBER NOT NULL,
    LoanDate DATE NOT NULL,
    ReturnDate DATE,
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID)
);
```

Create BookAuthors table

```
CREATE TABLE BookAuthors (
    BookID NUMBER NOT NULL,
    AuthorID NUMBER NOT NULL,
    PRIMARY KEY (BookID, AuthorID),
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (AuthorID) REFERENCES Authors(AuthorID)
);
```

Create BookCategories table

```
CREATE TABLE BookCategories (
    BookID NUMBER NOT NULL,
    CategoryID NUMBER NOT NULL,
    PRIMARY KEY (BookID, CategoryID),
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)
);
```

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.

ACID Properties of a Transaction:

ACID stands for Atomicity, Consistency, Isolation, and Durability. These properties ensure reliable processing of database transactions.

1. Atomicity: This guarantees that a transaction is treated as a single unit, which either completes in its entirety or not at all. If any part of the transaction fails, the entire transaction is rolled back, leaving the database unchanged.

2. Consistency: This ensures that a transaction brings the database from one valid state to another, maintaining database invariants. In other words, the data should be valid according to all defined rules, including constraints, cascades, and triggers.

3. Isolation: This property ensures that concurrent transactions do not affect each other. Transactions must produce the same results as if they were executed serially, even though they are run concurrently.

4. Durability: This property guarantees that once a transaction is committed, it will remain so, even in the case of a system failure. Changes made by the transaction are permanently recorded in the database.

Transactions with Locking and Isolation Levels

First, let's create a simple table to demonstrate transactions and isolation levels.

```
CREATE TABLE accounts (
   account_id NUMBER PRIMARY KEY,
   balance NUMBER
);

INSERT INTO accounts (account_id, balance) VALUES (1, 1000);
INSERT INTO accounts (account_id, balance) VALUES (2, 2000);
COMMIT;
```

Transaction with Locking and Different Isolation Levels

1. Read Uncommitted (Lowest Isolation Level)

Oracle doesn't support `READ UNCOMMITTED` isolation level directly, but you can simulate it by setting `READ ONLY` and allowing dirty reads (this is generally not recommended due to the risk of inconsistencies).

```
-- Session 1: Start a transaction to update the balance
SET TRANSACTION READ WRITE;
UPDATE accounts SET balance = balance + 500 WHERE account_id = 1;
-- Don't commit yet

-- Session 2: Read the balance before Session 1 commits
SET TRANSACTION READ ONLY;
SELECT balance FROM accounts WHERE account_id = 1;
-- This might show the uncommitted balance depending on the implementation
-- Session 1: Commit the transaction
COMMIT;
```

2. Read Committed (Default Isolation Level)

Transactions see committed changes but not uncommitted ones.

```
-- Session 1: Start a transaction to update the balance
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
UPDATE accounts SET balance = balance + 500 WHERE account_id = 1;
-- Don't commit yet

-- Session 2: Read the balance
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SELECT balance FROM accounts WHERE account_id = 1;
-- This will show the balance before the update by Session 1 because Session 1 hasn't committed yet

-- Session 1: Commit the transaction
```

```
COMMIT;

-- Session 2: Read the balance again after commit
SELECT balance FROM accounts WHERE account_id = 1;
-- This will show the updated balance now
```

3. Repeatable Read

Oracle doesn't directly support `REPEATABLE READ`, but `SERIALIZABLE` can be used to achieve similar behavior.

```
-- Session 1: Start a transaction to update the balance
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
UPDATE accounts SET balance = balance + 500 WHERE account_id = 1;
-- Don't commit yet

-- Session 2: Start a transaction and read the balance
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SELECT balance FROM accounts WHERE account_id = 1;
-- This will show the balance before the update by Session 1 because Session 1 hasn't committed yet

-- Session 1: Commit the transaction
COMMIT;

-- Session 2: Read the balance again
SELECT balance FROM accounts WHERE account_id = 1;
-- This will still show the old balance if Session 2's transaction started before Session 1 committed
COMMIT;
```

4. Serializable (Highest Isolation Level)

Transactions execute sequentially, guaranteeing complete isolation but impacting performance.

```
-- Session 1: Start a transaction to update the balance
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
UPDATE accounts SET balance = balance + 500 WHERE account_id = 1;
-- Don't commit yet

-- Session 2: Start a transaction to update the balance of another account
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
UPDATE accounts SET balance = balance + 300 WHERE account_id = 2;
-- This will succeed because it does not conflict with the update by Session 1

-- Try to read balance in Session 2
SELECT balance FROM accounts WHERE account_id = 1;
-- This will be blocked until Session 1 commits or rolls back

-- Session 1: Commit the transaction
COMMIT;

-- Session 2: Read the balance after Session 1 commits
SELECT balance FROM accounts WHERE account_id = 1;
-- This will show the updated balance now
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.

Create Authors table

```
CREATE TABLE Authors (
   AuthorID NUMBER PRIMARY KEY,
   FirstName VARCHAR(50) NOT NULL,
   LastName VARCHAR(50) NOT NULL
);
```

Create Categories table

```
CREATE TABLE Categories (
   CategoryID NUMBER PRIMARY KEY,
   CategoryName VARCHAR(100) UNIQUE NOT NULL
);
```

Create Publishers table

```
CREATE TABLE Publishers (
   PublisherID NUMBER PRIMARY KEY,
   PublisherName VARCHAR(100) UNIQUE NOT NULL
);
```

Create Books table

```
CREATE TABLE Books (
   BookID NUMBER PRIMARY KEY,
   Title VARCHAR(255) NOT NULL,
   PublisherID NUMBER NOT NULL,
   PublicationYear NUMBER CHECK (PublicationYear > 0),
   FOREIGN KEY (PublisherID) REFERENCES Publishers(PublisherID)
);
```

Create Members table

```
CREATE TABLE Members (
   MemberID NUMBER PRIMARY KEY,
   FirstName VARCHAR(50) NOT NULL,
   LastName VARCHAR(50) NOT NULL,
   Email VARCHAR(100) UNIQUE NOT NULL
);
```

Create Loans table

```
CREATE TABLE Loans (
   LoanID NUMBER PRIMARY KEY,
   BookID NUMBER NOT NULL,
   MemberID NUMBER NOT NULL,
   LoanDate DATE NOT NULL,
   ReturnDate DATE,
   FOREIGN KEY (BookID) REFERENCES Books(BookID),
   FOREIGN KEY (MemberID) REFERENCES Members(MemberID)
);
```

Create BookAuthors table

```
CREATE TABLE BookAuthors (
   BookID NUMBER NOT NULL,
   AuthorID NUMBER NOT NULL,
   PRIMARY KEY (BookID, AuthorID),
   FOREIGN KEY (BookID) REFERENCES Books(BookID),
   FOREIGN KEY (AuthorID) REFERENCES Authors(AuthorID)
);
```

Create BookCategories table

```
CREATE TABLE BookCategories (
   BookID NUMBER NOT NULL,
   CategoryID NUMBER NOT NULL,
    PRIMARY KEY (BookID, CategoryID),
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)
);
Alter
ALTER TABLE Books ADD COLUMN ISBN VARCHAR(13) UNIQUE;
DROP TABLE BookCategories;
```

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 TABLE employees (
    employee_id NUMBER PRIMARY KEY,
    employee_name VARCHAR2(20),
    department_id NUMBER
);

INSERT INTO employees (employee_id, employee_name, department_id)
VALUES (1, 'Puneeth', 10);
INSERT INTO employees (employee_id, employee_name, department_id)
VALUES (2, 'sai’, 20);
INSERT INTO employees (employee_id, employee_name, department_id)
VALUES (3, 'Baskar', 30);
INSERT INTO employees (employee_id, employee_name, department_id)
VALUES (4, 'Avinash', 10);
Commit;
```

create an index on the `department_id` column:

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

This creates an index named `dept_index` on the `department_id` column in the `employees` table.

execute a query that filters records based on the `department_id`:

```
SELECT * FROM employees WHERE department_id = 10;
```

With the index in place, Oracle can use it to quickly locate the rows where `department_id = 10`, resulting in faster query execution. The index essentially acts as a quick reference to the rows that satisfy the condition, rather than scanning through the entire table.

drop the index and see how it affects query execution:

```
DROP INDEX dept_index;
```

After dropping the index, if we execute the same query:

```
SELECT * FROM employees WHERE department_id = 10;
```

Oracle will no longer have the index to use for quick lookups. Instead, it may need to perform a full table scan, which can be slower, especially on larger tables. Without the index, Oracle has to read every row in the table to find the ones that match the condition.

In summary, indexes improve query performance by providing a faster way to locate rows that satisfy certain conditions, reducing the need for full table scans. However, indexes come with a cost in terms of storage space and maintenance overhead, so they should be used judiciously based on the specific needs of the application.

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.

Step 1: Create a new database user

```
CREATE USER employee IDENTIFIED BY employee123;
```

Step 2: Grant specific privileges to the user

```
GRANT CONNECT TO employee;
GRANT RESOURCE TO employee;
GRANT CREATE TABLE TO employee;
```

Step 3: Revoke certain privileges from the user

```
REVOKE RESOURCE FROM employee;
```

Step 4: Drop the user

```
DROP USER employee CASCADE;
```

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 a new record into the Books table
INSERT INTO Books (Bookid, title, publisherID, publicationYear)
VALUES (5, 'To Kill a Mockingbird', 'Harper Lee', 1960);

-- Update the genre of a book
UPDATE Books
SET title = 'One Piece'
WHERE Bookid = 5;
-- Delete records from the Loans table based on specific criteria (e.g., overdue books)
```

```
DELETE FROM Loans
WHERE returnDate < SYSDATE;


-- Assuming you have a CSV file named 'new_books.csv' with columns: book_id,
title, author, publication_year, genre, available_copies
-- Create an external table to access the CSV data
CREATE TABLE ext_books (
   book_id NUMBER,
   title VARCHAR2(100),
   author VARCHAR2(100),
   publication_year NUMBER,
   genre VARCHAR2(50),
   available_copies NUMBER
)
ORGANIZATION EXTERNAL (
   TYPE ORACLE_LOADER
   DEFAULT DIRECTORY ext_dir
   ACCESS PARAMETERS (
      RECORDS DELIMITED BY NEWLINE
      FIELDS TERMINATED BY ','
      MISSING FIELD VALUES ARE NULL
   )
   LOCATION ('new_books.csv')
);
-- Insert data from the external table into the Books table
INSERT INTO Books (bookid, title, publisherID,publication_year,)
SELECT * FROM ext_books;
```