DATABASE DESIGN: CONCEPTUAL AND LOGICAL MODELS

INFT-1111: Assignment 1

Abstract

In this assignment, we are designing and developing an Entity-Relationship (ER) model, a conceptual model, and a logical model for two case studies. The goal is to practice the skills needed to translate real-world scenarios into database designs.

Maheep Isher Singh Chawla
100909435

9-Nov-24

Table of Contents

Case Study 1: Library Management System

Conceptual Model

Business Definition Table

Entity	Attributes	Description
Member	MemberID (PK), Name, Address, Phone, Email	Individuals who are registered to borrow items from the library.
MembershipType	MembershipTypeID (PK), TypeName, Benefits	Different types of memberships available (e.g., Adult, Child, Senior).
MemberMembership	MemberID (PK), MembershipTypeID (PK)	This table links Members to the Membership types that they own.
Book	BookID (PK), Title, ISBN, PublicationYear, GenreID (FK)	Books those are available in the library.
Magazine	MagazineID (PK), Title, IssueNumber, PublicationDate, GenreID (FK)	Magazines that are available in the library.
Genre	GenreID (PK), GenreName	Categories by which books and magazines are classified.
Author	AuthorID (PK), FirstName, LastName, Biography	Authors who have written books that are available in the library.
Reservation	ReservationID (PK), MemberID (FK), ItemID, ReservationDate	Records of items reserved by members.
Checkout	CheckoutID (PK), MemberID (FK), ItemID, CheckoutDate, DueDate	Records of items currently checked out by members.
Item	ItemID (PK), Title, Type (Book/Magazine), GenreID (FK)	General representation of library items (books and magazines).
BookAuthor	BookID (FK), AuthorID (FK)	Junction table representing the many-to-many relationship between books and authors.

Relationships

- A Member can be related to many MembershipTypes.
- A member may be able to reserve many items, but he or she can only check out up to 5 items at a time.
- A Book has one Genre it belongs to and is written by several Authors.
- One Magazine is one Genre.
- Both Members and Items are associated with Reservations and Checkouts.

Constraints

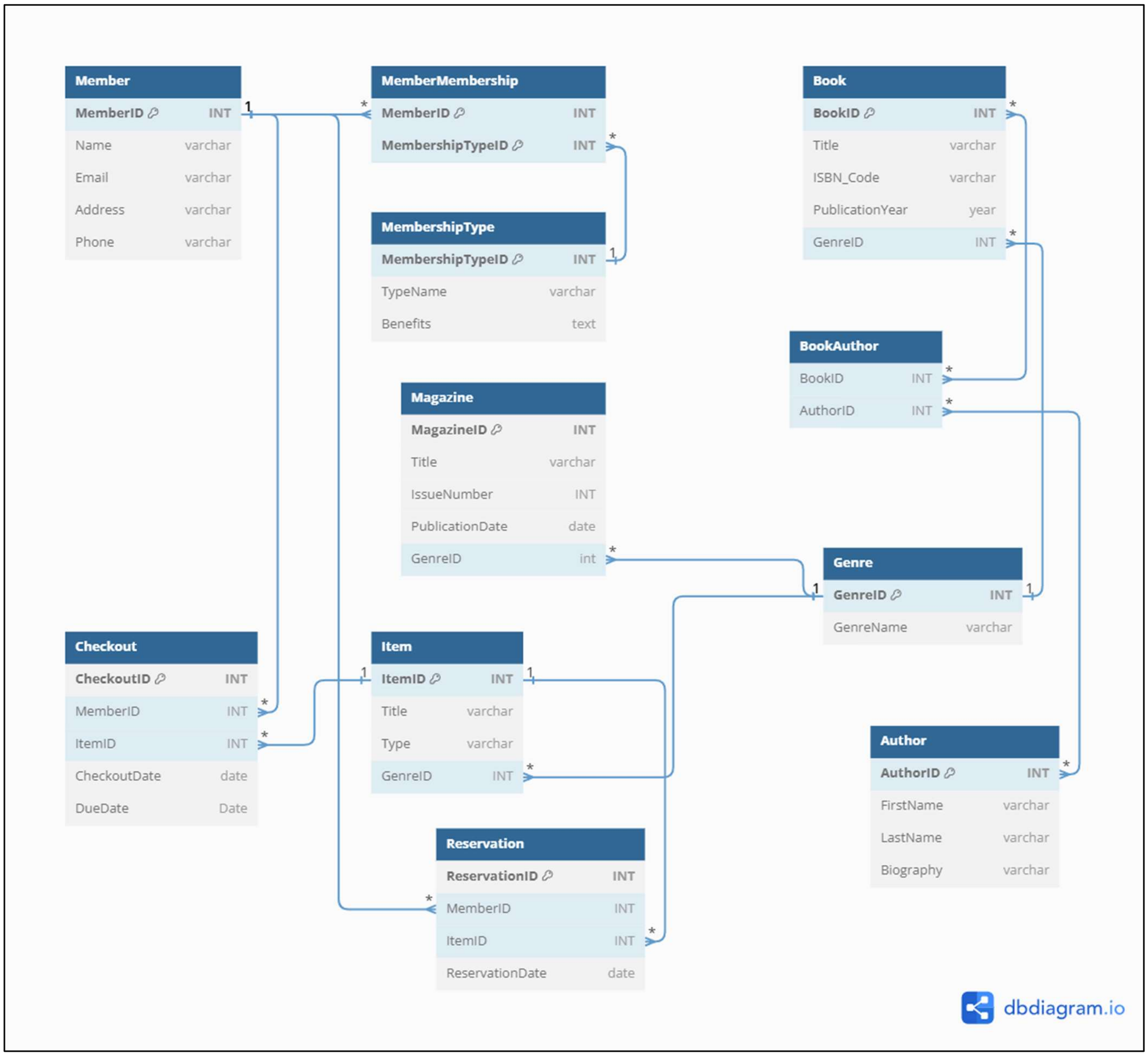
- A Member can have a maximum of 5 active Checkouts.
- ItemType must be either 'Book' or 'Magazine'.

Entity-Relationship Model

Relationships and Cardinality

- Member to MembershipType: One-to-Many (A member can have multiple membership types)
- Member to Reservation: One-to-Many (A member can have multiple reservations)
- Member to Checkout: One-to-Many (A member can have up to 5 checkouts)
- Book to Genre: Many-to-One (A book belongs to one genre)
- Magazine to Genre: Many-to-One (A magazine belongs to one genre)
- Book to Author: Many-to-Many via BookAuthor
- Reservation to Item: Many-to-One (A reservation is for one item)
- Checkout to Item: Many-to-One (A checkout is for one item)

ER Diagram



Member
MemberID INT
Name varchar
Email varchar
Address varchar
Phone varchar
MemberMembership
MemberID INT
MembershipTypeID INT
MembershipType
MembershipTypeID INT
TypeName varchar
Benefits text
Book
BookID INT
Title varchar
ISBN_Code varchar
PublicationYear year
GenreID INT
BookAuthor
BookID INT
AuthorID INT
Magazine
MagazineID INT
Title varchar
IssueNumber INT
PublicationDate date
GenreID int
Genre
GenreID INT
GenreName varchar
Checkout
CheckoutID INT
MemberID INT
ItemID INT
CheckoutDate date
DueDate Date
Item
ItemID INT
Title varchar
Type varchar
GenreID INT
Reservation
ReservationID INT
MemberID INT
ItemID INT
ReservationDate date
Author
AuthorID INT
FirstName varchar
LastName varchar
Biography varchar
1
*
dbdiagram.io

Logical Model: SQL Table Creation Scripts

Table: Genre

```sql
CREATE TABLE Genre (
   GenreID INT PRIMARY KEY,
   GenreName VARCHAR(100) NOT NULL
);
```

Table: Member

```sql
CREATE TABLE Member (
   MemberID INT PRIMARY KEY,
   Name VARCHAR(255) NOT NULL,
   Address VARCHAR(255),
   Phone VARCHAR(20),
   Email VARCHAR(100) UNIQUE
);
```

Table: MembershipType

```sql
CREATE TABLE MembershipType (
   MembershipTypeID INT PRIMARY KEY,
   TypeName VARCHAR(50) NOT NULL,
   Benefits TEXT
);
```

Table: MemberMembership

```sql
CREATE TABLE MemberMembership (
   MemberID INT,
   MembershipTypeID INT,
   PRIMARY KEY (MemberID, MembershipTypeID),
   FOREIGN KEY (MemberID) REFERENCES Member(MemberID),
   FOREIGN KEY (MembershipTypeID) REFERENCES MembershipType(MembershipTypeID)
);
```

Table: Item

```sql
CREATE TABLE Item (
   ItemID INT PRIMARY KEY,
   Title VARCHAR(255) NOT NULL,
   Type VARCHAR(20) CHECK (Type IN ('Book', 'Magazine')),
   GenreID INT,
   FOREIGN KEY (GenreID) REFERENCES Genre(GenreID)
);
```

Table: Book

```sql
CREATE TABLE Book (
   BookID INT PRIMARY KEY,
   Title VARCHAR(255) NOT NULL,
   ISBN VARCHAR(20) UNIQUE,
   PublicationYear YEAR,
   GenreID INT,
   FOREIGN KEY (GenreID) REFERENCES Genre(GenreID)
);
```

);

Table: Magazine

```
CREATE TABLE Magazine (
    MagazineID INT PRIMARY KEY,
    Title VARCHAR(255) NOT NULL,
    IssueNumber INT,
    PublicationDate DATE,
    GenreID INT,
    FOREIGN KEY (GenreID) REFERENCES Genre(GenreID)
);
```

Table: Author

```
CREATE TABLE Author (
    AuthorID INT PRIMARY KEY,
    FirstName VARCHAR(100),
    LastName VARCHAR(100),
    Biography TEXT
);
```

Table: BookAuthor

```
CREATE TABLE BookAuthor (
    BookID INT,
    AuthorID INT,
    PRIMARY KEY (BookID, AuthorID),
    FOREIGN KEY (BookID) REFERENCES Book(BookID),
    FOREIGN KEY (AuthorID) REFERENCES Author(AuthorID)
```

Table: Reservation

```
CREATE TABLE Reservation (
    ReservationID INT PRIMARY KEY,
    MemberID INT,
    ItemID INT,
    ReservationDate DATE,
    FOREIGN KEY (MemberID) REFERENCES Member(MemberID),
    FOREIGN KEY (ItemID) REFERENCES Item(ItemID)
);
```

Table: Checkout

```
CREATE TABLE Checkout (
    CheckoutID INT PRIMARY KEY,
    MemberID INT,
    ItemID INT,
    CheckoutDate DATE,
    DueDate DATE,
    FOREIGN KEY (MemberID) REFERENCES Member(MemberID),
    FOREIGN KEY (ItemID) REFERENCES Item(ItemID),
    CONSTRAINT chk_max_checkouts CHECK (
        (SELECT COUNT(*) FROM Checkout WHERE MemberID = Checkout.MemberID) <= 5
    )
);    --member can  checkout a maximum of 5 items
```

Case Study 2: College Registration System

Conceptual Model

Business Definition Table

Entity	Attributes	Description
Student	StudentID (PK), FirstName, LastName, Email, Phone, MajorID (FK), MinorID (FK)	Individuals enrolled in the college.
Professor	ProfessorID (PK), FirstName, LastName, Email, DepartmentID (FK)	Faculty members who teach courses.
Course	CourseID (PK), CourseName, CourseCode, DepartmentID (FK)	Courses offered by the college.
Department	DepartmentID (PK), DepartmentName	Academic departments within the college.
Enrollment	EnrollmentID (PK), StudentID (FK), CourseID (FK), EnrollmentDate	Records of students enrolled in courses.
Prerequisite	CourseID (FK), PrerequisiteCourseID (FK)	Courses that are prerequisites for other courses.
CourseOffering	OfferingID (PK), CourseID (FK), ProfessorID (FK), Semester, Year	Specific offerings of courses taught by professors in a semester.
Major	MajorID (PK), DepartmentID (FK), MajorName	Majors available to students, associated with departments.
Minor	MinorID (PK), DepartmentID (FK), MinorName	Minors available to students, associated with departments.

Relationships

- Student may have one Major and optionally one Minor.
- Student enrolls in multiple Courses via Enrollment.
- Course is offered by one Department and may have multiple Prerequisites.
- Professor belongs to one Department and teaches multiple CourseOfferings.
- CourseOffering links Course and Professor for specific semesters.

Constraints

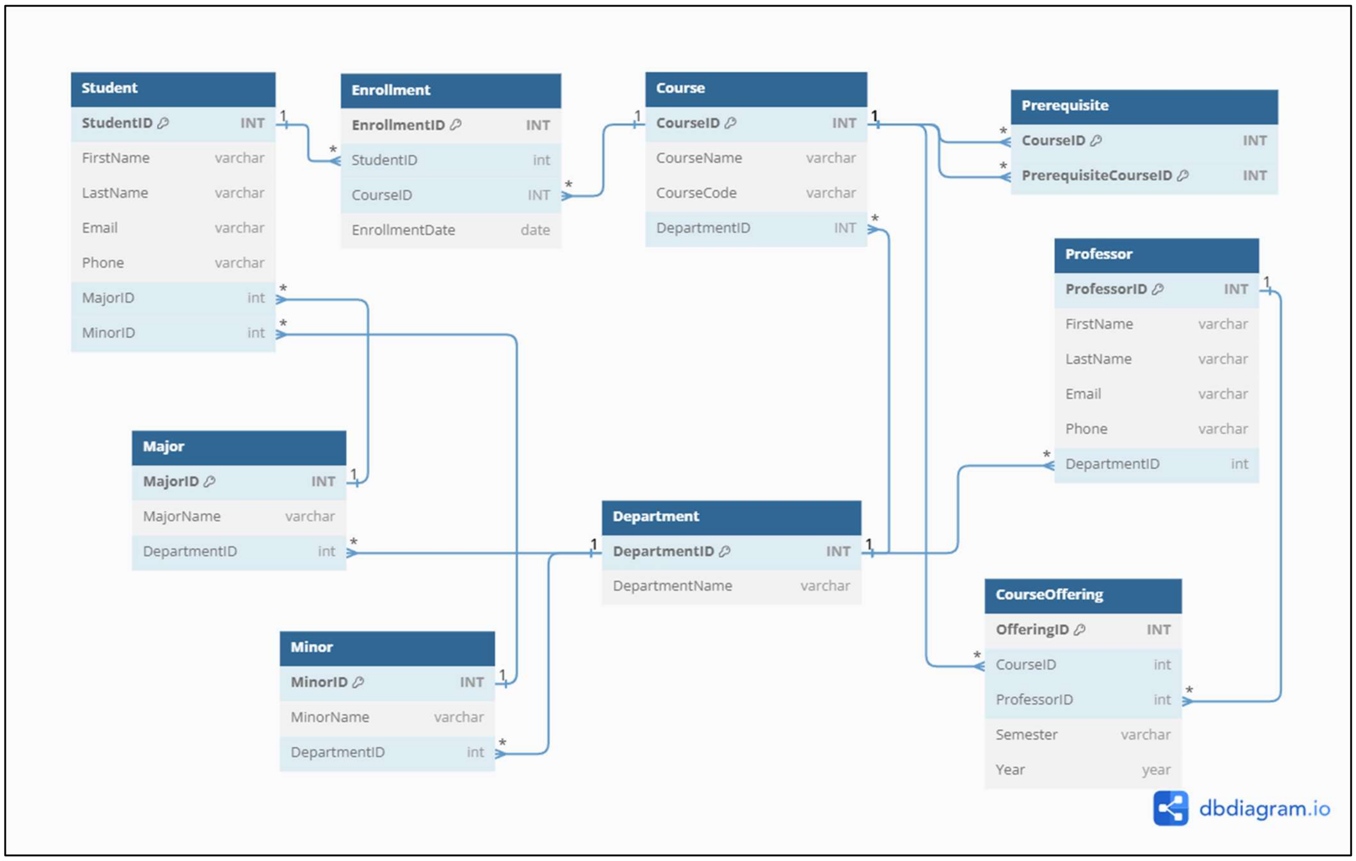
- A Course may have multiple Prerequisites.
- Student must satisfy all Prerequisites before enrolling in a Course (enforcement may require additional logic beyond the database schema).

Entity-Relationship Model

Relationships and Cardinality

- Student to Major: Many-to-One, one major per student.
- Student to Minor: Many-to-One; a student may have one minor.
- Student to Enrollment: One-to-Many; a student can enroll in multiple courses.
- Course to Department: Many-to-One; a course must be part of one department.
- Professor to Department: Many-to-One; a professor must be in one department.
- Professor to CourseOffering: One-to-Many; one professor can teach multiple course offerings.
- Course to Prerequisite: One-to-Many; a course can have multiple prerequisites.
- Course to CourseOffering: One-to-Many (One course can have multiple offerings)
- Enrollment links Student and Course

ER Diagram



Logical Model: SQL Table Creation Scripts

Table: Department

CREATE TABLE Department (

DepartmentID INT **PRIMARY KEY**,

DepartmentName VARCHAR(100) NOT NULL,

OfficeLocation VARCHAR(100)

);

Table: Major

CREATE TABLE Major (

MajorID INT **PRIMARY KEY**,

DepartmentID INT,

MajorName VARCHAR(100) NOT NULL,

FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID)

);

Table: Minor

```
CREATE TABLE Minor (
    MinorID INT PRIMARY KEY,
    DepartmentID INT,
    MinorName VARCHAR(100) NOT NULL,
    FOREIGN KEY (DepartmentID) REFERENCES
Department(DepartmentID)
);
```

Table: Student

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(100) NOT NULL,
    LastName VARCHAR(100) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    Phone VARCHAR(20),
    MajorID INT,
    MinorID INT,
    FOREIGN KEY (MajorID) REFERENCES
Major(MajorID),
    FOREIGN KEY (MinorID) REFERENCES
Minor(MinorID)
);
```

Table: Professor

```
CREATE TABLE Professor (
    ProfessorID INT PRIMARY KEY,
    FirstName VARCHAR(100) NOT NULL,
    LastName VARCHAR(100) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES
Department(DepartmentID)
);
```

Table: Course

```
CREATE TABLE Course (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(255) NOT NULL,
    CourseCode VARCHAR(20) UNIQUE NOT NULL,
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES
Department(DepartmentID)
);
```

Table: Prerequisite

```
CREATE TABLE Prerequisite (
    CourseID INT,
    PrerequisiteCourseID INT,
    PRIMARY KEY (CourseID, PrerequisiteCourseID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID),
    FOREIGN KEY (PrerequisiteCourseID) REFERENCES Course(CourseID)
);
--combination of CourseID and PrerequisiteCourseID is the primary key
```

Table: Enrollment

```
CREATE TABLE Enrollment (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT,
    CourseID INT,
    EnrollmentDate DATE,
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
);
```

Table: CourseOffering

```
CREATE TABLE CourseOffering (
    OfferingID INT PRIMARY KEY,
    CourseID INT,
    ProfessorID INT,
    Semester VARCHAR(20),
    Year YEAR,
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID),
    FOREIGN KEY (ProfessorID) REFERENCES Professor(ProfessorID)
);
```