LESSON 9

Defining Relationships and Constraints

Ihor Liutak

Content

Primary and Foreign Keys, Cascading Updates/Deletes

Relationships define how tables are connected in a relational database.

Why Relationships?

Ensure data consistency.

Reduce redundancy.

Allow complex queries across related tables.

A real-world analogy of relationships:

Table 1: Students

(e.g., name, ID)

Table 2: Courses

(e.g., course name, ID)

Primary Keys

Definition:

A unique identifier for each row in a table.

Characteristics:

Must be unique.

Cannot be NULL.

Syntax:

```
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);
```

```
CREATE TABLE Books (
    BookID INT PRIMARY KEY,
    Title VARCHAR(100),
    Author VARCHAR(50)
);
```

Foreign Keys

Definition: A column (or set of columns) that establishes a relationship between two tables.

Purpose: To enforce referential integrity.

Syntax:

```
CREATE TABLE Enrollments (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT,
    CourseID INT,
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID)
);
```

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

Cascading Updates

Definition: Automatically updates foreign key values when the primary key is updated.

Purpose: Maintain referential integrity without manual updates.

Syntax:

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
    ON UPDATE CASCADE
);
```

```
-- Update CustomerID in Customers
UPDATE Customers
SET CustomerID = 1010
WHERE CustomerID = 1001;

-- The change automatically cascades to the Orders table.
```

Cascading Deletes

Definition: Automatically deletes rows in a foreign key table when the corresponding primary key is deleted.

Syntax:

```
CREATE TABLE Enrollments (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT,
    CourseID INT,
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID)
    ON DELETE CASCADE
);
```

```
-- Delete StudentID from Students
DELETE FROM Students
WHERE StudentID = 123;

-- Cascades to Enrollments and removes related records.
```

Common Constraints

NOT NULL: Ensures a column cannot have a NULL value.

UNIQUE: Ensures all values in a column are unique.

DEFAULT: Provides a default value for a column.

CHECK: Ensures values in a column meet a condition.

```sql
-- NOT NULL Example
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(50) NOT NULL
);
```

```sql
-- UNIQUE Example
CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    Email VARCHAR(100) UNIQUE
);

-- CHECK Example
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    Price DECIMAL(10, 2) CHECK (Price > 0)
);
```

Example

We need to design an **Entity-Relationship Model (ERD)** for a library system, taking into account the following:

Key Entities and Relationships:

Students borrow books from the library.
Each book can be borrowed by one student at a time, but a student can borrow multiple books.
A student might violate return rules by not returning books on time.
The system must record information about these violations (e.g., date, type of violation, and penalties).

Functional Requirements:

Track book borrowing activities.
Record violations of book return deadlines.
Notify students and library authorities about overdue books.
Enforce relationships, including cardinalities and constraints (e.g., one-to-many, many-to-one).

Database Constraints:

Use specialization for defining specific roles (e.g., overdue borrowers).
Ensure referential integrity using primary and foreign keys.

Identify Entities and Attributes

Students:

- `StudentID` (Primary Key)
- `FirstName`
- `LastName`
- `Email`
- `PhoneNumber`

Books:

- `BookID` (Primary Key)
- `Title`
- `Author`
- `ISBN`
- `AvailableCopies`

Borrowing Records:

- `BorrowID` (Primary Key)
- `StudentID` (Foreign Key referencing `Students`)
- `BookID` (Foreign Key referencing `Books`)
- `BorrowDate`
- `ReturnDate`
- `Status` (`Returned`, `Overdue`)

Violations:

- `ViolationID` (Primary Key)
- `BorrowID` (Foreign Key referencing `BorrowingRecords`)
- `ViolationDate`
- `PenaltyAmount`

Define Relationships

Students → BorrowingRecords:

A student can have multiple borrowing records (1:N relationship).

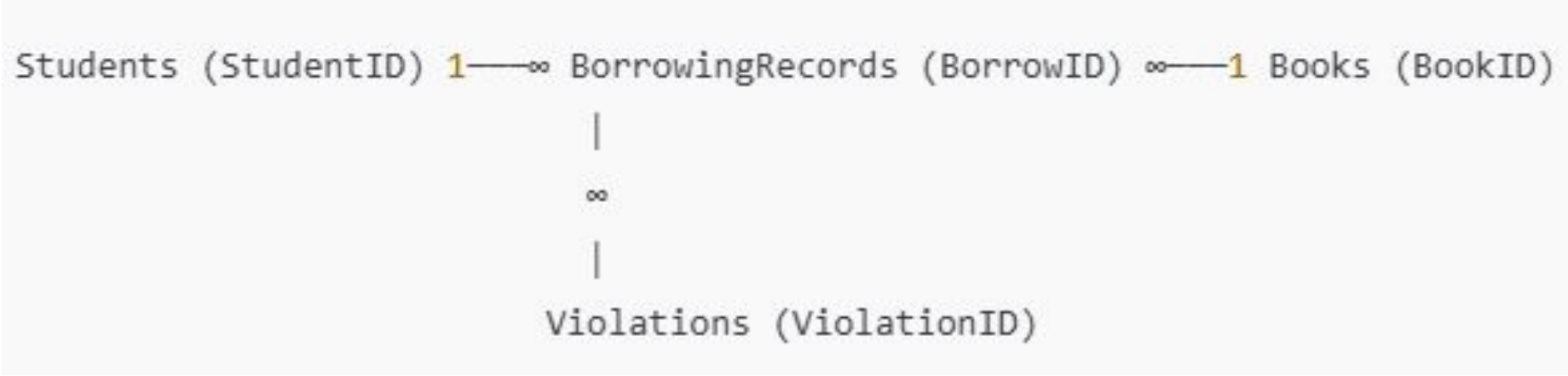
Books → BorrowingRecords:

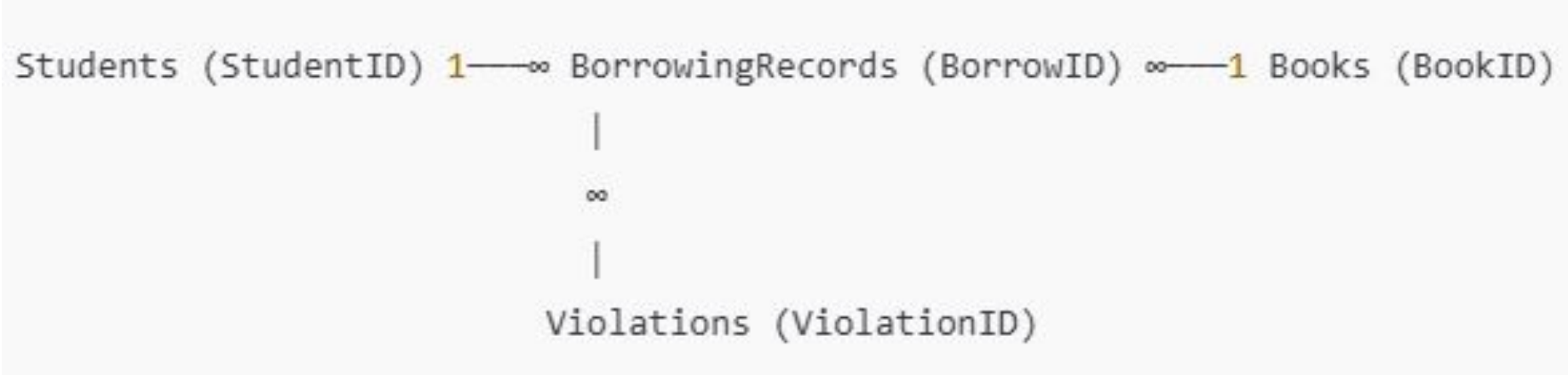
A book can appear in multiple borrowing records but only one active borrowing at a time (1:N relationship).

BorrowingRecords → Violations:

A borrowing record can have zero or more violations (1:N relationship).

Entity-Relationship Model (ERD)




Students (StudentID) 1——∞ BorrowingRecords (BorrowID) ∞——1 Books (BookID)
∞
Violations (ViolationID)

MySQL Schema

```sql
CREATE TABLE Students (
    StudentID INT AUTO_INCREMENT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Email VARCHAR(100) UNIQUE,
    PhoneNumber VARCHAR(15)
);
```

```sql
CREATE TABLE Books (
    BookID INT AUTO_INCREMENT PRIMARY KEY,
    Title VARCHAR(200),
    Author VARCHAR(100),
    ISBN VARCHAR(20) UNIQUE,
    AvailableCopies INT NOT NULL
);
```

```sql
CREATE TABLE BorrowingRecords (
    BorrowID INT AUTO_INCREMENT PRIMARY KEY,
    StudentID INT,
    BookID INT,
    BorrowDate DATE NOT NULL,
    ReturnDate DATE,
    Status ENUM('Returned', 'Overdue') DEFAULT 'Returned',
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID) ON DELETE CASCADE,
    FOREIGN KEY (BookID) REFERENCES Books(BookID) ON DELETE CASCADE
);
```

```sql
CREATE TABLE Violations (
    ViolationID INT AUTO_INCREMENT PRIMARY KEY,
    BorrowID INT,
    ViolationDate DATE NOT NULL,
    PenaltyAmount DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (BorrowID) REFERENCES BorrowingRecords(BorrowID) ON DELETE CASCADE
);
```

Example Queries

```sql
INSERT INTO Students (FirstName, LastName, Email, PhoneNumber)
VALUES ('John', 'Doe', 'johndoe@example.com', '1234567890');
```

```sql
INSERT INTO Books (Title, Author, ISBN, AvailableCopies)
VALUES ('1984', 'George Orwell', '9780451524935', 5);
```

```sql
INSERT INTO BorrowingRecords (StudentID, BookID, BorrowDate)
VALUES (1, 1, CURDATE());
```

```sql
INSERT INTO Violations (BorrowID, ViolationDate, PenaltyAmount)
VALUES (1, CURDATE(), 10.00);
```

```sql
SELECT b.Title, s.FirstName, s.LastName, br.BorrowDate, br.ReturnDate
FROM BorrowingRecords br
JOIN Books b ON br.BookID = b.BookID
JOIN Students s ON br.StudentID = s.StudentID
WHERE br.Status = 'Overdue';
```

Find All Overdue Books