

# **SQL Case Study for Film Club**

## **Information System**

**Yuanshuo Du**

**D22125495**

# SQL Case Study for Film Club

## 1. Introduction

In this assignment, we will be working with a Film Club Relational Schema, which is designed to capture the information and relationships of a film club company. As the new Database Administrator (DBA) of this company, our task is to finish the rest of the database development for the film club's information system. This case study will test our basic fundamental understanding of SQL language and enhance our skills in using SQL to create tables and relationships, as well as to manipulate data. We will be working with 9 tables, including Member, Film, Performance, Performer, Reserves, Book, StarsIn, Employee, and PreferredMember, and we will be adding relevant relationships to establish the connections between these tables.

## 2. Database Design and Development

First, install MySQL and access its terminal. The first step is to create the Film\_Club database using the "create database Film\_Club" command. If the response is "query OK," it means that the database has been successfully created. Then, use the "use Film\_Club" command to enter and use the newly created database. The "show databases" command can be used to display all current databases.

```
mysql> create database Film_Club;
Query OK, 1 row affected (0.01 sec)

mysql> use Film_Club;
Database changed
mysql> show databases;
+-----+
| Database |
+-----+
| film_club |
| information_schema |
| mysql |
| performance_schema |
| sakila |
| sys |
| world |
+-----+
7 rows in set (0.00 sec)
```

```
mysql> CREATE TABLE Member (
-> MemberID INT PRIMARY KEY,
-> Name VARCHAR(50) NOT NULL,
-> Street VARCHAR(50) NOT NULL,
-> City VARCHAR(50) NOT NULL,
-> State VARCHAR(50) NOT NULL
-> );
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE Film (
-> FilmID INT PRIMARY KEY,
-> Title VARCHAR(100) NOT NULL,
-> PubDate DATE NOT NULL,
-> PurchasePrice DECIMAL(10,2) NOT NULL,
-> Distributor VARCHAR(50) NOT NULL,
-> Kind VARCHAR(50) NOT NULL,
-> RecommendedAge INT NOT NULL,
-> SpokenLanguage VARCHAR(50) NOT NULL,
-> SubtitleLanguage VARCHAR(50) NOT NULL
-> );
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE Performance (
-> FilmID INT NOT NULL,
-> ShowTime DATETIME NOT NULL,
-> Status VARCHAR(50) NOT NULL,
-> PRIMARY KEY (FilmID, ShowTime),
-> FOREIGN KEY (FilmID) REFERENCES Film(FilmID)
-> );
Query OK, 0 rows affected (0.02 sec)
```

## DDL for film\_club and its explanation: (PART 1)

The database consists of 8 tables: Member, Film, Performance, Performer, Reserves, Book, StarsIn, and Employee.

```
mysql> CREATE TABLE Member (  
->     MemberID INT PRIMARY KEY,  
->     Name VARCHAR(50) NOT NULL,  
->     Street VARCHAR(50) ,  
->     City VARCHAR(50) NOT NULL,  
->     State VARCHAR(50)  
-> );  
Query OK, 0 rows affected (0.02 sec)
```

The Member table contains information about the members who have registered for the service, including their unique MemberID, name, street address, city, and state. The data type for MemberID is INT and it is specified as the primary key. The data type for the other columns is VARCHAR(50) which means they are variable-length character strings with a maximum length of 50 characters. The columns Name, Street, City, and State are specified as NOT NULL, which means they must contain a value when a new row is inserted into the table.

```
mysql> CREATE TABLE Film (  
->     FilmID INT PRIMARY KEY,  
->     Title VARCHAR(100) NOT NULL,  
->     PubDate DATE NOT NULL,  
->     PurchasePrice DECIMAL(10,2) NOT NULL,  
->     Distributor VARCHAR(50) ,  
->     Kind VARCHAR(50) NOT NULL,  
->     RecommendedAge INT NOT NULL,  
->     SpokenLanguage VARCHAR(50) NOT NULL,  
->     SubtitleLanguage VARCHAR(50) NOT NULL  
-> );  
Query OK, 0 rows affected (0.01 sec)
```

For the Film table, the table has nine columns: FilmID, Title, PubDate, PurchasePrice, Distributor, Kind, RecommendedAge, SpokenLanguage, and SubtitleLanguage. The data type for FilmID is INT and it is specified as the primary key. The data type for PubDate is DATE, which means it stores date values in the format YYYY-MM-DD. The data type for PurchasePrice is DECIMAL(10,2), which means it stores decimal numbers with a maximum total of 10 digits and 2 digits after the decimal point. The columns Title, Distributor, Kind, RecommendedAge, SpokenLanguage, and SubtitleLanguage are specified as VARCHAR(50) and NOT NULL, which means they allow

characters with a maximum length of 50 characters.

```
mysql> CREATE TABLE Performance (  
->     FilmID INT NOT NULL,  
->     ShowTime DATETIME NOT NULL,  
->     Status VARCHAR(50) NOT NULL,  
->     PRIMARY KEY (FilmID, ShowTime),  
->     FOREIGN KEY (FilmID) REFERENCES Film(FilmID)  
-> );  
Query OK, 0 rows affected (0.02 sec)
```

In the Performance table, FilmID is an integer type that represents the ID of the film being performed. ShowTime is a datetime type that represents the start time of the film performance. Status is a string type that represents the current status of the film performance. PRIMARY KEY (FilmID, ShowTime) indicates that the combination of FilmID and ShowTime should be unique and together they form the primary key of this table. FOREIGN KEY (FilmID) REFERENCES Film(FilmID) indicates that the FilmID in the Performance table is a foreign key referencing the FilmID column in the Film table.

```
mysql> CREATE TABLE Performer (  
->     PerformerID INT PRIMARY KEY,  
->     Name VARCHAR(50) NOT NULL  
-> );  
Query OK, 0 rows affected (0.01 sec)
```

In the Performer table, PerformerID is an integer type that represents the ID of the performer. Name is a string type that represents the name of the performer. PRIMARY KEY (PerformerID) indicates that the PerformerID should be unique and serves as the primary key of this table.

```
mysql> CREATE TABLE Reserves (  
->     MemberID INT NOT NULL,  
->     FilmID INT NOT NULL,  
->     Location VARCHAR(50) NOT NULL,  
->     PRIMARY KEY (MemberID, FilmID),  
->     FOREIGN KEY (MemberID) REFERENCES Member(MemberID),  
->     FOREIGN KEY (FilmID) REFERENCES Film(FilmID)  
-> );  
Query OK, 0 rows affected (0.02 sec)
```

In the Reserves table, MemberID is an integer type that represents the ID of the member who made the reservation. FilmID is an integer type that represents the ID of the film being reserved. Location is a string type that represents the location where the film is reserved. PRIMARY KEY (MemberID, FilmID) indicates that the combination of MemberID and FilmID should be unique

and together they form the primary key of this table. FOREIGN KEY (MemberID) REFERENCES Member(MemberID) indicates that the MemberID in the Reserves table is a foreign key referencing the MemberID column in the Member table. FOREIGN KEY (FilmID) REFERENCES Film(FilmID) indicates that the FilmID in the Reserves table is a foreign key referencing the FilmID column in the Film table.

```
mysql> CREATE TABLE Book (  
->     MemberID INT NOT NULL,  
->     FilmID INT NOT NULL,  
->     ShowTime DATETIME NOT NULL,  
->     Location VARCHAR(50) NOT NULL,  
->     ExtraCharge DECIMAL(10,2) ,  
->     PRIMARY KEY (MemberID, FilmID, ShowTime),  
->     FOREIGN KEY (MemberID) REFERENCES Member(MemberID),  
->     FOREIGN KEY (FilmID) REFERENCES Film(FilmID),  
->     FOREIGN KEY (ShowTime) REFERENCES Performance(ShowTime)
```

In the Book table:

MemberID is an integer type that cannot be null and references the Member table's MemberID column.

FilmID is an integer type that cannot be null and references the Film table's FilmID column.

ShowTime is a datetime type that cannot be null and references the Performance table's ShowTime column.

Location is a varchar type that cannot be null and represents the location of the book reservation.

ExtraCharge is a decimal type with a precision of 10 and a scale of 2 that cannot be null and represents any additional charges associated with the book reservation.

```
mysql> CREATE TABLE StarsIn (  
->     PerformerID INT NOT NULL,  
->     FilmID INT NOT NULL,  
->     Role VARCHAR(50) NOT NULL,  
->     PRIMARY KEY (PerformerID, FilmID),  
->     FOREIGN KEY (PerformerID) REFERENCES Performer(PerformerID),  
->     FOREIGN KEY (FilmID) REFERENCES Film(FilmID)  
-> );  
Query OK, 0 rows affected (0.02 sec)
```

In the StarsIn table:

PerformerID is an integer type that cannot be null and references the Performer table's

PerformerID column.

FilmID is an integer type that cannot be null and references the Film table's FilmID column.

Role is a varchar type that cannot be null and represents the role played by the performer in the film.

```
mysql>
mysql> CREATE TABLE Employee (
->     EmployeeID INT PRIMARY KEY,
->     MemberID INT NOT NULL,
->     Manager INT NOT NULL,
->     FOREIGN KEY (MemberID) REFERENCES Member(MemberID),
->     FOREIGN KEY (Manager) REFERENCES Employee(EmployeeID)
-> );
Query OK, 0 rows affected (0.02 sec)
```

The Employee table is created with the following columns:

EmployeeID: an integer primary key.

MemberID: an integer not null, which references the MemberID column in the Member table.

Manager: a varchar(50) not null.

```
mysql>
mysql> CREATE TABLE PreferredMember (
->     MemberID INT PRIMARY KEY,
->     DiscountLevel INT NOT NULL
-> );
Query OK, 0 rows affected (0.01 sec)
```

The PreferredMember table is created with the following columns:

MemberID: an integer not null, which references the MemberID column in the Member table.

DiscountLevel: an integer not null.

The primary key for this table is set to the MemberID column.

After the table is created, we use *show tables* to view the table that has been created:

```
mysql> show tables;
+-----+
| Tables_in_film_club |
+-----+
| book                 |
| employee             |
| film                 |
| member               |
| performance          |
| performer            |
| preferredmember      |
| reserves             |
| starsin              |
+-----+
9 rows in set (0.00 sec)
```

Explain how the data should be loaded into the database by the system.


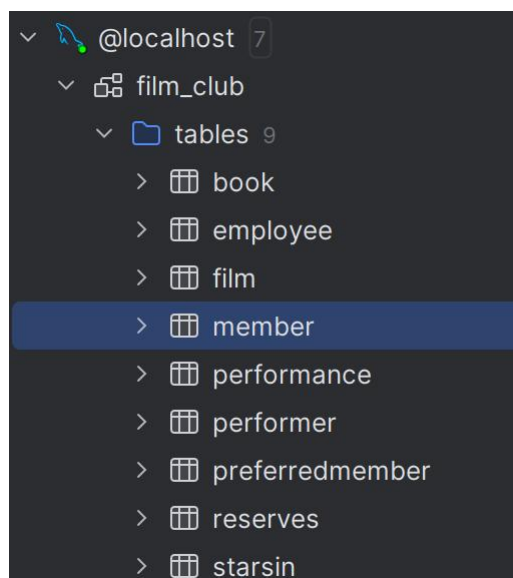
After creating the tables, we shall load the data to database. Here I will give two methods, the first is to input data by entering commands on the mysql terminal:

```
mysql> INSERT INTO Member (MemberID, Name, Street, City, State)
-> VALUES (1, 'John Smith', '123 Main St', 'New York', 'NY'),
->          (2, 'Jane Doe', '456 Broadway', 'Los Angeles', 'CA'),
->          (3, 'Bob Johnson', '789 1st Ave', 'Chicago', 'IL');
Query OK, 3 rows affected (0.08 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

However, this method is time-consuming and labor-intensive, and is prone to errors. Next, we will use the graphical display to operate the database:

The database operate tool is **DATAGRIP**

After entering the MySQL root password, download the relevant components and try to connect to the local MySQL localhost, then we will see the created table on DataGrip:

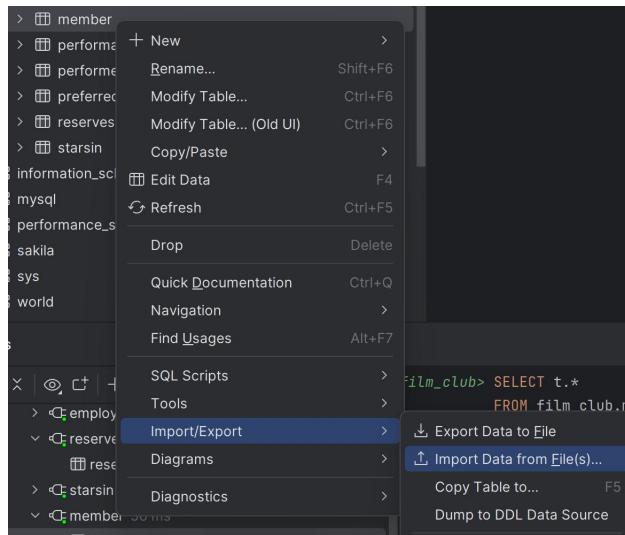


	MemberID	Name	Street	City	State
1	1	John Smith	123 Main St	New York	NY
2	2	Jane Doe	456 Broadway	Los Angeles	CA
3	3	Bob Johnson	789 1st Ave	Chicago	IL

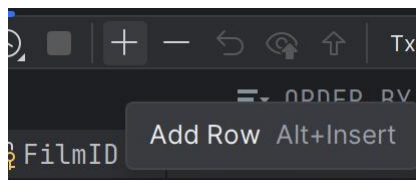
We can see that the data entered from the command line is already in the table

We can do this by right-clicking on the table via import, in the "Import Data" dialog box, select the file that contains the data you want to load into the table.

And then choose the appropriate file format and settings for the import. We can use CSV files to import. Because CSV is a text file format that uses commas to separate values in each row. It is widely used and supported by most database systems. Other format may use for instance Excel spreadsheets, JSON, XML, SQL dump files.



Or enter the sql statement in the console to enter the data. Of course, we can also add rows to the table to enter data.



Since there is no example data, we will add our own data to test the code of part 2, the data is shown below:

Member:

MemberID	Name	Street	City	State
1	Daniel	123 Main St	Dublin	Ireland
2	Sarah	456 Elm St	Galway	Ireland
3	John	789 Maple St	Cork	Ireland
4	John Smith	123 Main St	New York	NY
5	Jane Doe	456 Broadway	Los Angeles	CA
6	Bob Johnson	789 1st Ave	Chicago	IL

Performance:

FilmID	ShowTime	Status
1	2019-05-01 19:00:00	Playing
2	2019-05-01 22:00:00	Playing
3	2019-06-22 13:00:00	Playing
4	2019-06-01 17:00:00	Playing
5	2019-06-01 20:00:00	Playing
6	2019-07-20 15:00:00	Playing
7	2019-11-28 18:00:00	Playing
8	2019-11-28 21:00:00	Playing
9	2017-12-23 16:00:00	Playing
10	2016-12-17 20:00:00	Playing

Performer:

PerformerID	Name
8	Enna Stone
7	Ryan Gosling
6	Sally Hawkins
5	Al Pacino
4	Bong Joon-ho
3	Scarlett Johansson
2	Tom Hanks
1	Robert Downey Jr.

Reserves:

MemberID	FilmID	Location
1	1	1 Center
2	1	2 Left
3	1	4 Right
4	2	3 Center
5	2	5 Left
6	2	6 Right
7	3	2 Center
8	3	3 Left
9	3	7 Right

Book:

MemberID	FilmID	ShowTime	Location	ExtraCharge
1	1	2019-05-01 19:00:00	Center	0.00
2	1	2019-06-22 13:00:00	Left	0.00
3	2	2019-06-01 20:00:00	Right	0.00

Films:

F..	Title	PubDate	PurchasePrice	Distributor	Kind	..	...	Subtit
1	1 Avengers: E...	2019-04-26	20.00	Marvel Studios	Action	12	English	English
2	2 Toy Story 4	2019-06-21	15.00	Pixar Animation...	Animation	0	English	English
3	3 Parasite	2019-05-30	18.00	CJ Entertainment	Foreign	16	Korean	English
4	4 The Lion Ki...	2019-07-19	17.00	Walt Disney Pic...	Animation	0	English	English
5	5 The Irishman	2019-11-27	25.00	NetFlix	Drama	18	English	English
6	6 The Shape o...	2017-12-22	15.00	Fox SearchLight...	Drama	16	English	English
7	7 La La Land	2016-12-16	10.00	Summit Entertai...	Musical	12	English	English



## PART 2

### 3. SQL Statements

Provide a list of the SQL statements that need to be written to manipulate and retrieve data in the database, including the following:

**Let Dara reserve all the films that Daniel has reserved.**

```
USE film_club; ## first to use film_club database
```

```
UPDATE reserves
```

```
SET MemberID = (SELECT MemberID FROM member WHERE Name = 'Dara')
```

```
WHERE MemberID = (SELECT MemberID FROM member WHERE Name = 'Daniel');
```

```
film_club> UPDATE Reserves
      SET MemberID = (SELECT MemberID FROM Member WHERE Name = 'Dara')
      WHERE MemberID = (SELECT MemberID FROM Member WHERE Name = 'Daniel')
[2023-04-18 12:32:04] 3 rows affected in 6 ms
```

After test, the member ID of Daniel's booking record has been successfully changed to Dara's member ID in the reserves table:

MemberID	FilmID	Location
1	1	Center
1	2	Left
1	4	Right
2	3	Center
2	5	Left
2	6	Right
3	2	Center
3	3	Left
3	7	Right

**Delete the films with a purchase price above the average purchase price.**

```
SET @avg_purchase_price = (SELECT AVG(PurchasePrice) FROM film);
```

```
DELETE FROM film
```

```
WHERE PurchasePrice > @avg_purchase_price;
```

Some problems encountered, by modifying the foreign key constraints connected to the film database, change the delete action configuration from RESTRICT to cascade

```
SHOW CREATE TABLE performance;

ALTER TABLE performance
DROP FOREIGN KEY performance_ibfk_1;

ALTER TABLE performance
ADD CONSTRAINT performance_ibfk_1
FOREIGN KEY (FilmID)
REFERENCES Film(FilmID)
ON DELETE CASCADE;

SHOW CREATE TABLE reserves;

ALTER TABLE reserves
DROP FOREIGN KEY reserves_ibfk_2;

ALTER TABLE reserves
ADD CONSTRAINT reserves_ibfk_2
```

```
film_club> DELETE FROM Film
      WHERE PurchasePrice > @avg_purchase_price
[2023-04-18 14:35:54] 3 rows affected in 55 ms
```

FilmID	Title	PubDate	PurchasePrice	Distributor	Kind	Language	Subtitles
1	Avengers: E...	2019-04-26	20.00	Marvel Studios	Action	English	English
2	Toy Story 4	2019-06-21	15.00	Pixar Animation...	Animation	English	English
3	Parasite	2019-05-30	18.00	CJ Entertainment	Foreign	Korean	English
4	The Lion Ki...	2019-07-19	17.00	Walt Disney Pic...	Animation	English	English
5	The Irishman	2019-11-27	25.00	Netflix	Drama	English	English
6	The Shape o...	2017-12-22	15.00	Fox Searchlight...	Drama	English	English
7	La La Land	2016-12-16	10.00	Summit Entertai...	Musical	English	English

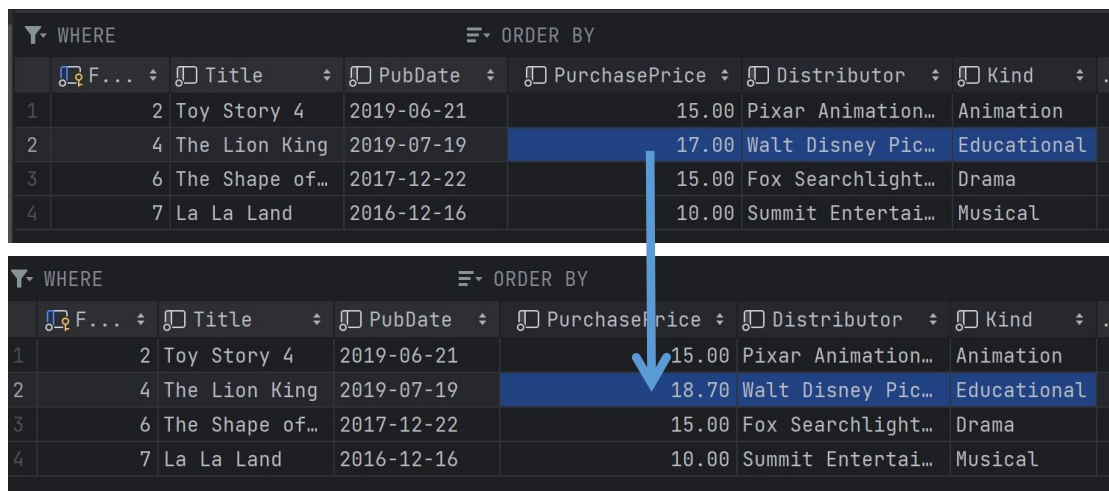
MemberID	FilmID	Location
2	2	Left
2	4	Right
2	6	Right
3	7	Right

**Increase the purchase price of educational films by 10%.**

```
UPDATE film
SET PurchasePrice = PurchasePrice * 1.1
WHERE kind = 'Educational';
```

```
film_club> UPDATE film
          SET PurchasePrice = PurchasePrice * 1.1
          WHERE kind = 'Educational'
[2023-04-18 15:03:00] 1 row affected in 7 ms
```

After test, the purchase price successfully increased 1.1time.



	F...	Title	PubDate	PurchasePrice	Distributor	Kind
1	2	Toy Story 4	2019-06-21	15.00	Pixar Animation...	Animation
2	4	The Lion King	2019-07-19	17.00	Walt Disney Pic...	Educational
3	6	The Shape of...	2017-12-22	15.00	Fox Searchlight...	Drama
4	7	La La Land	2016-12-16	10.00	Summit Entertai...	Musical

	F...	Title	PubDate	PurchasePrice	Distributor	Kind
1	2	Toy Story 4	2019-06-21	15.00	Pixar Animation...	Animation
2	4	The Lion King	2019-07-19	18.70	Walt Disney Pic...	Educational
3	6	The Shape of...	2017-12-22	15.00	Fox Searchlight...	Drama
4	7	La La Land	2016-12-16	10.00	Summit Entertai...	Musical

**List the streets of Members who have reserved musical films.**

```
SELECT DISTINCT m.street
FROM member m
JOIN reserves r ON m.MemberID = r.MemberID
JOIN film f ON r.FilmID = f.FilmID
WHERE f.kind = 'Musical';
```

The purpose of this query is to filter the database for the addresses of members who have pre-ordered music and movies, and then find the streets where those members live. That is, the query is to find out what the specific street names are for members who have pre-ordered music and movies.

```
film_club> SELECT DISTINCT m.street
          FROM member m
          JOIN reserves r ON m.MemberID = r.MemberID
          JOIN film f ON r.FilmID = f.FilmID
          WHERE f.Kind = 'Musical'
[2023-04-18 15:15:31] 1 row retrieved starting from 1 in 27 ms (execution: 3 ms, fetching: 24 ms)
```

After testing, the address of the scheduled member who chose the musical type was successfully found

The first screenshot shows a table with columns: F..., Title, PubDate, PurchasePrice, Distributor, and Kind. The row for 'La La Land' is highlighted.

The second screenshot shows a table with columns: MemberID, FilmID, and Location. The row for MemberID 3 is highlighted.

The third screenshot shows a table with columns: MemberID, Name, Street, City, and State. The row for MemberID 3 is highlighted, showing the address '789 Maple St'.

The fourth screenshot shows the 'Output' window for the query 'film\_club.member'. The 'street' column is selected, and the value '789 Maple St' is displayed.

Which foreign films name ends in “war” ?

```
SELECT title
FROM film
WHERE kind = 'Foreign' AND title LIKE '%war';
```

```
film_club> SELECT title
FROM film
WHERE kind = 'Foreign' AND title LIKE '%war'
[2023-04-18 15:26:09] 1 row retrieved starting from 1 in 23 ms (execution: 3 ms, fetching: 20 ms)
```

After testing, successfully found foreign movie names ending in war

The first screenshot shows a table with columns: F..., Title, PubDate, PurchasePrice, Distributor, Kind, and Spoke... The row for 'Parasite war' is highlighted.

The second screenshot shows the 'Output' window for the query '# Which foreign film...s name ends in “war”?'. The 'title' column is selected, and the value 'Parasite war' is displayed.

### What is the highest purchase price of reserved films by kind?

```
SELECT Kind, MAX(PurchasePrice)
FROM Film f JOIN Reserves r ON f.FilmID = r.FilmID
GROUP BY Kind;
```

```
film_club> SELECT Kind, MAX(PurchasePrice)
            FROM Film f JOIN Reserves r ON f.FilmID = r.FilmID
            GROUP BY Kind
[2023-04-18 15:38:56] 4 rows retrieved starting from 1 in 21 ms (execution: 2 ms, fetching: 19 ms)
```

	Kind	MAX(PurchasePrice)
1	Animation	15.00
2	Educational	18.70
3	Drama	15.00
4	Musical	10.00

### List the Members' name whose average purchase price for reserved films is greater than \$400.

```
SELECT m.Name
FROM Member m
JOIN Reserves r ON m.MemberID = r.MemberID
JOIN Film f ON r.FilmID = f.FilmID
GROUP BY m.MemberID
HAVING AVG(f.PurchasePrice) > 400;
```

```
film_club> SELECT m.Name
            FROM Member m
            JOIN Reserves r ON m.MemberID = r.MemberID
            JOIN Film f ON r.FilmID = f.FilmID
            GROUP BY m.MemberID
            HAVING AVG(f.PurchasePrice) > 400
[2023-04-18 16:11:01] 0 rows retrieved in 28 ms (execution: 7 ms, fetching: 21 ms)
```

Since there is not enough data to support it, there are no members who spend more than \$400, but the code works fine.

	Name
0 rows	

## How many members live in Dublin?

```
SELECT COUNT(*) FROM member
```

```
WHERE city = 'Dublin';
```

```
film_club> SELECT COUNT(*) FROM member
        WHERE city = 'Dublin'
[2023-04-18 16:22:24] 1 row retrieved starting from 1 in 23 ms (execution: 2 ms, fetching: 21 ms)
```

	MemberID	Name	Street	City	State
1	1	Daniel	123 Main St	Dublin	Ireland
2	2	Dana	456 Elm St	Galway	Ireland
3	3	John	789 Maple St	Cork	Ireland
4	4	John Smith	123 Main St	New York	NY
5	5	Jane Doe	456 Broadway	Los Angeles	CA
6	6	Bob Johnson	789 1st Ave	Chicago	IL

Output		# How many members live in Dublin?
1	1	

## Create a view to gather all members who live in Galway and reserved foreign film.

```
CREATE VIEW galway_foreign AS
```

```
SELECT m.Name, f.Title
```

```
FROM member m JOIN reserves r ON m.MemberID = r.MemberID
```

```
JOIN film f ON r.FilmID = f.FilmID
```

```
WHERE m.city = 'Galway' AND f.kind = 'Foreign';
```

```
film_club> # Create a view to gather all members who live in Galway and reserved foreign film.
CREATE VIEW galway_foreign AS
SELECT m.Name, f.Title
FROM member m JOIN reserves r ON m.MemberID = r.MemberID
JOIN film f ON r.FilmID = f.FilmID
WHERE m.city = 'Galway' AND f.kind = 'Foreign'
[2023-04-18 16:29:29] completed in 8 ms
```

	MemberID	Name	Street	City	State
1	1	Daniel	123 Main St	Dublin	Ireland
2	2	Dana	456 Elm St	Galway	Ireland
3	3	John	789 Maple St	Cork	Ireland
4	4	John Smith	123 Main St	New York	NY
5	5	Jane Doe	456 Broadway	Los Angeles	CA
6	6	Bob Johnson	789 1st Ave	Chicago	IL

	MemberID	FilmID	Location
1	1	2	Left
2	1	4	Right
3	2	3	Center
4	2	6	Right
5	3	Center	
6	3	7	Right

	F...	Title	PubDate	PurchasePrice	Distributor	Kind
1	2	Toy Story 4	2019-06-21	15.00	Pixar Animation	Animation
2	3	Parasite war	2019-05-30	18.00	CJ Entertainment	Foreign
3	4	The Lion King	2019-07-19	18.70	Walt Disney Pic...	Educational

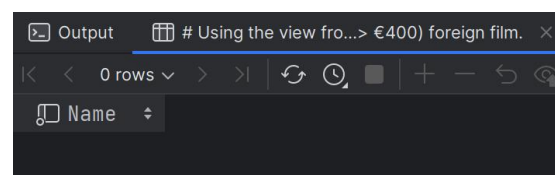
	Name	Title
1	Dana	Parasite war

After testing, we can successfully find out the foreign movies booked by members in Galway from the table

Using the view from the previous question to find out the member who lives in Galway and reserved expensive (purchase price > €400) foreign film.

```
SELECT Name
FROM galway_foreign
WHERE title IN (SELECT title FROM film WHERE kind='Foreign' AND PurchasePrice > 400);
```

```
film_club> SELECT Name
            FROM galway_foreign
            WHERE title IN (SELECT title FROM film WHERE kind='Foreign' AND PurchasePrice > 400)
[2023-04-18 16:39:28] 0 rows retrieved in 28 ms (execution: 8 ms, fetching: 20 ms)
```



After testing, there are no members who spend more than \$400 on foreign movies in Galway, but the code runs successfully and outputs results.

#### 4. Conclusion

In conclusion, SQL statements are an essential tool for working with relational databases, enabling users to manipulate and retrieve data in a flexible and powerful way. Completing the database development for the film club's information system is crucial in providing valuable insights into the business operations, ensuring data accuracy, integrity, and security, and improving overall efficiency and productivity. With a well-designed database, the film club can make informed decisions about inventory management, marketing strategies, and data analysis, leading to improved performance and profitability.