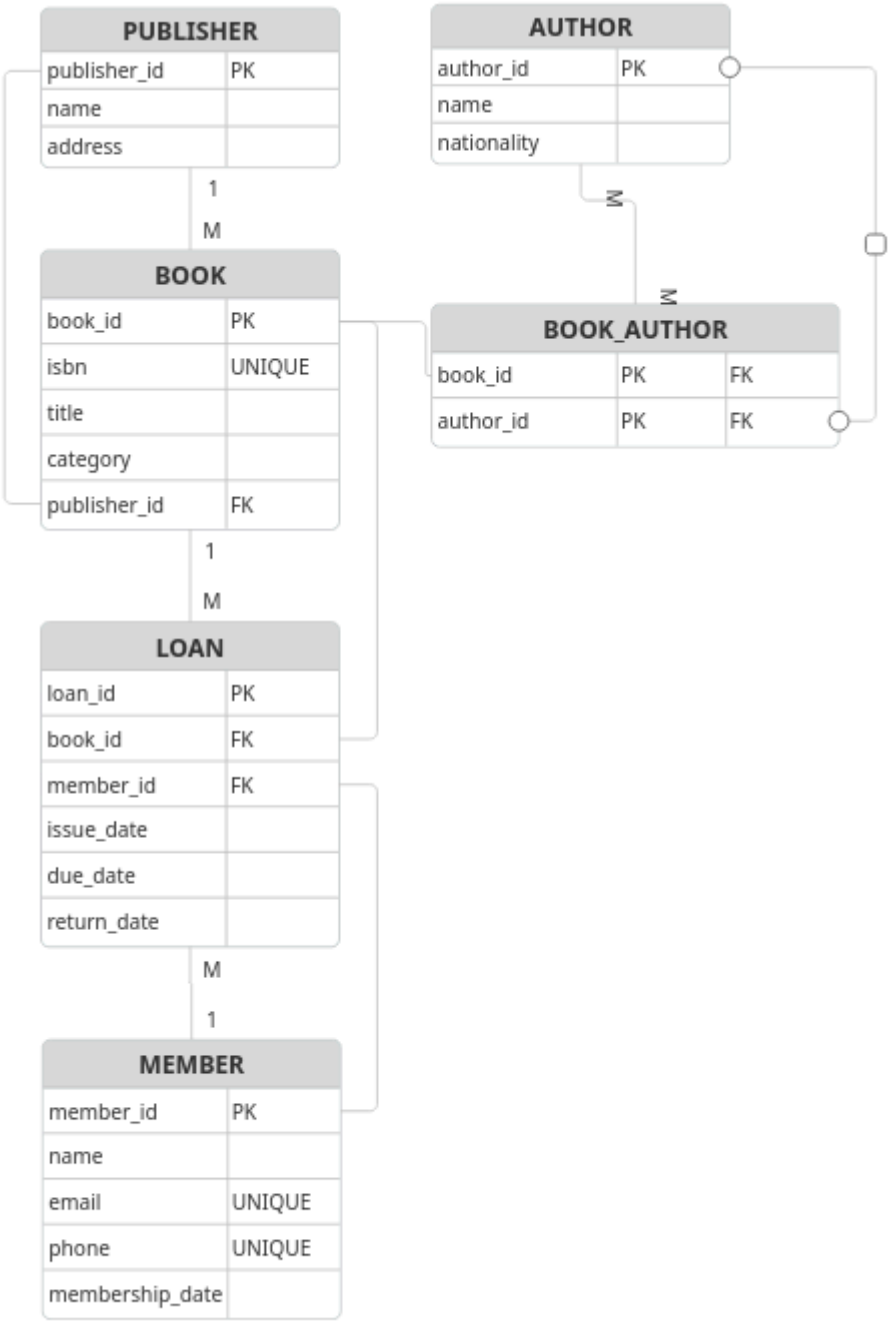
RDBMS and SQL Assignment

(Day-1)

(55998) Vinayak Majhi

Q.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.



Relationship	**Cardinality**
Publisher → Book	1 : M
Book ↔ Author	M : M (via BOOK_AUTHOR)
Member → Loan	1 : M
Book → Loan	1 : M

1NF:

All attributes are atomic
No repeating groups
Each table has a primary key

2NF:
Second Normal Form (2NF)
No partial dependency
Composite key table (BOOK_AUTHOR) has attributes fully dependent on both keys

3NF:
No transitive dependencies
Non-key attributes depend only on the primary key
Publisher details are separated from Book

Q.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 TABLE Member (
    member_id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(15) UNIQUE,
    membership_date DATE NOT NULL,
    status VARCHAR(10) CHECK (status IN ('ACTIVE', 'INACTIVE'))
```

```
);

INSERT INTO Member (member_id, name, email, phone, membership_date, status)
VALUES (201, 'Alice Brown', 'alice@gmail.com', '9876543210', '2024-01-10', 'ACTIVE');

INSERT INTO Member (member_id, name, email, phone, membership_date, status)
VALUES (202, 'Bob Smith', 'bob@gmail.com', '9123456780', '2024-02-15', 'INACTIVE');



CREATE TABLE Publisher (
    publisher_id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE,
    address VARCHAR(200)
);

INSERT INTO Publisher (publisher_id, name, address)
VALUES (1, 'Penguin Random House', 'New York');

INSERT INTO Publisher (publisher_id, name, address)
VALUES (2, Om Books International', 'New Delhi');



CREATE TABLE Author (
    author_id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    nationality VARCHAR(50)
);

INSERT INTO Author (author_id, name, nationality)
VALUES (101, 'George Orwell', 'British');
```

```
INSERT INTO Author (author_id, name, nationality)
VALUES (102, 'Aldous Huxley', 'British');



CREATE TABLE Book (
    book_id INT PRIMARY KEY,
    isbn VARCHAR(13) UNIQUE NOT NULL,
    title VARCHAR(150) NOT NULL,
    category VARCHAR(50),
    publisher_id INT NOT NULL,

    FOREIGN KEY (publisher_id)
        REFERENCES Publisher(publisher_id)
        ON DELETE RESTRICT
);
INSERT INTO Book (book_id, isbn, title, category, publisher_id)
VALUES (301, '9780451524935', '1984', 'Dystopian', 1);

INSERT INTO Book (book_id, isbn, title, category, publisher_id)
VALUES (302, '9780060850524', 'Brave New World', 'Science Fiction',
2);



CREATE TABLE Book_Author (
    book_id INT,
    author_id INT,

    PRIMARY KEY (book_id, author_id),

    FOREIGN KEY (book_id)
        REFERENCES Book(book_id)
        ON DELETE CASCADE,

    FOREIGN KEY (author_id)
```

```
        REFERENCES Author(author_id)
        ON DELETE CASCADE
);

INSERT INTO Book_Author (book_id, author_id)
VALUES (301, 101);

INSERT INTO Book_Author (book_id, author_id)
VALUES (302, 102);



CREATE TABLE Loan (
    loan_id INT PRIMARY KEY,
    book_id INT NOT NULL,
    member_id INT NOT NULL,
    issue_date DATE NOT NULL,
    due_date DATE NOT NULL,
    return_date DATE,

    CHECK (due_date > issue_date),
    CHECK (return_date IS NULL OR return_date >= issue_date),

    FOREIGN KEY (book_id)
        REFERENCES Book(book_id)
        ON DELETE CASCADE,

    FOREIGN KEY (member_id)
        REFERENCES Member(member_id)
        ON DELETE CASCADE
);

INSERT INTO Loan (loan_id, book_id, member_id, issue_date,
due_date, return_date)
VALUES (401, 301, 201, '2024-03-01', '2024-03-15', NULL);
```

```
INSERT INTO Loan (loan_id, book_id, member_id, issue_date, due_date, return_date)
VALUES (402, 302, 202, '2024-03-05', '2024-03-20', '2024-03-18');
```

Q.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:

A transaction is a sequence of database operations that must be executed as a single logical unit of work.

The ACID properties ensure the reliability and correctness of transactions in a database system.

1. Atomicity

Atomicity means that a transaction is treated as an indivisible unit.
Either all the operations of the transaction are executed successfully, or none of them are applied.

If any operation within the transaction fails, the entire transaction is rolled back, and the database returns to its previous state.

Example:

In a bank transfer, money is debited from one account and credited to another.
If the debit succeeds but the credit fails, atomicity ensures that the debit is undone.

2. Consistency

Consistency ensures that a transaction brings the database from one valid state to another valid state.
All integrity constraints, such as primary keys, foreign keys, CHECK constraints, and business rules, must remain satisfied after the transaction commits.

Example:
In a library system, the due_date of a loan must always be greater than the issue_date.
If a transaction violates this rule, it will not be committed.

3. Isolation

Isolation ensures that multiple transactions executing concurrently do not affect each other.
Each transaction behaves as if it is the only transaction running in the system, even though many transactions may execute at the same time.

Intermediate changes made by one transaction should not be visible to other transactions until the transaction is committed.

Example

If one user is updating an account balance, another user should not see the partially updated balance.

4. Durability

Durability guarantees that once a transaction is committed, its changes are permanently stored in the database.
The data will remain intact even in case of system crash, power failure, or restart.

Example:
Once a loan record is committed in the database, it will still exist after the system restarts.

SQL Transactions, Locking, and Concurrency Control

```
CREATE TABLE Account (
    acc_id INT PRIMARY KEY,
    balance INT CHECK (balance >= 0)
);
```

Transaction with Explicit Locking:

```
START TRANSACTION;

SELECT balance
FROM Account
WHERE acc_id = 1
FOR UPDATE;
```

The **FOR UPDATE** clause places a row-level lock on the selected record.
This prevents other transactions from modifying the same row until the current transaction finishes.

```
UPDATE Account
SET balance = balance - 100
WHERE acc_id = 1;

COMMIT;
```

The row is locked when selected.
The update is performed safely.
The commit releases the lock.
This ensures atomicity and isolation.

Isolation Levels and Concurrency Problems

Isolation levels control how much one transaction can see the changes made by other concurrent transactions.

READ UNCOMMITTED

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
START TRANSACTION;

SELECT balance FROM Account WHERE acc_id = 1;

- Allows reading uncommitted data from other transactions.
- Can cause dirty reads.
- This level provides the least isolation and is rarely used.

READ COMMITTED

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;

SELECT balance FROM Account WHERE acc_id = 1;

Only committed data is visible.

Prevents dirty reads.

The same query may return different results.

REPEATABLE READ

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION;

SELECT balance FROM Account WHERE acc_id = 1;

COMMIT;

Ensures that rows read once cannot change during the transaction.
Prevents dirty reads and non-repeatable reads.
Phantom reads may still occur in some databases.

SERIALIZABLE

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
START TRANSACTION;

SELECT * FROM Account WHERE balance > 1000;

COMMIT;

Transactions are executed as if they are running one after another.
Prevents dirty reads, non-repeatable reads, and phantom reads.
Provides the highest isolation but lowest performance.

Dirty Read

Transaction T1

START TRANSACTION;
UPDATE Account SET balance = balance - 500 WHERE acc_id = 1;

T1 has not committed yet

Transaction T2 (READ UNCOMMITTED)

SELECT balance FROM Account WHERE acc_id = 1;

If T1 later rolls back, T2 has read **invalid data**, which is called a **dirty read**.

Q.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 DATABASE LibraryDB;

USE LibraryDB;

CREATE TABLE Member (
    member_id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(15),
    membership_date DATE NOT NULL
);

CREATE TABLE Publisher (
    publisher_id INT PRIMARY KEY,
    name VARCHAR(100) UNIQUE NOT NULL,
    address VARCHAR(200)
);

CREATE TABLE Author (
    author_id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    nationality VARCHAR(50)
);

CREATE TABLE Book (
    book_id INT PRIMARY KEY,
    isbn VARCHAR(13) UNIQUE NOT NULL,
    title VARCHAR(150) NOT NULL,
    category VARCHAR(50),
    publisher_id INT NOT NULL,
```

```
    FOREIGN KEY (publisher_id)
        REFERENCES Publisher(publisher_id)
);

CREATE TABLE Book_Author (
    book_id INT,
    author_id INT,

    PRIMARY KEY (book_id, author_id),

    FOREIGN KEY (book_id)
        REFERENCES Book(book_id),

    FOREIGN KEY (author_id)
        REFERENCES Author(author_id)
);

CREATE TABLE Loan (
    loan_id INT PRIMARY KEY,
    book_id INT NOT NULL,
    member_id INT NOT NULL,
    issue_date DATE NOT NULL,
    due_date DATE NOT NULL,
    return_date DATE,

    CHECK (due_date > issue_date),

    FOREIGN KEY (book_id)
        REFERENCES Book(book_id),

    FOREIGN KEY (member_id)
        REFERENCES Member(member_id)
);

ALTER TABLE Member
ADD status VARCHAR(10)
CHECK (status IN ('ACTIVE', 'INACTIVE'));
```

```
ALTER TABLE Book
MODIFY title VARCHAR(200);

ALTER TABLE Member
ADD CONSTRAINT uniq_phone UNIQUE (phone);

ALTER TABLE Loan
DROP FOREIGN KEY Loan_ibfk_1;


ALTER TABLE Loan
ADD CONSTRAINT fk_loan_book
FOREIGN KEY (book_id)
REFERENCES Book(book_id)
ON DELETE CASCADE;

CREATE TABLE Book_Category (
    category_id INT PRIMARY KEY,
    category_name VARCHAR(50)
);

DROP TABLE Book_Category;
```

Q.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.

An index is a database object used to speed up data retrieval from a table.
It works similarly to an index in a book, allowing the database to locate rows quickly without scanning the entire table.

```
CREATE TABLE Book (
    book_id INT PRIMARY KEY,
    isbn VARCHAR(13) UNIQUE NOT NULL,
```

```
    title VARCHAR(200) NOT NULL,
    category VARCHAR(50),
    publisher_id INT
);
```

Query without any index

```
SELECT *
FROM Book
WHERE title = '1984';
```

Without an index on title, the database performs a full table scan.
Every row is checked to find the matching title.
Performance degrades as the table grows larger.

```
CREATE INDEX idx_book_title
ON Book(title);
```

The index stores sorted references to rows based on title.
The database can directly locate the matching rows.
This significantly reduces search time for SELECT queries.

Query with Index

```
SELECT *
FROM Book
WHERE title = '1984';
```

Database uses the index instead of scanning the entire table.
Query execution becomes much faster, especially for large datasets.

```
DROP INDEX idx_book_title ON Book;
```

Query again after dropping index

```
SELECT *
FROM Book
WHERE title = '1984';
```

Database can no longer use the index.
Query reverts to a full table scan.
Execution time increases as table size increases.

Q.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.

Creating a new database user

```
CREATE USER 'library_user'@'localhost'
IDENTIFIED BY 'lib@123';
```

Granting privileges

```
GRANT SELECT, INSERT, UPDATE
ON LibraryDB.*
TO 'library_user'@'localhost';

FLUSH PRIVILEGES;

SHOW GRANTS FOR 'library_user'@'localhost';
```

Revoking privileges

```
REVOKE UPDATE
ON LibraryDB.*
FROM 'library_user'@'localhost';

FLUSH PRIVILEGES;
```

Flush is used to apply changes

Drop the user

DROP USER 'library_user'@'localhost';

Completely removes the user account
All associated privileges are automatically deleted

Q.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 Publisher (publisher_id, name, address)
VALUES (1, 'Penguin Random House', 'New York');

INSERT INTO Publisher (publisher_id, name, address)
VALUES (2, 'HarperCollins', 'London');

INSERT INTO Author (author_id, name, nationality)
VALUES (101, 'George Orwell', 'British');

INSERT INTO Author (author_id, name, nationality)
VALUES (102, 'Aldous Huxley', 'British');

INSERT INTO Member (member_id, name, email, phone, membership_date, status)
VALUES (201, 'Alice Brown', 'alice@gmail.com', '9876543210', '2024-01-10', 'ACTIVE');

INSERT INTO Book (book_id, isbn, title, category, publisher_id)
VALUES
(301, '9780451524935', '1984', 'Dystopian', 1),
(302, '9780060850524', 'Brave New World', 'Science Fiction', 2);

```
INSERT INTO Loan (loan_id, book_id, member_id, issue_date, due_date, return_date)
VALUES (401, 301, 201, '2024-03-01', '2024-03-15', NULL);

UPDATE Member
SET status = 'INACTIVE'
WHERE member_id = 201;


UPDATE Book
SET category = 'Political Fiction'
WHERE title = '1984';

UPDATE Loan
SET return_date = '2024-03-12'
WHERE loan_id = 401;

UPDATE Loan
SET due_date = DATE_ADD(due_date, INTERVAL 7 DAY)
WHERE return_date IS NULL;

DELETE FROM Member
WHERE status = 'INACTIVE';

DELETE FROM Loan
WHERE return_date < '2023-01-01';

DELETE FROM Book
WHERE book_id NOT IN (
   SELECT DISTINCT book_id FROM Loan
);
```

External file: books_data.csv

Books_data.csv Format:

303,9780140449136,The Odyssey,Classic,1
304,9780140449181,The Iliad,Classic,1

```
LOAD DATA INFILE '/path/to/books_data.csv'
INTO TABLE Book
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
(book_id, isbn, title, category, publisher_id);
```

(Day-2)

Q.1)

```
SELECT *
FROM customers;

SELECT name, email
FROM customers
WHERE city = 'Mumbai';
```

Q.2)

```
SELECT
    c.customer_id,
    c.name,
    c.email,
    c.region,
    o.order_id,
    o.order_date,
    o.amount
FROM customers c
INNER JOIN orders o
    ON c.customer_id = o.customer_id
WHERE c.region = 'South';
```

```
SELECT
    c.customer_id,
    c.name,
    c.email,
    c.region,
    o.order_id,
    o.order_date,
    o.amount
FROM customers c
LEFT JOIN orders o
    ON c.customer_id = o.customer_id;

SELECT
    c.customer_id,
    c.name,
    c.email
FROM customers c
LEFT JOIN orders o
    ON c.customer_id = o.customer_id
WHERE o.order_id IS NULL;
```

Q.3)

```
SELECT DISTINCT c.customer_id, c.name, c.email
FROM customers c
JOIN orders o
    ON c.customer_id = o.customer_id
WHERE o.amount > (
    SELECT AVG(amount)
    FROM orders
);

SELECT customer_id, name, region
FROM customers
WHERE region = 'South'
```

```
UNION

SELECT customer_id, name, region
FROM customers
WHERE region = 'North';
```

Q.4)

```
START TRANSACTION;

INSERT INTO orders (order_id, order_date, quantity, customer_id, product_id)
VALUES (201, '2024-04-10', 2, 101, 1);

COMMIT;

START TRANSACTION;

UPDATE products
SET stock = stock - 2
WHERE product_id = 1;

ROLLBACK;
```

BEGIN / START TRANSACTION:	Starts a transaction
COMMIT:	Saves changes permanently
ROLLBACK:	Undoes changes

Q.5)

```
START TRANSACTION;
```

```
INSERT INTO orders (order_id, order_date, quantity, customer_id, product_id)
VALUES (301, '2024-05-01', 2, 101, 1);

SAVEPOINT sp1;


INSERT INTO orders (order_id, order_date, quantity, customer_id, product_id)
VALUES (302, '2024-05-02', 1, 102, 2);

SAVEPOINT sp2;


INSERT INTO orders (order_id, order_date, quantity, customer_id, product_id)
VALUES (303, '2024-05-03', 5, 103, 3);

SAVEPOINT sp3;

ROLLBACK TO sp2;

COMMIT;
```

Q.6)

Report on the Use of Transaction Logs for Data Recovery

1. Introduction

A transaction log is a critical component of a database management system (DBMS) that records all changes made to the database. It stores information about each transaction, such as when it starts, the operations performed, and whether it is committed or rolled back. Transaction logs play a vital role in ensuring data durability, consistency, and recovery.

2. Purpose of Transaction Logs

The main purposes of transaction logs are:

- To support data recovery after system failures
- To ensure durability of committed transactions
- To enable rollback of incomplete or failed transactions
- To maintain database consistency during crashes

Every change made to the database is first written to the transaction log before being applied to the actual data files. This principle is known as Write-Ahead Logging (WAL).

3. Role of Transaction Logs in Data Recovery

When a system crashes or shuts down unexpectedly, the database may be left in an inconsistent state. During recovery, the DBMS uses the transaction log to:

Redo all committed transactions that were not fully written to disk

Undo all uncommitted transactions that were active at the time of failure

This ensures that the database is restored to a consistent state as if only fully completed transactions had occurred.

4. Hypothetical Data Recovery Scenario

Scenario Description

Consider an online shopping system that stores orders and product inventory in a database. Multiple users are placing orders simultaneously.

At 10:30 AM, the system experiences an unexpected power failure while transactions are being processed.

Transactions in Progress

Transaction T1 (Committed)

START TRANSACTION
INSERT INTO orders (order_id, product_id, quantity) VALUES (501, 10, 2)
UPDATE products SET stock = stock - 2 WHERE product_id = 10
COMMIT

Transaction T2 (Uncommitted)

START TRANSACTION
INSERT INTO orders (order_id, product_id, quantity) VALUES (502, 12, 1)
UPDATE products SET stock = stock - 1 WHERE product_id = 12
-- System crashes before COMMIT

When the system restarts, the DBMS performs recovery using the transaction log:

- Redo Phase:
  - Transaction T1 is marked as committed.
  - Its changes are reapplied if they were not written to disk.
- Undo Phase:
  - Transaction T2 has no COMMIT record.
  - All changes made by T2 are rolled back using the log.

Final Database State

- Order 501 exists in the database
- Stock for product 10 is reduced correctly
- Order 502 does not exist
- Stock for product 12 remains unchanged
- The database is restored to a consistent and correct state.

Conclusion

Transaction logs are essential for ensuring reliable data recovery in database systems. By recording all changes and using write-ahead logging, a DBMS can recover from unexpected failures without data loss or corruption. The use of transaction logs guarantees the atomicity and durability properties of transactions, making them a fundamental component of modern database systems.