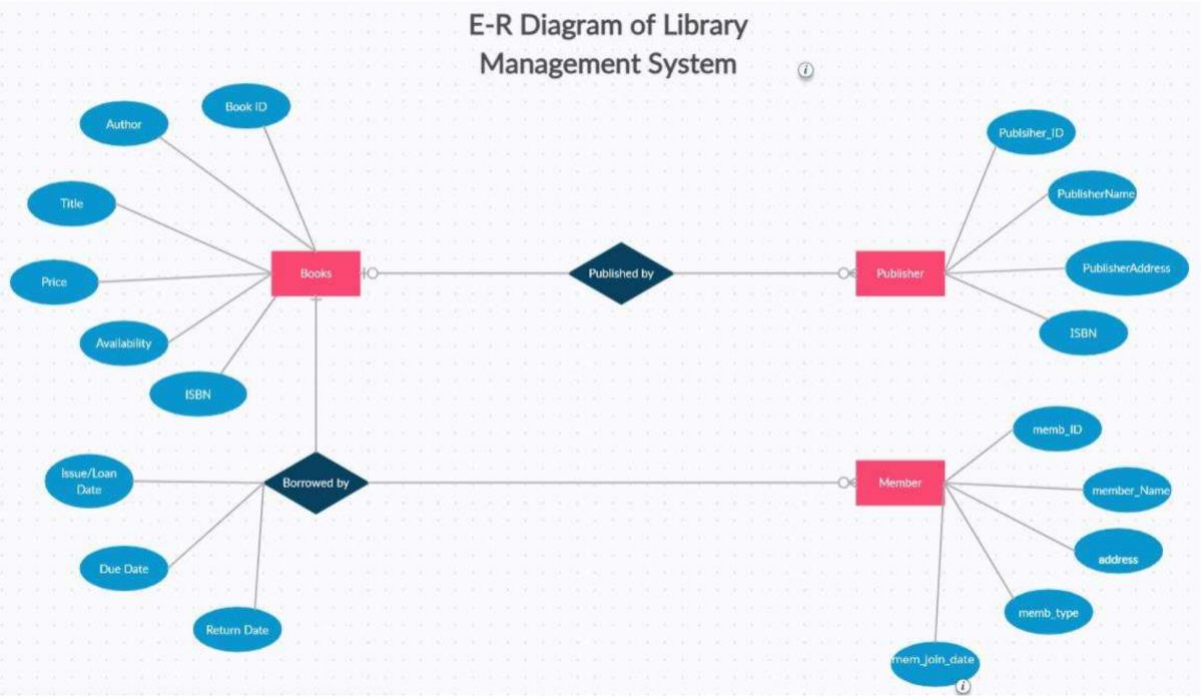
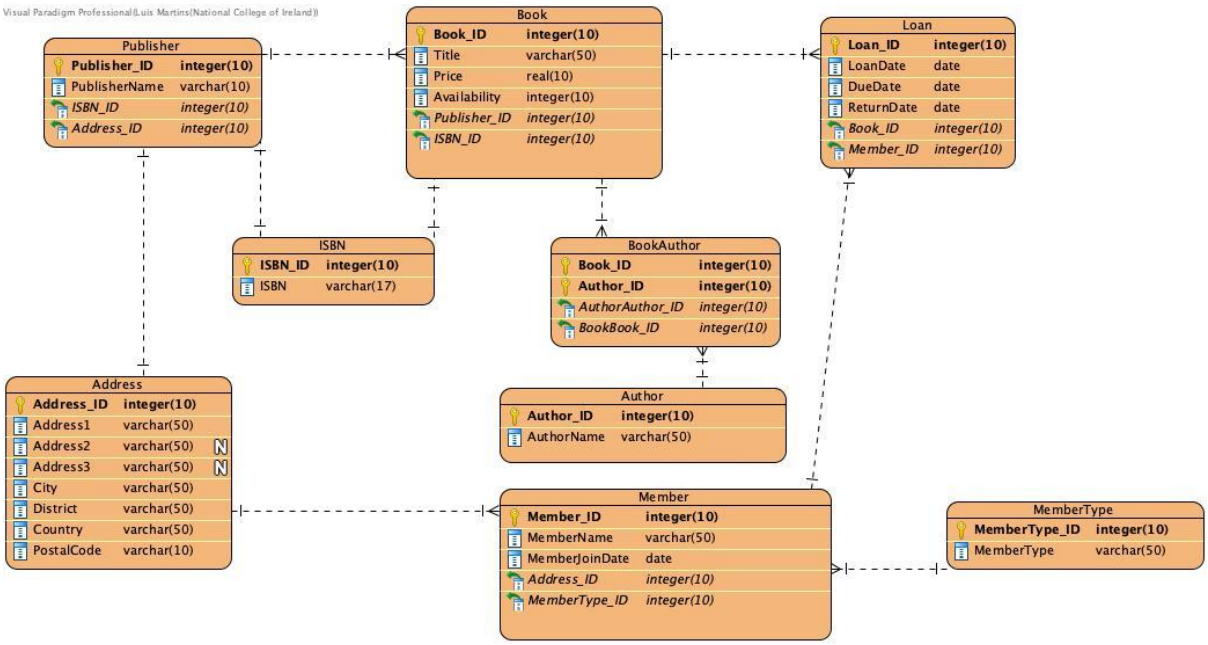
National College of Ireland

Table of Contents

PART A



1. Transform the conceptual design (ER diagram) into a relational model by converting the entities and relationships into their appropriate tables. Show if your tables are normalized using 1st, 2nd, and 3rd normal form.



All tables are all normalized to 3NF.

1NF:

- Contain only atomic values (values that cannot be divided/single values per cell)
- No repeating groups

2NF:

- Meet 1NF
- No Partial Dependencies: Non-key attributes (columns that are not part of the primary key) are fully dependent on the entire primary key, not just a part of it. This is achieved in the schema as all tables have single-attribute primary keys, eliminating the possibility of partial dependencies.

3NF:

- Meet 1NF & 2NF
- No Transitive Dependencies: Non-key attributes do not depend on other non-key attributes. They depend only on the primary key. This is also satisfied in the schema as the tables are structured to avoid transitive dependencies. For example, all the attributes in the “Books” table directly depend on the primary key "Book_ID", rather than depending on each other.

2. Create a database called LMS and convert all the resulting logical tables from question 1 into a physical database design using DDL. Choose the appropriate datatype, primary and foreign keys for the attributes. Fill your table with some data of your choice (you can use Mockaroo to create the data). Provide detailed assumptions for any of your design decisions.

DDL – Data Definition Language

```
CREATE TABLE Book (
Book_ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
Title varchar(50) NOT NULL,
Price real(10) NOT NULL,
Availability integer(10) NOT NULL,
Publisher_ID integer(10) NOT NULL,
ISBN_ID integer(10) NOT NULL,
FOREIGN KEY(Publisher_ID) REFERENCES Publisher(Publisher_ID),
FOREIGN KEY(ISBN_ID) REFERENCES ISBN(ISBN_ID),
UNIQUE (Title, ISBN_ID)
```

```
);

CREATE TABLE BookAuthor (
    Book_ID INTEGER NOT NULL,
    Author_ID INTEGER NOT NULL,
    PRIMARY KEY (Book_ID, Author_ID),
    FOREIGN KEY (Book_ID) REFERENCES Book(Book_ID),
    FOREIGN KEY (Author_ID) REFERENCES Author(Author_ID)
);

CREATE TABLE Publisher (
    Publisher_ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    PublisherName varchar(10) NOT NULL,
    ISBN_ID integer(10) NOT NULL,
    Address_ID integer(10) NOT NULL,
    FOREIGN KEY(Address_ID) REFERENCES Address(Address_ID),
    FOREIGN KEY(ISBN_ID) REFERENCES ISBN(ISBN_ID)
);

CREATE TABLE Member (
    Member_ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    MemberName varchar(50) NOT NULL,
    MemberJoinDate date NOT NULL,
    Address_ID integer(10) NOT NULL,
    MemberType_ID integer(10) NOT NULL,
    FOREIGN KEY(Address_ID) REFERENCES Address(Address_ID),
    FOREIGN KEY(MemberType_ID) REFERENCES MemberType(MemberType_ID),
    UNIQUE (MemberName, Address_ID)
);

CREATE TABLE Loan (
    Loan_ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    LoanDate date NOT NULL,
    DueDate date NOT NULL,
    ReturnDate date NOT NULL,
    Book_ID integer(10) NOT NULL,
    Member_ID integer(10) NOT NULL,
    FOREIGN KEY(Book_ID) REFERENCES Book(Book_ID),
    FOREIGN KEY(Member_ID) REFERENCES Member(Member_ID)
);
```

```
CREATE TABLE Address (
    Address_ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    Address1 varchar(50) NOT NULL,
    Address2 varchar(50),
    Address3 varchar(50),
    City varchar(50) NOT NULL,
    District varchar(50) NOT NULL,
    Country varchar(50) NOT NULL,
    PostalCode varchar(10) NOT NULL,
    UNIQUE (Address1, Address2, Address3, City, District, PostalCode)
);

CREATE TABLE ISBN (
    ISBN_ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    ISBN varchar(17) NOT NULL UNIQUE
);

CREATE TABLE MemberType (
    MemberType_ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    MemberType varchar(50) NOT NULL
);

CREATE TABLE Author (
    Author_ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    AuthorName varchar(50) NOT NULL
);
```

DML – Data Manipulation Language

```
-- Insert data into Address table
INSERT INTO Address (Address1, Address2, Address3, City, District, Country, PostalCode)
VALUES ('123 Main Street', 'Apt 4', NULL, 'Cityville', 'District A', 'Country X', '12345'),
       ('456 Oak Avenue', NULL, NULL, 'Townsville', 'District B', 'Country Y', '56789');

-- Insert data into ISBN table
INSERT INTO ISBN (ISBN)
VALUES ('978-1-123456-78-9'), ('978-2-234567-89-0'), ('978-2-234567-89-2');

-- Insert data into MemberType table
```

```sql
INSERT INTO MemberType (MemberType)
VALUES ('Regular'), ('Premium');

-- Insert data into Author table
INSERT INTO Author (AuthorName)
VALUES ('John Doe'), ('Jane Smith');

-- Insert data into Publisher table
INSERT INTO Publisher (PublisherName, ISBN_ID, Address_ID)
VALUES ('ABC Publications', 1, 1), ('XYZ Press', 2, 2);

-- Insert data into Book table
INSERT INTO Book (Title, Price, Availability, Publisher_ID, ISBN_ID)
VALUES ('Introduction to SQL', 29.99, 50, 1, 1),
    ('Data Science Essentials', 39.99, 25, 2, 2),
    ('The Art of Programming', 45.00, 30, 1, 3);

-- Insert data into BookAuthor
INSERT INTO BookAuthor (Book_ID, Author_ID)
VALUES (1, 1),
    (2, 2),
    (3, 1),
    (3, 2);

-- Insert data into Member table
INSERT INTO Member (MemberName, MemberJoinDate, Address_ID, MemberType_ID)
VALUES ('Alice Johnson', '2023-01-15', 1, 1),
    ('Bob Smith', '2022-05-20', 2, 2);

-- Insert data into Loan table
INSERT INTO Loan (LoanDate, DueDate, ReturnDate, Book_ID, Member_ID)
VALUES ('2023-02-01', '2023-02-15', '2023-02-10', 1, 1),
    ('2022-06-10', '2022-06-25', '2022-06-20', 2, 2);
```

Design decisions

Table Structure:

- Separate tables for entities: This model assumes that books, authors, publishers, members, and addresses have distinct properties and relationships, using separate tables to maintain data integrity and avoid redundancy.
- Composite primary key for BookAuthor: This choice reflects the understanding that a book can have multiple authors, and a unique combination of book ID and author ID is necessary for accurate representation.
- Normalization: The database adheres to normalization principles (up to third normal form) to minimize redundancy, ensure data consistency, and simplify updates.

Data Types:

- INTEGER for primary keys: Primary keys will be numerical and unique, providing efficient indexing and retrieval.
- VARCHAR for text fields: This accommodates varying lengths of text data, such as book titles, author names, and addresses.
- REAL for price: This allows for fractional values to represent book prices accurately.
- DATE for dates: This ensures consistent storage and handling of date-related information, such as member join dates and loan dates.

Foreign Keys:

- Enforcing relationships: Foreign keys are used to create relationships/links between tables, ensuring referential integrity and preventing inconsistencies. For example, a book's publisher ID must refer to a valid publisher in the Publisher table.

Constraints:

- NOT NULL: These constraints enforce mandatory fields.
- UNIQUE: This constraint enforces unique values within designated columns, ensuring data integrity and preventing duplicates.

3. Using your database created in the previous step, answer the following questions using DML statements:

I. Insert a new record for Book table with all the relevant information.

```
INSERT INTO Book (Title, Price, Availability, Publisher_ID, ISBN_ID)
VALUES ('The New Book', 15.99, 5, 1, 2);
```

II. Increase the price of the Book by 20% (for the above inserted item).

```
UPDATE Book
SET Price = Price * 1.20
WHERE Book_ID = (SELECT MAX(Book_ID) FROM Book);
```

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 MemberJoinDate BETWEEN '2023-01-01' AND '2023-12-31';
```

V. List the author, Title, Publisher name where the Book ISBN matches '1292061189'

```
SELECT Author.AuthorName, Book.Title, Publisher.PublisherName
FROM Book
JOIN BookAuthor ON BookAuthor.Book_ID = Book.Book_ID
JOIN Author ON Author.Author_ID = BookAuthor.Author_ID
JOIN Publisher ON Publisher.Publisher_ID = Book.Publisher_ID
WHERE Book.ISBN_ID = (SELECT ISBN_ID FROM ISBN WHERE ISBN =
'1292061189');
```

VI. List the loaned Book Titles, Book Loaned/Issued Date, Book Due date, Book return date and Member name where member_id = 2.

```
SELECT Book.Title AS BookTitle, Loan.LoanDate AS LoanedDate,
Loan.DueDate, Loan.ReturnDate, Member.MemberName
FROM Loan
JOIN Book ON Book.Book_ID = Loan.Book_ID
JOIN Member ON Member.Member_ID = Loan.Member_ID
WHERE Member.Member_ID = 2;
```

PART B

4. Discuss the pros and cons of non-relational databases by choosing one of the contexts below. Explain your answers with examples. Discuss the various types of non-relational databases along with the scenarios suitable to use and avoid them.

Context C: Covid-19 contact tracing or data from health apps/IoT devices;

The rapid-growing health tech industry, pushed by wearables and health apps, generates massive amounts of data. As individuals embrace a data-driven approach to health and wellness, wearables and health apps create an ever-growing ecosystem of interconnected devices and data streams. From fitness trackers monitoring activity levels to smart thermometers recording temperatures, the amount and variety of this data presents a unique challenge for managing and extracting valuable insights.

Advantages of NoSQL for Health Apps and IoT Devices:

- Flexible Data Structures: Unlike relational databases with rigid schemas, NoSQL accommodates the diverse data structures generated by health apps and IoT devices. i.e. sensor readings; GPS logs; sleep activity logs; etc… NoSQL stores them all in flexible document-like formats, adapting effortlessly to new devices and tracking methods. This flexibility allows developers to quickly incorporate new data sources and features without schema modifications.
- Scalability for Explosive Growth: The exponential growth of user adoption and device integration needs a database capable of scaling seamlessly. NoSQL's horizontal scalability, achieved by adding more nodes, ensures efficient data management even as data volumes surge. This ensures smooth performance and real-time insights, regardless of user base expansion.
- Rapid Development and Adaptation: NoSQL's simpler data models and fewer constraints accelerate development cycles, enabling faster deployment of new features like personalised workout plans or sleep-tracking insights. NoSQL allows developers to deliver these functionalities faster.
- Real-Time Data Analysis for Proactive Wellness: Traditional databases often lag behind in data analysis, impeding timely insights. NoSQL excels at real-time data processing.

Limitations of NoSQL for Health Apps and IoT Devices:

- Data Integrity Concerns: Not all NoSQL databases guarantee ACID properties (Atomicity, Consistency, Isolation, Durability) necessary for data integrity. This can raise concerns in sensitive healthcare scenarios where ensuring data accuracy and reliability is essential.
- Complex Analysis: While NoSQL shines in real-time data processing, in-depth analysis of complex data relationships may require the structured nature of

relational databases. NoSQL may not be good for complex joins and aggregations needed for comprehensive health data analysis.

- Development and Maintenance Complexity: Having many different options of NoSQL databases, with different data models and query languages, presents challenges in choosing the right one for each application. Additionally, the lack of standardised development practices can increase maintenance complexity compared to relational databases.

Types of NoSQL databases include:

Document Databases (MongoDB):

- Suitable for:
    - Storing diverse health data with varying structures (e.g., sensor readings, medical records, fitness logs).
    - Handling rapid schema evolution as new devices and data types emerge.
    - Enabling real-time personalisation and recommendations based on user health data.
- Avoid for:
    - Applications requiring strict ACID compliance for critical medical records.
    - Complex queries involving complex joins across multiple data types.

2. Key-Value Stores (Redis):

- Suitable for:
    - Caching frequently accessed health data for fast retrieval (e.g., user profiles, recent activity).
    - Implementing session management for health apps with high-volume user interactions.
- Avoid for:
    - Storing large, complex datasets that require extensive querying and analysis.
    - Handling relationships between different data entities.

3. Graph Databases (Neo4j):

- Suitable for:
    - Modelling relationships between health entities (e.g., patients, medications, symptoms, diagnoses).

    - Conducting research on disease networks and drug interactions.
- Avoid for:
    - Simple storage and retrieval of key-value pairs.
    - Datasets with few inherent relationships.

4. Column-Based Stores (Apache Cassandra):

- Suitable for:
    - Handling massive volumes of time-series health data (e.g., sensor readings, continuous glucose monitoring).
    - Enabling fast writes and horizontal scalability for growing datasets.
    - Supporting analytics and trend analysis over historical health data.
- Avoid for:
    - Applications requiring strong consistency guarantees for critical health data updates.
    - Complex queries involving multiple attributes and relationships.

5. Discuss why Database Security matters in the context of relational database systems. State the various types of database security threats and explain how it can be prevented using database security counter measures. List and explain any of the five commonly used security counter measures to ensure database security.

In today's data-driven world, information is king. Organisations are made out of it: customer records; financial transactions; intellectual property; etc…

Information this that is stored in relational databases, that are an essential component of any modern business. A single breach can cause a snowball effect of problems with an unpredictable magnitude such as: financial losses; operational disruptions; reputational damage; legal repercussions; etc…

That are many threats that target databases, these threats come in various forms: hackers employing brute force or deception; phishing scams posing as trusted sources; dissatisfied insiders with internal access; outside individuals exploiting software weaknesses; etc…

Their goals usually are either financial gain, or if working for a governmental/private organisation, advantage or leverage against their target.

This is why prioritising database security is not just a prudent measure, but an absolute necessity for any organisation holding sensitive information.

To combat these threats, robust countermeasures need to be implemented. These measures safeguard data and ensure continuousness in the face of potential attacks:

- Authentication

  Authentication establishes whether a user is who they claim to be. This involves verifying unique identifiers and credentials, typically through passwords or multi-factor authentication. Making it significantly more difficult for attackers to breach the system.

- Authorization

  Once identity is confirmed, authorization determines what a user can do within the system. This involves assigning roles and permissions, granting specific rights and privileges to access or manipulate data. By enforcing roles and permissions, it prevents unauthorized individuals from accessing sensitive information or performing unauthorized operations.

- Data Encryption

  Data encryption is important to keep information safe. Using strong encryption algorithms, masking data, turns it unreadable to unauthorized individuals even if it is breached. This ensures that data remains secure, both at rest and in transit.

- Journaling/Logging and Auditing

  Logging and auditing systems includes maintaining a detailed record of user activity and database operations. This log provides valuable insights for investigations, allowing the identification of suspicious behaviour and tracing potential threats back to their source. Such detailed historical data is important for incident response and proactive threat mitigation.

- Backup and Recovery

  Recognising the potential for unexpected events, backup and recovery strategies provide a safety net for critical data. These measures ensure rapid restoration of information in case of an attack or accidental loss, minimising downtime and mitigating potential damage. By ensuring data availability even in the face of adversity, organisations can maintain operational continuity and avoid disruptions.

In conclusion, it is worth to note that database security is an ongoing process of vigilance and adaptation, that must adapt to the dynamic and evolving nature of cyber threats, through continuous monitoring and proactive countermeasures