Principles of Database Systems
Assignment #2 - Relation Model

He Tianyang, 3022001441

September 24, 2024

1 Class Assignment

Create 5 tables that show in slides all `PRIMARY KEY` and `FOREIGN KEY` relationships. Also, insert example data that is displayed in the slides. The schema is shown in Fig. 1.

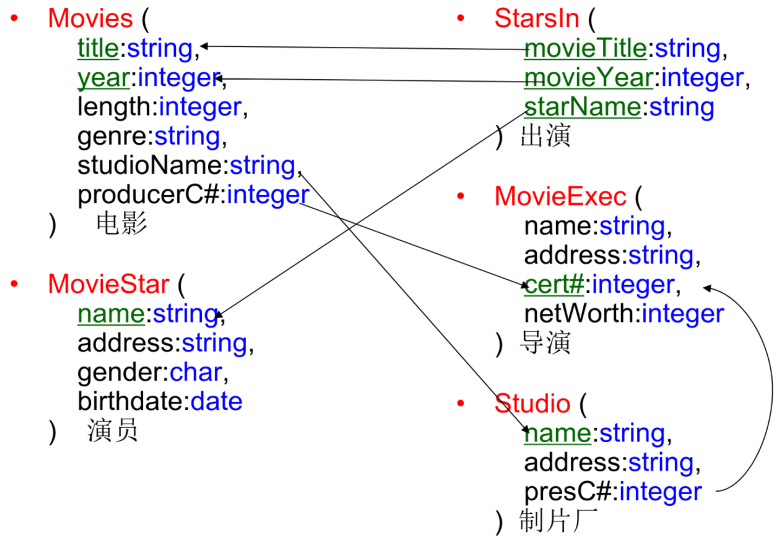

Figure 1: Example Database Schema

Solutions

First, we need to identify the `PRIMARY KEY` and `FOREIGN KEY`. The `PRIMARY KEY`s are:

1. `Movies: title + year`
2. `MovieStar: name`
3. `StarsIn: movieTitle + movieYear + starName`
4. `MovieExec: cert#`
5. `Studio: name`

The `FOREIGN KEY` relationships are:

1. `Movies.studioName` references `Studio.name`
2. `Movies.producerC#` references `MovieExec.cert#`
3. `StarsIn.movieTitle` and `StarsIn.movieYear` reference `Movies.title` and `Movies.year`
4. `StarsIn.starName` references `MovieStar.name`
5. `Studio.presC#` references `MovieExec.cert#`

The SQL table definitions with `PRIMARY KEY` and `FOREIGN KEY` constraints are as follows:

SQL Table Definitions

```
-- 1. Create MovieExec first (no dependencies)
CREATE TABLE MovieExec (
    name VARCHAR(255),
    address VARCHAR(255),
    "cert#" INT,
    netWorth DECIMAL(10, 2),
    PRIMARY KEY ("cert#")
);

-- 2. Create Studio (depends on MovieExec)
CREATE TABLE Studio (
    name VARCHAR(255),
    address VARCHAR(255),
    "presC#" INT,
    PRIMARY KEY (name),
    FOREIGN KEY ("presC#") REFERENCES MovieExec("cert#")
);

-- 3. Create MovieStar (no dependencies)
CREATE TABLE MovieStar (
    name VARCHAR(255),
    address VARCHAR(255),
    gender CHAR(1),
    birthdate DATE,
    PRIMARY KEY (name)
);

-- 4. Create Movies (depends on Studio and MovieExec)
CREATE TABLE Movies (
    title VARCHAR(255),
    year INT,
    length INT,
    genre VARCHAR(50),
```

```
    studioName VARCHAR(255),
    "producerC#" INT,
    PRIMARY KEY (title, year),
    FOREIGN KEY (studioName) REFERENCES Studio(name),
    FOREIGN KEY ("producerC#") REFERENCES MovieExec("cert#")
);

-- 5. Create StarsIn (depends on Movies and MovieStar)
CREATE TABLE StarsIn (
    movieTitle VARCHAR(255),
    movieYear INT,
    starName VARCHAR(255),
    PRIMARY KEY (movieTitle, movieYear, starName),
    FOREIGN KEY (movieTitle, movieYear) REFERENCES Movies(title, year),
    FOREIGN KEY (starName) REFERENCES MovieStar(name)
);
```

Example Data Insertions

```
-- 1. Insert into MovieExec Table
INSERT INTO MovieExec (name, address, "cert#", netWorth)
VALUES ('John Smith', 'Hollywood Blvd, USA', 101, 5000000.00);

-- 2. Insert into Studio Table
INSERT INTO Studio (name, address, "presC#")
VALUES ('Warner Bros', 'California, USA', 101);

-- 3. Insert into MovieStar Table
INSERT INTO MovieStar (name, address, gender, birthdate)
VALUES ('Leonardo DiCaprio', 'LA, USA', 'M', '1974-11-11');

-- 4. Insert into Movies Table
INSERT INTO Movies (title, year, length, genre, studioName, "producerC#")
VALUES ('Inception', 2010, 148, 'Sci-Fi', 'Warner Bros', 101);

-- 5. Insert into StarsIn Table
INSERT INTO StarsIn (movieTitle, movieYear, starName)
VALUES ('Inception', 2010, 'Leonardo DiCaprio');
```

This schema demonstrates the relationships between tables, highlighting the `PRIMARY KEY` and `FOREIGN KEY` constraints and showing how to insert example data accordingly.

Running in MySQL

Here we use `PostgreSQL` as the database, and use `DataGrip` as the client to run the SQL queries. The results are shown in Fig. 2-4.

```
postgres.public> CREATE TABLE MovieExec (
                     name VARCHAR(255),
                     address VARCHAR(255),
                     "cert#" INT,
                     netWorth DECIMAL(10, 2),
                     PRIMARY KEY ("cert#")
                 )
[2024-09-24 15:04:34] completed in 22 ms
postgres.public> CREATE TABLE Studio (
                     name VARCHAR(255),
                     address VARCHAR(255),
                     "presC#" INT,
                     PRIMARY KEY (name),
                     FOREIGN KEY ("presC#") REFERENCES MovieExec("cert#")
                 )
[2024-09-24 15:04:34] completed in 13 ms
postgres.public> CREATE TABLE MovieStar (
                     name VARCHAR(255),
                     address VARCHAR(255),
                     gender CHAR(1),
                     birthdate DATE,
                     PRIMARY KEY (name)
                 )
[2024-09-24 15:04:34] completed in 10 ms
postgres.public> CREATE TABLE Movies (
                     title VARCHAR(255),
                     year INT,
                     length INT,
                     genre VARCHAR(50),
                     studioName VARCHAR(255),
                     "producerC#" INT,
                     PRIMARY KEY (title, year),
                     FOREIGN KEY (studioName) REFERENCES Studio(name),
                     FOREIGN KEY ("producerC#") REFERENCES MovieExec("cert#")
                 )
[2024-09-24 15:04:34] completed in 19 ms
```

Figure 2: Results of SQL Queries

```
postgres.public> CREATE TABLE StarsIn (
                     movieTitle VARCHAR(255),
                     movieYear INT,
                     starName VARCHAR(255),
                     PRIMARY KEY (movieTitle, movieYear, starName),
                     FOREIGN KEY (movieTitle, movieYear) REFERENCES Movies(title, year),
                     FOREIGN KEY (starName) REFERENCES MovieStar(name)
                 )
[2024-09-24 15:04:34] completed in 13 ms
```

Figure 3: Results of SQL Queries

```
[2024-09-24 16:18:58] Connected
postgres> INSERT
          INTO MovieExec (name, address, "cert#", netWorth)
          VALUES ('John Smith', 'Hollywood Blvd, USA', 101, 5000000.00)
[2024-09-24 16:18:58] 1 row affected in 7 ms
postgres> INSERT INTO Studio (name, address, "presC#")
          VALUES ('Warner Bros', 'California, USA', 101)
[2024-09-24 16:18:58] 1 row affected in 9 ms
postgres> INSERT INTO MovieStar (name, address, gender, birthdate)
          VALUES ('Leonardo DiCaprio', 'LA, USA', 'M', '1974-11-11')
[2024-09-24 16:18:58] 1 row affected in 5 ms
postgres> INSERT INTO Movies (title, year, length, genre, studioName, "producerC#")
          VALUES ('Inception', 2010, 148, 'Sci-Fi', 'Warner Bros', 101)
[2024-09-24 16:18:58] 1 row affected in 5 ms
postgres> INSERT INTO StarsIn (movieTitle, movieYear, starName)
          VALUES ('Inception', 2010, 'Leonardo DiCaprio')
[2024-09-24 16:18:58] 1 row affected in 5 ms
```

Figure 4: Results of Data Insertions

2 Exercise 2.2.1

In Fig. 5 are instances of two relations that might constitute part of a banking database. Indicate the following:

1. The attributes of each relation.
2. The tuples of each relation.
3. The components of one tuple from each relation.
4. The relation schema for each relation.
5. The database schema.
6. A suitable domain for each attribute.
7. Another equivalent way to present each relation.

acctNo	*type*	*balance*
12345	savings	12000
23456	checking	1000
34567	savings	25

The relation Accounts

firstName	*lastName*	*idNo*	*account*
Robbie	Banks	901-222	12345
Lena	Hand	805-333	12345
Lena	Hand	805-333	23456

The relation Customers

Figure 5: Two relations of a banking database

Solutions:

1. Attributes

For relation `Accounts`, the attributes are: `acctNo`, `type`, `balance`.
For relation `Customers`, the attributes are: `firstName`, `lastName`, `idNo`, and `account`.

2. Tuples

For relation `Accounts`, the tuples are:

- `(12345, savings, 12000)`
- `(23456, checking, 1000)`
- `(34567, savings, 25)`

For relation `Customers`, the tuples are:

- `(Robbie, Banks, 901-222, 12345)`
- `(Lena, Hand, 805-333, 12345)`
- `(Lena, Hand, 805-333, 23456)`

3. Components of One Tuple from Each Relation

For the relation `Accounts`, one tuple is (12345, savings, 12000).

- `acctNo` = 12345
- `type` = savings
- `balance` = 12000

For the relation `Customers`, one tuple is (Robbie, Banks, 901-222, 12345).

- `firstName` = Robbie
- `lastName` = Banks
- `idNo` = 901-222
- `account` = 12345

4. Relation Schema

The schema for `Accounts` relation is:

```
Accounts(acctNo, type, balance)
```

The schema for `Customers` relation is:

```
Customers(firstName, lastName, idNo, account)
```

5. Database Schema

The database schema consists of the following relations:

```
Accounts(
    acctNo: INTEGER,
    type: STRING,
    balance: DECIMAL
)

Customers(
    firstName: STRING,
    lastName: STRING,
    idNo: STRING,
    account: INTEGER
)
```

6. Suitable Domain for Each Attribute

The suitable domains for the attributes are:

- `acctNo`: integer (e.g., a 5-digit number)
- `type`: string (values like `"savings"` or `"checking"`)
- `balance`: integer or floating point
- `firstName`: string
- `lastName`: string
- `idNo`: string (in a format like a social security number)
- `account`: integer (corresponds to `acctNo`)

7. Another Equivalent Way to Present Each Relation

Another way to represent the `Accounts` relation is in JSON format:

```
[
  {"acctNo": 12345, "type": "savings", "balance": 12000},
  {"acctNo": 23456, "type": "checking", "balance": 1000},
  {"acctNo": 34567, "type": "savings", "balance": 25}
]
```

Similarly, for the `Customers` relation:

```
    [
        {
          "firstName": "Robbie",
          "lastName": "Banks",
          "idNo": "901-222",
          "account": 12345
        },
        {
          "firstName": "Lena",
          "lastName": "Hand",
          "idNo": "805-333",
          "account": 12345
        },
        {
          "firstName": "Lena",
          "lastName": "Hand",
          "idNo": "805-333",
          "account": 23456
        }
     ]
```

3 Exercise 2.3.1

In this exercise we introduce one of our running examples of a relational database schema. The database schema consists of four relations, whose schemas are:

- `Product(maker, model, type)`
- `PC(model, speed, ram, hd, price)`
- `Laptop(model, speed, ram, hd, screen, price)`
- `Printer(model, color, type, price)`

The `Product` relation gives the manufacturer, model number and type (PC, laptop, or printer) of various products. We assume for convenience that model numbers are unique over all manufacturers and product types; that assumption is not realistic, and a real database would include a

code for the manufacturer as part of the model number. The `PC` relation gives for each model number that is a PC the speed (of the processor, in gigahertz), the amount of RAM (in megabytes), the size of the hard disk (in gigabytes), and the price. The `Laptop` relation is similar, except that the screen size (in inches) is also included. The `Printer` relation records for each printer model whether the printer produces color output (true, if so), the process type (laser or ink-jet, typically), and the price.

Write the following declarations:

1. A suitable schema for relation `Product`.
2. A suitable schema for relation `PC`.
3. A suitable schema for relation `Laptop`.
4. A suitable schema for relation `Printer`.
5. An alteration to your `Printer` schema from (4) to delete the attribute `color`.
6. An alteration to your `Laptop` schema from (3) to add the attribute `od` (optical-disk type, e.g., cd or dvd). Let the default value for this attribute be 'none' if the laptop does not have an optical disk.

Solutions

1. Schema of Relation `Product`

```
CREATE TABLE Product (
    maker VARCHAR(255),
    model VARCHAR(255) PRIMARY KEY,
    type VARCHAR(50)
);
```

2. Schema of Relation `PC`

```
CREATE TABLE PC (
    model VARCHAR(255) PRIMARY KEY,
    speed FLOAT,
    ram INT,
    hd INT,
    price DECIMAL(10, 2),
    FOREIGN KEY (model) REFERENCES Product(model)
);
```

3. Schema of Relation `Laptop`

```
CREATE TABLE Laptop (
    model VARCHAR(255) PRIMARY KEY,
    speed FLOAT,
    ram INT,
    hd INT,
```

```
    screen FLOAT,
    price DECIMAL(10, 2),
    FOREIGN KEY (model) REFERENCES Product(model)
);
```

4. Schema of Relation `Printer`

```
CREATE TABLE Printer (
    model VARCHAR(255) PRIMARY KEY,
    color BOOLEAN,
    type VARCHAR(50),
    price DECIMAL(10, 2),
    FOREIGN KEY (model) REFERENCES Product(model)
);
```

5. Alter the `Printer` table to delete the attribute `color`:

```
ALTER TABLE Printer DROP COLUMN color;
```

6. Alter the `Laptop` table to add the attribute `od` with default value 'none'

```
ALTER TABLE Laptop
ADD COLUMN od VARCHAR(50) DEFAULT 'none';
```

4 Exercise 2.3.2

This exercise introduces another running example, concerning World War II capital ships. It involves the following relations:

- `Classes(class, type, country, numGuns, bore, displacement)`
- `Ships(name, class, launched)`
- `Battles(name, date)`
- `Outcomes(ship, battle, result)`

Ships are built in "classes" from the same design, and the class is usually named for the first ship of that class. The relation `Classes` records the name of the class, the type ('bb' for battleship or 'be' for battlecruiser), the country that built the ship, the number of main guns, the bore (diameter of the gun barrel, in inches) of the main guns, and the displacement (weight, in tons). Relation `Ships` records the name of the ship, the name of its class, and the year in which the ship was launched. Relation `Battles` gives the name and date of battles involving these ships, and relation `Outcomes` gives the result (sunk, damaged, or ok) for each ship in each battle.

Write the following declarations:

1. A suitable schema for relation `Classes`.
2. A suitable schema for relation `Ships`.

3. A suitable schema for relation `Battles`.

4. A suitable schema for relation `Outcomes`.

5. An alteration to your `Classes` relation from (a) to delete the attribute `bore`.

6. An alteration to your `Ships` relation from (b) to include the attribute `yard` giving the shipyard where the ship was built.

Solutions

1. Schema for Relation `Classes`:

```
CREATE TABLE Classes (
    class VARCHAR(50) PRIMARY KEY,
    type CHAR(2) CHECK (type IN ('bb', 'be')),
    country VARCHAR(50),
    numGuns INT,
    displacement FLOAT
);
```

2. Schema for Relation `Ships`

```
CREATE TABLE Ships (
    name VARCHAR(50) PRIMARY KEY,
    class VARCHAR(50),
    launched YEAR,
    FOREIGN KEY (class) REFERENCES Classes(class)
);
```

3. Schema for Relation `Battles`

```
CREATE TABLE Battles (
    name VARCHAR(50) PRIMARY KEY,
    date DATE
);
```

4. Schema for Relation `Outcomes`

```
CREATE TABLE Outcomes (
    ship VARCHAR(50),
    battle VARCHAR(50),
    result CHAR(3) CHECK (result IN ('sunk', 'dam', 'ok')),
    PRIMARY KEY (ship, battle),
    FOREIGN KEY (ship) REFERENCES Ships(name),
    FOREIGN KEY (battle) REFERENCES Battles(name)
);
```

5. Alteration to `Classes` to delete the attribute `bore`:

```
ALTER TABLE Classes
DROP COLUMN bore;
```

6. Alteration to `Ships` to include the attribute `yard`

```
ALTER TABLE Ships
ADD yard VARCHAR(50);
```