Student: Alberto Asensio

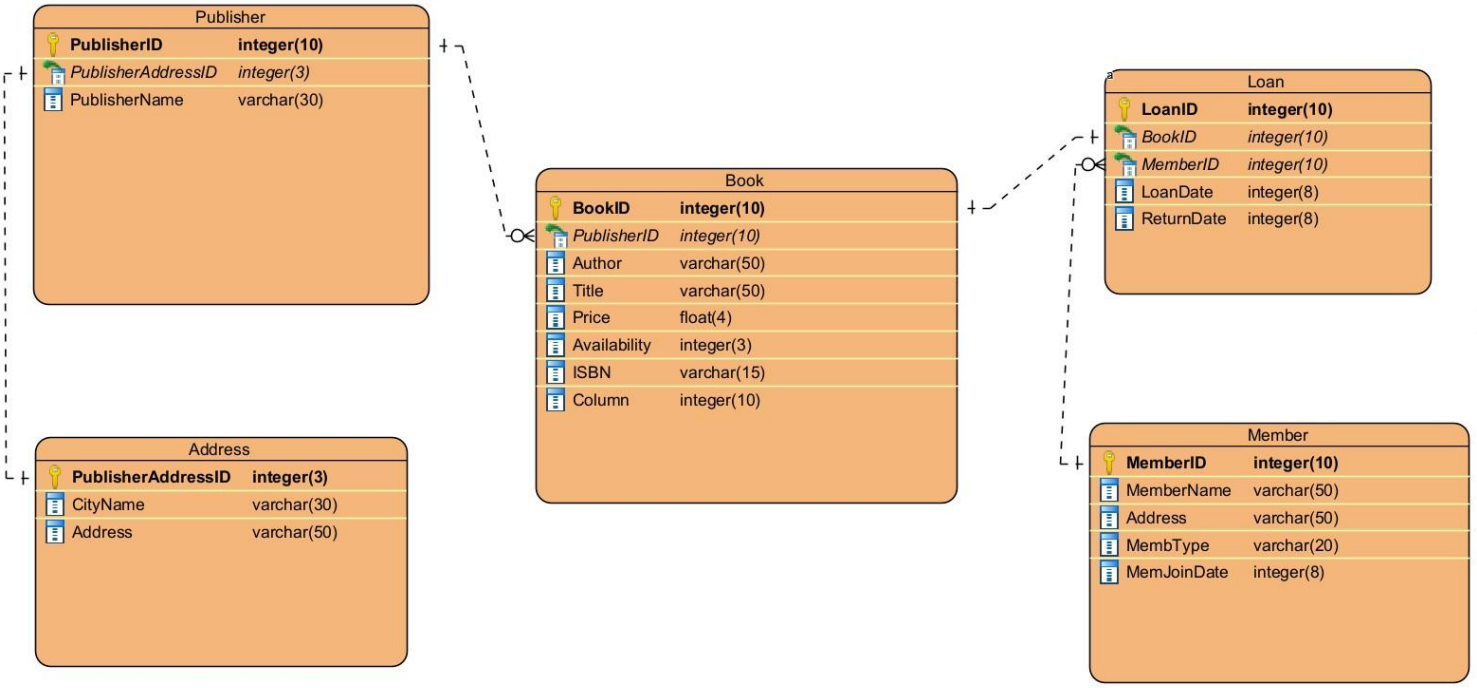
Module: Introduction to Databases

Student Number NCI: x23171782

TABA

Index

1. Question 1: Creating the Relational Model Diagram



Relationships

The database schema contains five tables: Publisher, Address, Book, Member, and Loan, demonstrating various relationships that outline the interconnections between entities.

1. **Publisher and Address (One-to-One Relationship):**

The Publisher table has a one-to-one association with the Address table through the PublisherAddressID in Publisher linked to the AddressID in Address. Each Publisher entry corresponds only to an Address entry, making a strict one-to-one relationship.

2. **Book and Publisher (One-to-Many Relationship):**

The Book table shares a one-to-many relationship with the Publisher table. While a book record corresponds to a single publisher, a single publisher can relate to multiple books. This link is established via the PublisherID, which is a foreign key in the table Publisher.

3. **Loan and Member (Many-to-One Relationship):**

A many-to-one relationship exists between Loan and Member tables. Multiple books can be borrowed by the same member, but different members can't have the same loan. This way, one member can borrow more than one book and have more than one loan. The MemberID attribute acts as a foreign key in the Loan table.

Normal Forms

1. **First Normal Form (1NF)**

I ensured that each table cell holds atomic (indivisible) values. This means no multi-valued or composite attributes are present in the tables, what is essential for normalisation. Therefore, in this diagram each column contains single, indivisible values, without arrays, lists, or multiple values, which satisfies the 1NF criteria.

2. **Second Normal Form (2NF)**

In this relational model diagram, I made sure to remove partial dependencies within a table to comply with the 2NF. A table meets the 2NF criteria if it's in 1NF and if all non-prime attributes are fully functionally dependent on the primary key. Specifically, attributes like Author, Title, Price, Availability, and ISBN in the Book table, and MemberName, Address, MembType, MemJoinDate in the Member table, appear to be functionally dependent on their respective primary keys (BookID and MemberID), satisfying the 2NF conditions.

3. Third Normal Form (3NF)

I ensured that there are no transitive dependencies among non-prime attributes to meet the 3NF criteria. The schema seems to satisfy 3NF since there are no evident transitive dependencies among non-prime attributes in the provided tables.

Structured the tables in alignment with normalization principles provided substantial **advantages**. It significantly enhanced the overall integrity of our data by mitigating anomalies encountered during insertion, update, and deletion operations. Additionally, this approach substantially minimized redundancy, optimizing storage efficiency and streamlining the maintenance process. Furthermore, it notably improved the overall structural organization, facilitating more straightforward data retrieval and manipulation procedures.

Moreover, this meticulous structuring effectively avoided several **problems**. It eliminated the occurrence of update anomalies, ensuring consistency and accuracy while modifying redundant data. The design prevented insertion anomalies, guaranteeing smooth integration of new data without disruptions caused by structural inefficiencies. Lastly, it prevented issues where essential information could be accidentally lost when deleting data. This ensured that the database stayed intact and complete, preserving its reliability and completeness.

2. Question 2: Creating the database

DLL code to create the table
CREATE TABLE Publisher (**PublisherID int(10) NOT NULL,** **PublisherAddressID int(3) NOT NULL,** **PublisherName varchar(30) NOT NULL,** **PRIMARY KEY (PublisherID),** **CONSTRAINT FK_Publisher_Address FOREIGN KEY (PublisherAddressID) REFERENCES Address(PublisherAddressID)** **);** **CREATE TABLE Member (** **MemberID int(10) NOT NULL,** **MemberName varchar(50) NOT NULL,** **Address varchar(50) NOT NULL,** **MembType varchar(20) NOT NULL,** **MemJoinDate int(8) NOT NULL,** **PRIMARY KEY (MemberID)** **);** **CREATE TABLE Address (** **PublisherAddressID int(3) NOT NULL,** **CityName varchar(30) NOT NULL,** **Address varchar(50) NOT NULL,** **PRIMARY KEY (PublisherAddressID)** **);** **CREATE TABLE Loan (** **LoanID int(10) NOT NULL,** **BookID int(10) NOT NULL,** **MemberID int(10) NOT NULL,** **LoanDate int(8) NOT NULL,** **ReturnDate int(8) NOT NULL,** **PRIMARY KEY (LoanID),** **CONSTRAINT FK_Loan_Book FOREIGN KEY (BookID) REFERENCES Book(BookID),** **CONSTRAINT FK_Loan_Member FOREIGN KEY (MemberID) REFERENCES Member(MemberID)** **);** **CREATE TABLE Book (** **BookID int(10) NOT NULL,** **PublisherID int(10) NOT NULL,** **Author varchar(50) NOT NULL,** **Title varchar(50) NOT NULL,** **Price float NOT NULL,** **Availability int(3) NOT NULL,** **ISBN varchar(15) NOT NULL,**

```
PRIMARY KEY (BookID),
CONSTRAINT FK_Book_Publisher FOREIGN KEY (PublisherID) REFERENCES Publisher(PublisherID)
);
```

Inserting some data to the tables

```
-- Data for Address table
INSERT INTO Address (PublisherAddressID, CityName, Address) VALUES
(1, 'City A', '111 Publisher St'),
(2, 'City B', '222 Publisher St'),
(3, 'City C', '333 Publisher St');

-- Data for Publisher table
INSERT INTO Publisher (PublisherID, PublisherAddressID, PublisherName) VALUES
(1, 1, 'Publisher A'),
(2, 2, 'Publisher B'),
(3, 3, 'Publisher C');

-- Data for Member table
INSERT INTO Member (MemberID, MemberName, Address, MembType, MemJoinDate) VALUES
(1, 'John Doe', '123 Main St', 'Regular', 20230101),
(2, 'Jane Smith', '456 Elm St', 'Premium', 20230215),
(3, 'David Johnson', '789 Oak St', 'Regular', 20230320);

-- Data for Book table
INSERT INTO Book (BookID, PublisherID, Author, Title, Price, Availability, ISBN) VALUES
(1, 1, 'Author 1', 'Book Title 1', 25.99, 5, 'ISBN12345'),
(2, 2, 'Author 2', 'Book Title 2', 19.99, 10, 'ISBN67890'),
(3, 3, 'Author 3', 'Book Title 3', 30.50, 3, 'ISBNABCDE');

-- Data for Loan table
INSERT INTO Loan (LoanID, BookID, MemberID, LoanDate, ReturnDate) VALUES
(1, 1, 1, 20230105, 20230120),
(2, 2, 2, 20230220, 20230310),
(3, 3, 3, 20230301, 20230325);
```

Question 3

i. Insert a new record for Book table with all the relevant information.
INSERT INTO Book (BookID, PublisherID, Author, Title, Price, Availability, ISBN) **VALUES (4, 1, 'New Author', 'New Book Title', 50.00, 10, 'NEWISBN');**
ii. Increase the price of the Book by 20% (for the above inserted item).
UPDATE Book SET Price = Price * 1.20 WHERE ISBN = 'NEWISBN';
iii. List the Title of all books that cost 20 or more.
SELECT Title FROM Book WHERE Price >= 20;
iv. Display the names of all members who have a membership joining date in the last week (between your two chosen dates).
SELECT MemberName FROM Member WHERE MemJoinDate BETWEEN 20230313 AND 20230320;
v. List the author, Title, Publisher name where the Book ISBN matches '1292061189'
SELECT b.Author, b.Title, p.PublisherName **FROM Book b** **JOIN Publisher p ON b.PublisherID = p.PublisherID** **WHERE b.ISBN = '1292061189';**
vi. List the loaned Book Titles, Book Loaned/Issued Date, Book Due date, Book Return date and Member name where memberID = 2.
SELECT b.Title, l.LoanDate, l.ReturnDate, m.MemberName **FROM Loan l** **JOIN Book b ON l.BookID = b.BookID** **JOIN Member m ON l.MemberID = m.MemberID** **WHERE l.MemberID = 2;**

Question 4

Non-relational databases, commonly known as NoSQL databases, have been designed to handle vast amounts of unstructured or semi-structured data, offering

distinct advantages and disadvantages in the realms of e-commerce and social media when compared to traditional relational databases.

Pros and Cons of Non-Relational Databases in E-commerce:

Pros:

- ***Performance***: Non-relational databases provide faster read and write operations compared to their relational counterparts. In e-commerce, rapid access to product details, inventory data, and customer profiles is crucial. For instance, MongoDB, a document-oriented NoSQL database, excels in quickly retrieving diverse product information or customer records due to its flexible document model.

- ***Scalability***: E-commerce platforms often encounter fluctuating traffic during sales events or holidays. Non-relational databases, such as Cassandra, a wide-column store NoSQL database, specialize in horizontal scalability. Suppose an e-commerce platform faces a sudden surge in users during a promotional period. Cassandra's capability to scale across multiple nodes seamlessly ensures uninterrupted service despite the increased load.

- ***Flexibility***: NoSQL databases efficiently handle unstructured or semi-structured data. For instance, in e-commerce, product catalogs encompass various data types like images, descriptions, and user reviews. NoSQL databases such as Couchbase or DynamoDB allow storing these diverse data types in a single database, streamlining data management and retrieval.

Cons:

- ***Consistency***: Some NoSQL databases compromise strong consistency for scalability. Imagine an e-commerce scenario where maintaining real-time inventory counts is crucial. However, in databases following an eventual consistency model, brief inconsistencies across nodes might occur, affecting accurate inventory tracking or transaction management.

Pros and Cons of Non-Relational Databases in Social Media:

Pros:

- ***Performance***: Non-relational databases excel in efficiently storing and retrieving complex relationships within social networks. For example, graph databases like Neo4j are ideal for social media platforms, enabling quick traversal of user connections, facilitating friend recommendations, or identifying user communities within a network.

- ***Scalability***: Social media platforms handle massive data volumes. NoSQL databases like Apache HBase or Redis, tailored for high-throughput and low-latency access, enable seamless scaling to accommodate growing user bases and increasing data loads.

- ***Reliability***: In social media applications, maintaining data reliability poses a challenge due to NoSQL databases' eventual consistency models. For instance, ensuring immediate updates across all database nodes when a user likes a post can be difficult. Such inconsistencies may impact real-time analytics or user notifications based on recent interactions.

Types of Non-Relational Databases and Suitability:

1. **Document-oriented Databases** (e.g., MongoDB, Couchbase): Ideal for storing diverse product details, customer information, and order history in e-commerce due to their flexibility in managing various data structures.

2. **Wide-column Stores** (e.g., Cassandra, ScyllaDB): Wide-column stores like Cassandra and ScyllaDB are apt choices for social media platforms due to their ability to efficiently manage user-generated content and activity logs. Their design accommodates scalability needs, ensuring seamless handling of extensive data volumes common in social media interactions. These databases offer the flexibility to scale horizontally across multiple servers, enabling these platforms to sustain high traffic loads and storage demands, crucial for maintaining a smooth user experience amidst growing user interactions and content creation.

3. **Graph Databases** (e.g., Neo4j, ArangoDB): Graph databases such as Neo4j and ArangoDB work good in social media networks by efficiently navigating complex relationships among users, posts, and interactions, thereby facilitating accurate friend suggestions and in-depth network analysis due to their innate graph structure and traversal capabilities.

4. **Key-Value Stores** (e.g., Redis, DynamoDB): Useful for caching frequently accessed data in both e-commerce and social media platforms, improving performance by swiftly retrieving commonly accessed information like session data or popular posts.

In conclusion, while non-relational databases offer advantages in performance, scalability, and flexibility, their suitability in e-commerce and social media contexts varies based on specific use cases. Understanding the exchange, particularly in consistency and

reliability, is crucial in selecting the appropriate NoSQL database type for different application requirements.

Question 5

Database security stands as an essential pillar within relational database systems, serving as the guardian of sensitive information against a myriad of threats. These threats, which encompass unauthorized access, data breaches, manipulation, and loss, can potentially compromise the confidentiality, integrity, and availability of data. To fortify database security, robust countermeasures must be implemented, ensuring data remains shielded from external threats and preserving the integrity of the database.

Threats to Database Security:

1. Unauthorized Access: This threat occurs when individuals gain illicit entry into the database, posing risks of data breaches and manipulations.

2. Data Breaches: These incidents involve the illicit retrieval of sensitive data, jeopardizing confidential information by exposing it to unauthorized entities.

3. Data Manipulation: When data undergoes unauthorized alterations like insertions, modifications, or deletions, it can lead to inaccurate or misleading information.

4. Data Loss: Events such as hardware failures, natural calamities, or human errors can result in the loss of critical data, causing significant disruptions or irretrievable information loss.

Robust Database Security Measures:

1. **Backup and Recovery**: Regularly backing up database content ensures the capability to recover data in case of system failures, corruption, or catastrophic events.
    - Countermeasure: Utilizing RAID technology for fault tolerance by creating redundant data copies across multiple disks, accompanied by automated backup schedules, fortifies data preservation and recovery.

2. **Journaling**: The process of maintaining comprehensive logs of all database changes aids in tracking modifications, supporting recovery, and facilitating forensic analysis in case of security breaches.
    - Countermeasure: Employing secure logging mechanisms to record database activities, ensuring regular review and protection against tampering, bolsters transparency and accountability.

3. **Integrity Measures**: Guaranteeing data accuracy and consistency is achieved by implementing integrity constraints, data validation, and encryption to thwart unauthorized modifications.
    - Countermeasure: Employing encryption methods for sensitive data, implementing validation checks, and deploying access controls help maintain data integrity and prevent unauthorized alterations.

4. Authentication and Authorization: Authentication validates user identities, while authorization regulates user access based on their privileges.
    - Countermeasure: Utilizing robust authentication techniques like unique identifiers, passwords, and multi-factor authentication, coupled with role-based access controls, fortifies defenses against unauthorized access.

5. **SSL and Secure HTTP Protocols:** Encryption protocols such as SSL/TLS secure data transmission, safeguarding data from interception during transit.
    - Countermeasure: Implementing SSL/TLS protocols to encrypt communication channels ensures confidentiality and integrity of data transmitted over networks, reinforcing data protection.

Furthermore, beyond shielding against technical threats, comprehensive database security addresses legal, ethical, and intellectual property concerns. Complying with regulations, preserving data privacy, and safeguarding proprietary information are essential facets of database security. Integrating these multifaceted countermeasures fortifies relational database systems, mitigates potential threats, and ensures the safety of critical data assets.