Principles of Software Engineering and Data Bases

Davide Yi Xian Hu
Email: davideyi.hu@polimi.it
Date: 19 November 2024
Exercise Lecture: 04 - Logical Design

POLITECNICO
MILANO 1863

Exercise 1 - Library Management System

Design a database for a library management system to keep track of books, authors, and library members.
The database should store the following information:

- 👉 **Books**: Each book has a unique ID, title, publication year, and genre. A book may have one or more authors.
- 👉 **Authors**: Each author has a unique ID, first name, last name, and year of birth.
- 👉 **Library Members**: Each member has a unique ID, name, address, and membership start date.
- 👉 A member can borrow multiple books, and the database should record the borrowing date and the due date for each borrowed book.

Exercise 1 - Library Management System

Book (book_id, title, publication_year, genre)
Author (author_id, first_name, last_name, birth_year)
Member (member_id, name, address, membership_date)
BookBorrowing (book_id, member_id, borrowing_date, due_date)
BookAuthor (book_id, *author_id*)

Exercise 1 - Library Management System

```
-- Authors Table
CREATE TABLE Author (
    author_id INTEGER PRIMARY KEY,
    first_name TEXT,
    last_name TEXT,
    birth_year INTEGER
);

-- Books Table
CREATE TABLE Book (
    book_id INTEGER PRIMARY KEY,
    title TEXT,
    publication_year INTEGER,
    genre TEXT
);
```

Exercise 1 - Library Management System

```
-- Library Members Table
CREATE TABLE Member (
    member_id INTEGER PRIMARY KEY,
    name TEXT,
    address TEXT,
    membership_date DATE
);
```

Exercise 1 - Library Management System

```
-- BookAuthor Table
CREATE TABLE BookAuthor (
    book_id INTEGER,
    author_id INTEGER,
    PRIMARY KEY (book_id, author_id),
    FOREIGN KEY (book_id) REFERENCES Book(book_id),
    FOREIGN KEY (author_id) REFERENCES Author(author_id)
);

-- Book Borrowing Table
CREATE TABLE BookBorrowing (
    member_id INTEGER,
    book_id INTEGER,
    borrowing_date DATE,
    due_date DATE,
    PRIMARY KEY (member_id, book_id, borrowing_date),
    FOREIGN KEY (member_id) REFERENCES Member(member_id),
    FOREIGN KEY (book_id) REFERENCES Book(book_id)
);
```

Exercise 1 - Library Management System

Query

Retrieve the titles and genres of all books published after 2010.

Exercise 1 - Library Management System

Query

Retrieve the titles and genres of all books published after 2010.

SQL

```
SELECT title, genre
FROM Book
WHERE publication_year > 2010;
```

Exercise 1 - Library Management System

Query

Find the names of authors
who have written books in the 'Science Fiction' genre.

Exercise 1 - Library Management System

Query

Find the names of authors

who have written books in the 'Science Fiction' genre.

SQL

```
SELECT DISTINCT Author.first_name, Author.last_name
FROM Author
JOIN BookAuthor ON Author.author_id = BookAuthor.author_id
JOIN Book ON Book.book_id = BookAuthor.book_id
WHERE Book.genre = 'Science Fiction';
```

Exercise 1 - Library Management System

Query

Search the member that borrowed most books.

Exercise 1 - Library Management System

Query

Search the member that borrowed most books.

SQL

```
SELECT Member.name, COUNT(*) AS total_books_borrowed
FROM Member
JOIN BookBorrowing ON Member.member_id = BookBorrowing.member_id
GROUP BY Member.member_id, Member.name
ORDER BY total_books_borrowed DESC
LIMIT 1;
```

Exercise 2 - Movie Screening Application

Design a database for an application related to movie scheduling in cinemas. The database must store the following information:

- 👉 **Movies**: title, genre, director, duration, and release date.
- 👉 **Cinemas**: name, city, address, and the number of seats.
- 👉 **Actors**: first name, last name, age, phone, and acting style.
- 👉 Relationships:
  - 🔎 Each movie has one and only one director.
  - 🔎 Each movie can have zero or more actors.
  - 🔎 Each movie can be scheduled at one or more cinemas.

Exercise 2 - Movie Screening Application

Movie (movie_id, title, genre, director, duration, release_date)
Cinema (cinema_id, name, city, address, number_of_seats)
Actor (actor_id, first_name, last_name, age, phone, acting_style)
MovieActor (movie_id, actor_id)
MovieCinema (movie_id, cinema_id)

Exercise 2 - Movie Screening Application

```sql
-- Movie Table
CREATE TABLE Movie (
    movie_id INTEGER PRIMARY KEY,
    title TEXT NOT NULL,
    genre TEXT,
    director TEXT NOT NULL,
    duration INTEGER,
    release_date DATE
);

-- Cinema Table
CREATE TABLE Cinema (
    cinema_id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    city TEXT,
    address TEXT,
    number_of_seats INTEGER
);
```

Exercise 2 - Movie Screening Application

```
-- Actor Table
CREATE TABLE Actor (
    actor_id INTEGER PRIMARY KEY,
    first_name TEXT NOT NULL,
    last_name TEXT NOT NULL,
    age INTEGER,
    phone TEXT,
    acting_style TEXT
);
```

Exercise 2 - Movie Screening Application

```sql
-- MovieActor Table
CREATE TABLE MovieActor (
    movie_id INTEGER,
    actor_id INTEGER,
    PRIMARY KEY (movie_id, actor_id),
    FOREIGN KEY (movie_id) REFERENCES Movie(movie_id),
    FOREIGN KEY (actor_id) REFERENCES Actor(actor_id)
);

-- MovieCinema Table
CREATE TABLE MovieCinema (
    movie_id INTEGER,
    cinema_id INTEGER,
    PRIMARY KEY (movie_id, cinema_id),
    FOREIGN KEY (movie_id) REFERENCES Movie(movie_id),
    FOREIGN KEY (cinema_id) REFERENCES Cinema(cinema_id)
);
```

Exercise 2 - Movie Screening Application

Query

Find all movies directed by director "Christopher Nolan".

Exercise 2 - Movie Screening Application

Query

Find all movies directed by director "Christopher Nolan".

SQL

```
SELECT title, genre, release_date
FROM Movie
WHERE director = 'Christopher Nolan';
```

Exercise 2 - Movie Screening Application

Query

Find movies that have more than 10 actors.

Exercise 2 - Movie Screening Application

Query

Find movies that have more than 10 actors.

SQL

```
SELECT Movie.title, COUNT(MovieActor.actor_id) AS actor_count
FROM Movie
JOIN MovieActor ON Movie.movie_id = MovieActor.movie_id
GROUP BY Movie.movie_id, Movie.title
HAVING COUNT(MovieActor.actor_id) > 10;
```

Exercise 2 - Movie Screening Application

Query

For each actor,

calculate the total time they have worked with every other actor.

Exercise 2 - Movie Screening Application

Query

For each actor,
calculate the total time they have worked with every other actor.

SQL

```
SELECT MA1.actor_id AS actor_1_id,
       MA2.actor_id AS actor_2_id,
       SUM(Movie.duration) AS total_time_worked
FROM Movie
JOIN MovieActor MA1 ON Movie.movie_id = MA1.movie_id
JOIN MovieActor MA2 ON Movie.movie_id = MA2.movie_id
WHERE MA1.actor_id ≠ MA2.actor_id
  AND MA1.actor_id < MA2.actor_id
GROUP BY MA1.actor_id, MA2.actor_id;
```

Exercise 3 - University Course

Design a database to track students, the courses they enroll in, and the grades they receive for each enrollment. Additionally, the database should record which professor teaches each course for a specific term.

- 👉 **Students**: student_id, first_name, last_name, email, year_of_enrollment.
- 👉 **Courses**: course_id, course_name, credits.
- 👉 **Professors**: professor_id, first_name, last_name, department.
- 👉 **Enrollments**: grade, term, and year.

Exercise 3 - University Course

Student (student_id, first_name, last_name, email, year_of_enrollment)
Course (course_id, course_name, credits)
Professor (professor_id, first_name, last_name, department)
Enrollment (student_id, course_id, professor_id, grade, term, year)

Exercise 4 - Normal Form

First Normal Form (1NF)

removes repeated groups from a table to guarantee atomicity.

Second Normal Form (2NF)

lessens redundancy by eliminating partial dependencies.

Third Normal Form (3NF)

reduces data duplication by removing transitive dependencies.

Exercise 4 - Normal Form

First Normal Form (1NF)

removes repeated groups from a table to guarantee atomicity.

Example (Before 1NF):

StudentID	Name	Courses
1	Alice	Math, Physics

After 1NF:

StudentID	Name	Course
1	Alice	Math
1	Alice	Physics

Exercise 4 - Normal Form

Second Normal Form (2NF)

lessens redundancy by eliminating partial dependencies.

Example (Before 2NF):

OrderID	ProductID	ProductName	Quantity
1	101	Pen	10

After 2NF: Orders Table:

OrderID	ProductID	Quantity
1	101	10

Products Table:

ProductID	ProductName
101	Pen

Exercise 4 - Normal Form

Third Normal Form (3NF)

reduces data duplication by removing transitive dependencies.

Example (Before 3NF):

StudentID	Name	Course	InstructorName
1	Alice	Math	Dr. Smith

After 3NF: **Students Table**:

StudentID	Name
1	Alice

Courses Table:

Course	InstructorName
Math	Dr. Smith

Exercise 5 - Normal Orders

Normalize the following table Order.

OrderID	CustomerName	Products	Quantities	Address	City	State
1	Alice	Pen, Notebook	10, 5	123 Main St	Springfield	IL
2	Bob	Pencil, Eraser	20, 15	456 Elm St	Springfield	IL

Exercise 5 - Normal Orders

Normalize the following table Order.

OrderID	CustomerName	Products	Quantities	Address	City	State
1	Alice	Pen, Notebook	10, 5	123 Main St	Springfield	IL
2	Bob	Pencil, Eraser	20, 15	456 Elm St	Springfield	IL

Problems:

- 👉 Products and Quantities are not atomic (contain multiple values).
- 👉 Repeating data in Address, City, and State for each customer.

OrderID	CustomerName	Products	Quantities	Address	City	State
1	Alice	Pen, Notebook	10, 5	123 Main St	Springfield	IL
2	Bob	Pencil, Eraser	20, 15	456 Elm St	Springfield	IL

Exercise 5 - Normal Orders

First Normal Form (1NF)

OrderID	CustomerName	Product	Quantity	Address	City	State
1	Alice	Pen	10	123 Main St	Springfield	IL
1	Alice	Notebook	5	123 Main St	Springfield	IL
2	Bob	Pencil	20	456 Elm St	Springfield	IL
2	Bob	Eraser	15	456 Elm St	Springfield	IL

Exercise 5 - Normal Orders

Second Normal Form (2NF)

1. **Orders Table**:

OrderID	CustomerName	Address	City	State
1	Alice	123 Main St	Springfield	IL
2	Bob	456 Elm St	Springfield	IL

2. **OrderDetails Table**:

OrderID	Product	Quantity
1	Pen	10
1	Notebook	5
2	Pencil	20
2	Eraser	15

Exercise 5 - Normal Orders

Third Normal Form (3NF)

1. **Orders Table**:

OrderID	CustomerName	Address
1	Alice	123 Main St
2	Bob	456 Elm St

2. **Address Table**:

Address	City	State
123 Main St	Springfield	IL
456 Elm St	Springfield	IL

3. **OrderDetails Table** (remains unchanged):

OrderID	Product	Quantity
1	Pen	10
1	Notebook	5
2	Pencil	20
2	Eraser	15