Database Group Project

CSCI 2141

Clinton Morrison

Stirling Brown-Sweeting

Group 12

Contents

Problem Statement

The goal of this project was to design a database for an online video game website. Users log in to this type of website to play various games. The database must store information about the users and games on this website. The exact requirements of this problem are outlined below.

The database system must store several pieces of information about each of the online games. Firstly, the name of each game must be stored. A description of the game and the number of times it was played must also be stored. Two different games cannot have the same name.

Each game on the website was created by some author. The system must keep a record of the authors of each game. The name of the author as well as a short bio about the author should be stored. Each author must have a unique name. This is reasonable because game authors are usually companies, and thus almost never have identical names. The games created by each author must also be stored. A game can only have one author.

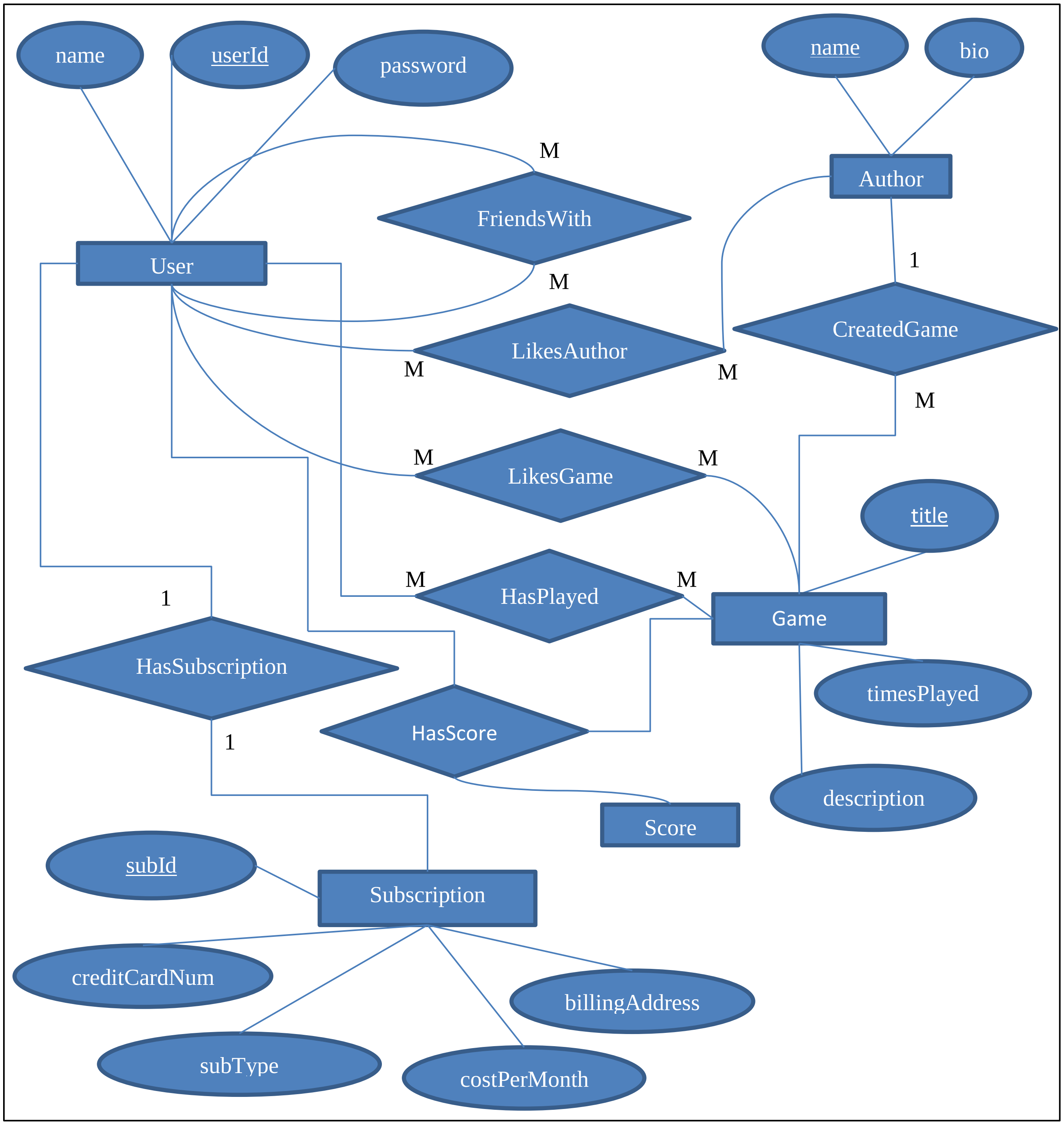
There is also information about each user that must be stored. All users have a user id, password, and a name. The user id uniquely identifies users. Users can have multiple favorite games and favorite authors. Users can buy subscriptions to play additional games on the website. For these users billing information must be stored. This includes the billing address, monthly cost, credit card number, and the subscription type. Not all users have subscriptions.

The system must keep track of which games a given user has played, their favorite games, and all the scores they have obtained in every game that they have played. An important query would be to fetch the highest scores for a given game.

Finally, users can also be friends with other users. The database must be able to keep track of who a given user is friends with.

Database Modeling and Design

The ER-diagram below was designed to meet all of the requirements outlined in the first section. The application has been modeled as a series of entity sets and relations between them. The design is explained on the following pages.



The entity set “Game” stores information about games on the website. The name of the game is stored in the “title” attribute. This is used as the key for this entity set because it is unique to each game. A description of the game is stored in the “description” attribute. Finally, the number of times the game has been played is stored in the “timesPlayed” attribute.

The entity set “User” stores information about the users of the website. The name of user is stored in the “name” attribute. The username used for a user to login to the website is stored in the “userId” attribute. This is used as the key because all users must have different user ids. The user’s password is stored in the “password” attribute.

Subscription information is stored separately from users because not all users have purchased subscriptions. For this purpose there is an entity set “Subscription”. The id which uniquely identifies each subscription is stored in the “subId” attribute. This is used as a key since it is unique to each subscription. The type of the subscription is stored in the “subType” attribute. The monthly charge to the user is stored in the “costPerMonth” attribute. The credit card number to charge is represented by the “creditCardNum” attribute. Lastly, the billing address is stored in the “BillingAddress” attribute.

Users who have purchased a subscription are related to their subscription information via the “HasSubscription” relationship. This is a relationship between the entity set “User” and the entity set “Subscription”. Since it is relationship between two entities it is a binary relation. It is 1-1 since one user may only have one subscription and two users cannot have the same subscription. All subscription ids are unique.

The entity set “Score” stores scores that a user has obtained in a given game. Score only has the attribute “Score” which is also the key. The relationship “HasScore” is a third degree relationship between “User”, “Score”, and “Game”. It records the scores that the user has achieved in the given game. This can be used to determine who has the highest score in a game. Note that a single user may have many different scores in a given game.

The relationship “HasPlayed” keeps track of what games have been played by what users. It is a binary relationship between “User” and “Game”. It is a M-M relationship because a user may play many different games and many users may play the same games. If this relationship is defined for a given user and game pair, then that user has played that game. If it is not, that user has not played that game.

The entity set “Author” stores information about the companies and people who have created games that are hosted on the website. The name of the author is stored in the “name” attribute. This is also used as a key since each author must have a unique name. A short biography of the author is also stored in the “bio” attribute.

Authors are related to the games they have created by the “CreatedGame” relation. This is a binary relation that relates “Author” to “Game”. It is 1-M because one author may produce many games, but each game may only have one author. This relation can be used to look up the games by a given author, or the author of a specific game.

Users can be friends with other users. This is represented by the relationship “FriendsWith”. This is a recursive binary relationship between “User” and “User”. It is M-M because a user can have many friends. The user can also be the friend of many other users.

Users can like their favorite games. The relationship “LikesGame” is a binary relationship between “User” and “Game”. It is a M-M relationship because a single user may like many games and many users may like the same games.

Users can also like their favorite authors. The relationship “LikesAuthor” relates “User” and “Author”. It is a binary M-M because one user can like many authors and other users may like the same authors.

Relational Database Design

Below is the relational design for the previous ER-diagram.

User(userId, name, password)

Author(name, bio)

Game(title, timesPlayed, description, authorName)

Subscription(subId, creditCardNum, subType, costPerMonth, billingAddress)

HasSubscription(userId, subId)

HasPlayed(userId, title)

LikesAuthor(userId, authorName)

HasScore(userId, title, score)

FriendsWith(userId1, userId2)

LikesGame(userId, title)

There are a few points worth making about the design. Firstly, note that only “userId” is used as a key for the relation “HasSubscription” because it is a 1-1 relationship. It would be redundant to use both “userId” and “subId” as keys. “HasSubscription” was created as a separate scheme from “User” because not all users have a subscription. Also, score is not used as a relation scheme because it only occurs within the relationship “HasScore”. “score” is part of the key of “HasScore” because a single player may have many different scores in a single game. Instead of creating a scheme for the relationship “CreatedGame”, “authorName” was simply included in Game. This makes sense because games must have an author, and they may only have one author. If “CreatedGame” was a separate scheme than a new game could be could be added to the database without an author for that game being added to the system. In the relationship “FriendsWith”, “userId1” corresponds to the user that is friends with the user “userId2”. For example, if “clinton” is friends with “stirling” and vice versa, then (“clinton”, “stirling”) and (“stirling”, “clinton”) must be two instances of the “FriendsWith” scheme. “clinton” may be friends with “stirling” without “stirling” being friends with “clinton”.

To verify the quality of the design, each scheme must be checked for redundancies. The functional dependencies of each scheme must be considered to show every relation is in BCNF. Specifically, if the left hand side of each functional dependency is a super key for the scheme than that scheme is in BCNF. The set F of all functional dependencies for each relation is shown below.

Scheme: User. F = { userId → name, userId → password }. User is in BCNF.

Scheme: Author. F = { name → bio }. Author is in BCNF.

Scheme: Game. F = { title → timesPlayed, title → description, title → authorName }. Game is in BCNF.

Scheme: Subsctipion. F = { subId → creditCardNum, subId → subtype, subId → cosPerMonth, subId → billingAddress }. Subscription is in BCNF.

Scheme: HasSubscription. F = { userId → subscriptionId }. HasSubscription is in BCNF.

Scheme: HasPlayed. F = { }. The set is empty because all the attributes are keys. HasPlayed is in BCNF.

Scheme: HasScore. F = { }. The set is empty because all the attributes are keys. HasScore is in BCNF.

Scheme: FriendsWith. F = { }. Again, the attributes are both keys so do not imply each other. FriendsWith is in BCNF.

Scheme: LikesGame. F = { }. The attributes are both unique keys so F is empty. LikesGame is in BCNF.

It is apparent that all of the designed schemes are in BCNF. This means that there is a minimum amount of redundancy within the design.

Implementation

The database designed was implemented using SQL on Dalhousie University's SQL server. This section shows examples of some of the commands used to implement and test the database. Note that the tables created to test the database are shown in Appendix A. Output from the queries discussed is shown in Appendix B.

Creation of Tables

First, the necessary tables were created. The below SQL commands were used to create the tables described by the relational design.

```
Create table User (
     userId varchar(30),
     name varchar(30),
     password varchar(30),
     constraint pkey primary key(userId));

Create table Author (
     name varchar(30),
     bio varchar(200),
     constraint pkey primary key(name));

Create table Game (
     title varchar(30),
     timesPlayed int,
     description varchar(200),
     authorName varchar(30),
     constraint pkey primary key(title),
     foreign key (authorName) references Author(name));

Create table Subscription (
     subId varchar(30),
     creditCardNum int,
     subType varchar(30),
     costPerMonth float,
     billingAddress varchar(60),
     constraint pkey primary key(subId));

Create table HasSubscription (
     userId varchar(30),
     subId varchar(30),
     constraint pkey primary key(userId, subId),
     foreign key (userId) references User(userId),
     foreign key (subId) references Subscription(subId));

Create table HasPlayed (
     userId varchar(30),
     title varchar(30),
     constraint pkey primary key(userId, title),
     foreign key (userId) references User(userId),
     foreign key (title) references Game(title));

Create table LikesAuthor (
     userId varchar(30),
     authorName varchar(30),
     constraint pkey primary key(userId, authorName),
     foreign key (userId) references User(userId),
     foreign key (authorName) references Author(name));

Create table HasScore (
     userId varchar(30),
     title varchar(30),
     score int,
     constraint pkey primary key(userId, title, score),
     foreign key (userId) references User(userId),
     foreign key (title) references Game(title));

Create table FriendsWith (
     userId1 varchar(30),
     userId2 varchar(30),
     constraint pkey primary key(userId1, userId2),
     foreign key (userId1) references User(userId),
     foreign key (userId2) references User(userId));

Create table LikesGame (
     userId varchar(30),
     title varchar(30),
     constraint pkey primary key(userId, title),
     foreign key (userId) references User(userId),
     foreign key (title) references Game(title));
```

Insertion of Data

After the tables were created, data was inserted into all of the tables. Some examples of this are shown below. All of the rows entered into the tables are shown in Appendix A.

```
Insert INTO User values('clint1', 'Clinton Morrison', 'abcde');

Insert INTO Author values('Atari', 'A game company. ');

Insert INTO Game values('Pong', 1000, 'A fun ping pong game.',
'Atari');

Insert INTO Subscription values('sub1', 1038593750, 'Golden',
50.00, '180 University Avenue, Halifax');

Insert INTO HasSubscription values('clint1', 'sub1');

Insert INTO HasPlayed values('clint1', 'Pong');

Insert INTO LikesAuthor values('clint1', 'Atari’);

Insert INTO HasScore values('clint1', 'Pong', 100);

Insert INTO FriendsWith values('clint1', 'stirling');

Insert INTO LikesGame values('clint1', 'Pong');
```

Queries

The database system designed can answer many different queries. The section shows several samples of queries that were performed on the database. However, this only represents a small subset of all possible queries. Note that Appendix B shows the results of all of these queries on a sample set of data.

Query: List all the name of all players who have played the game “Pong”.

```
Select User.name from User, HasPlayed where User.userId =
HasPlayed.userId and HasPlayed.title='Pong';
```

Query: List the highest score achieved in the game “Mario” and the id of the user who got this score.

```
Select userId, Score from HasScore where HasScore.title =
'Mario' order by Score Desc limit 1;
```

Query: List the top 5 highest scores achieved in any game and the userId of the players who obtained each score.

```
Select userId, Score from HasScore order by Score Desc limit 5;
```

Query: List the all userIds and their scores for the game “Pong” in descending order of score.

```
Select User.userId, score from User, HasScore where User.userId
= HasScore.userId and HasScore.title='Pong' order by score desc;
```

Query: List the names of all users who are friends with the user whose user id is “clint1”.

```
Select distinct User.name from User, FriendsWith where
FriendsWith.userId1='clint1' and User.userId =
FriendsWith.userId2;
```

Query: List the name of the author who made the game titled “Pong”.

```
Select Author.name from Author, Game where Author.name =
Game.authorName and Game.title = 'Pong';
```

Query: List all favorite authors of the user with user id “clint1”.

```
Select Author.name from Author, LikesAuthor where
LikesAuthor.userId = 'clint1' and Author.name =
LikesAuthor.authorName;
```

Query: List all the names of all users who have not played the game “Snake”.

```
Select distinct name from User where name NOT IN (Select
User.name from User, HasPlayed where User.userId =
HasPlayed.userId and HasPlayed.title='Snake');
```

Query: List the names of all users who currently have a subscription with type “Silver”.

```
Select distinct name from User, HasSubscription, Subscription
where User.userId = HasSubscription.userId and
HasSubscription.subId = Subscription.subId and
Subscription.subType='Silver';
```

Query: Count how many users have played the game “Sonic The Hedgehog”.

```
Select Count(User.userId) from User, HasPlayed where User.userId
= HasPlayed.userId and HasPlayed.title='Sonic The Hedgehog';
```

Query: List the credit card numbers of all users with subscriptions.

```
Select Subscription.creditCardNum from Subscription where
Subscription.subId IN (Select subId from User, HasSubscription
where User.userId = HasSubscription.userId);
```

Query: List all the userIds of all users that have not liked the author “Nintendo”.

```
Select userId from User where userId NOT IN (Select userId from
LikesAuthor where authorName = 'Nintendo');
```

Query: List how many different games the user with userId ‘clint1’ has both liked and played.

```
Select count(HasPlayed.title) from HasPlayed, LikesGame where
HasPlayed.title = LikesGame.title and HasPlayed.userId =
'clint1' and LikesGame.userId = 'clint1';
```

Query: List the games that have been liked by users who have subscriptions and pay less than $40 per month.

```
Select distinct HasPlayed.title from HasPlayed, HasSubscription,
LikesGame, Subscription where HasPlayed.userId =
HasSubscription.userId and HasPlayed.userId = LikesGame.userId
```

```
and Subscription.subId = HasSubscription.subId and
Subscription.costPerMonth < 47;
```

References

Farrag, A. A. (2013). *CSCI 2141: Introduction to Database Systems.* Halifax: Dalhousie University.

Ullman, J. D., & Widom, J. (2008). *A First Course in Database Systems.* Toronto: Pearson Education.

Appendix A – Tables Used

The tables used to test the database queries are shown below.

```
mysql> select * from Game;
+--------------------+-------------+-----------------------------+------------+
| title              | timesPlayed | description                 | authorName |
+--------------------+-------------+-----------------------------+------------+
| Mario              |        3000 | A fun side scrolling game.  | Nintendo   |
| Pong               |        1000 | A fun ping pong game.       | Atari      |
| Snake              |         500 | A classic arcade game.      | Atari      |
| Sonic The Hedgehog |         785 | A fun game.                 | Sega       |
+--------------------+-------------+-----------------------------+------------+
4 rows in set (0.00 sec)


mysql> select * from User;
+----------+-------------------------+----------+
| userId   | name                    | password |
+----------+-------------------------+----------+
| clint1   | Clinton Morrison        | abcde    |
| ed       | Edward M                | 57322    |
| fredy    | Fred Cook               | asdfg    |
| george25 | George Brown            | 12341234 |
| joe21    | Joe Smith               | joerocks |
| stirling | Stirling Brown-Sweeting | passwd   |
+----------+-------------------------+----------+
6 rows in set (0.00 sec)


mysql> select * from HasSubscription;
+----------+-------+
| userId   | subId |
+----------+-------+
| clint1   | sub1  |
| ed       | sub2  |
| stirling | sub3  |
+----------+-------+
3 rows in set (0.00 sec)


mysql> select * from Subscription;
+-------+---------------+---------+--------------+--------------------------------+
| subId | creditCardNum | subType | costPerMonth | billingAddress                 |
+-------+---------------+---------+--------------+--------------------------------+
| sub1  |    1038593750 | Gold    |           50 | 180 University Avenue, Halifax |
| sub2  |      43560349 | Silver  |           25 | 453 University Avenue, Halifax |
| sub3  |       5039485 | Gold    |           45 | 394 Maple Street, Halifax      |
+-------+---------------+---------+--------------+--------------------------------+
3 rows in set (0.00 sec)
```

```
mysql> select * from Author;
+----------+-------------------------+
| name     | bio                     |
+----------+-------------------------+
| Atari    | A game company.         |
| Nintendo | The creators of Mario.  |
| Sega     | A game company!         |
+----------+-------------------------+
3 rows in set (0.00 sec)


mysql> select * from LikesAuthor;
+----------+------------+
| userId   | authorName |
+----------+------------+
| fredy    | Atari      |
| clint1   | Nintendo   |
| stirling | Nintendo   |
| ed       | Sega       |
| stirling | Sega       |
+----------+------------+
5 rows in set (0.00 sec)


mysql> select * from HasPlayed;
+----------+--------------------+
| userId   | title              |
+----------+--------------------+
| clint1   | Mario              |
| ed       | Mario              |
| stirling | Mario              |
| clint1   | Pong               |
| fredy    | Pong               |
| stirling | Pong               |
| clint1   | Snake              |
| fredy    | Snake              |
| george25 | Snake              |
| joe21    | Sonic The Hedgehog |
| stirling | Sonic The Hedgehog |
+----------+--------------------+
11 rows in set (0.00 sec)
```

```
mysql> select * from HasScore;
+----------+--------------------+-------+
| userId   | title              | score |
+----------+--------------------+-------+
| clint1   | Mario              |    58 |
| clint1   | Mario              |   133 |
| ed       | Mario              |   756 |
| stirling | Mario              |   234 |
| clint1   | Pong               |   988 |
| fredy    | Pong               |   650 |
| fredy    | Pong               |   789 |
| george25 | Snake              |   246 |
| joe21    | Sonic The Hedgehog |   306 |
| joe21    | Sonic The Hedgehog |   606 |
| joe21    | Sonic The Hedgehog |   668 |
+----------+--------------------+-------+
11 rows in set (0.00 sec)


mysql> select * from FriendsWith;
+----------+----------+
| userId1  | userId2  |
+----------+----------+
| stirling | clint1   |
| stirling | ed       |
| joe21    | george25 |
| clint1   | stirling |
| ed       | stirling |
+----------+----------+
5 rows in set (0.00 sec)


mysql> select * from LikesGame;
+----------+--------------------+
| userId   | title              |
+----------+--------------------+
| stirling | Mario              |
| clint1   | Pong               |
| fredy    | Pong               |
| joe21    | Sonic The Hedgehog |
| stirling | Sonic The Hedgehog |
+----------+--------------------+
5 rows in set (0.00 sec)
```

Appendix B – Query Output

The output of the queries which were performed on the database is shown below.

Query: List all the name of all players who have played the game "Pong".

```
mysql> Select User.name from User, HasPlayed where User.userId =
HasPlayed.userId and HasPlayed.title='Pong';
+--------------------------+
| name                     |
+--------------------------+
| Clinton Morrison         |
| Fred Cook                |
| Stirling Brown-Sweeting |
+--------------------------+
3 rows in set (0.00 sec)
```

Query: List the highest score achieved in the game "Mario" and the id of the user who got this score.

```
mysql> Select userId, Score from HasScore where HasScore.title =
'Mario' order by Score Desc limit 1;
+--------+-------+
| userId | Score |
+--------+-------+
| ed     |   756 |
+--------+-------+
1 row in set (0.00 sec)
```

Query: List the top 5 highest scores achieved in any game and the userId of the players who obtained each score.

```
mysql> Select userId, Score from HasScore order by Score desc limit 5;
+--------+-------+
| userId | Score |
+--------+-------+
| clint1 |   988 |
| fredy  |   789 |
| ed     |   756 |
| joe21  |   668 |
| fredy  |   650 |
+--------+-------+
5 rows in set (0.00 sec)
```

Query: List the all userIds and their scores for the game "Pong" in descending order of score.

```
mysql> Select User.userId, score from User, HasScore where User.userId
= HasScore.userId and HasScore.title='Pong' order by score desc;
+--------+-------+
| userId | score |
+--------+-------+
| clint1 |   988 |
| fredy  |   789 |
| fredy  |   650 |
+--------+-------+
3 rows in set (0.00 sec)
```

Query: List the names of all users who are friends with the user whose user id is "clint1".

```
mysql> Select distinct User.name from User, FriendsWith where
FriendsWith.userId1='clint1' and User.userId = FriendsWith.userId2;
+-------------------------+
| name                    |
+-------------------------+
| Stirling Brown-Sweeting |
+-------------------------+
1 row in set (0.00 sec)
```

Query: List the name of the author who made the game titled "Pong".

```
mysql> Select Author.name from Author, Game where Author.name =
Game.authorName and Game.title = 'Pong';
+-------+
| name  |
+-------+
| Atari |
+-------+
1 row in set (0.00 sec)
```

Query: List all favorite authors of the user with user id "clint1".

```
mysql> Select Author.name from Author, LikesAuthor where
LikesAuthor.userId = 'clint1' and Author.name =
LikesAuthor.authorName;
+----------+
| name     |
+----------+
| Nintendo |
+----------+
1 row in set (0.00 sec)
```

Query: List all the names of all users who have not played the game “Snake”.

```
mysql> Select distinct name from User where name NOT IN (Select
User.name from User, HasPlayed where User.userId = HasPlayed.userId
and HasPlayed.title='Snake');
+-------------------------+
| name                    |
+-------------------------+
| Edward M                |
| Joe Smith               |
| Stirling Brown-Sweeting |
+-------------------------+
3 rows in set (0.00 sec)
```

Query: List the names of all users who currently have a subscription with type “Silver”.

```
mysql> Select distinct name from User, HasSubscription, Subscription
where User.userId = HasSubscription.userId and HasSubscription.subId =
Subscription.subId and Subscription.subType='Silver';
+----------+
| name     |
+----------+
| Edward M |
+----------+
1 row in set (0.00 sec)
```

Query: Count how many users have played the game “Sonic The Hedgehog”.

```
mysql> Select Count(User.userId) from User, HasPlayed where
User.userId = HasPlayed.userId and HasPlayed.title='Sonic The
Hedgehog';
+--------------------+
| Count(User.userId) |
+--------------------+
|                  2 |
+--------------------+
1 row in set (0.00 sec)
```

Query: List the credit card numbers of all users with subscriptions.

```
mysql> Select Subscription.creditCardNum from Subscription where
Subscription.subId IN (Select subId from User, HasSubscription where
User.userId = HasSubscription.userId);
+---------------+
| creditCardNum |
+---------------+
|    1038593750 |
|      43560349 |
|       5039485 |
+---------------+
3 rows in set (0.00 sec)
```

Query: List all the userIds of all users that have not liked the author “Nintendo”.

```
mysql> Select userId from User where userId NOT IN (Select userId from
LikesAuthor where authorName = 'Nintendo');
+----------+
| userId   |
+----------+
| ed       |
| fredy    |
| george25 |
| joe21    |
+----------+
4 rows in set (0.02 sec)
```

Query: List how many different games the user with userId ‘clint1’ has both liked and played.

```
mysql> Select count(HasPlayed.title) from HasPlayed, LikesGame where
HasPlayed.title = LikesGame.title and HasPlayed.userId = 'clint1' and
LikesGame.userId = 'clint1';
+------------------------+
| count(HasPlayed.title) |
+------------------------+
|                      1 |
+------------------------+
1 row in set (0.02 sec)
```

Query: List the games that have been liked by users who have subscriptions and pay less than $40 per month.

```
mysql> Select distinct HasPlayed.title from HasPlayed,
HasSubscription, LikesGame, Subscription where HasPlayed.userId =
HasSubscription.userId and HasPlayed.userId = LikesGame.userId and
Subscription.subId = HasSubscription.subId and
Subscription.costPerMonth < 47;
+---------------------+
| title               |
+---------------------+
| Mario               |
| Pong                |
| Sonic The Hedgehog  |
+---------------------+
3 rows in set (0.00 sec)
```