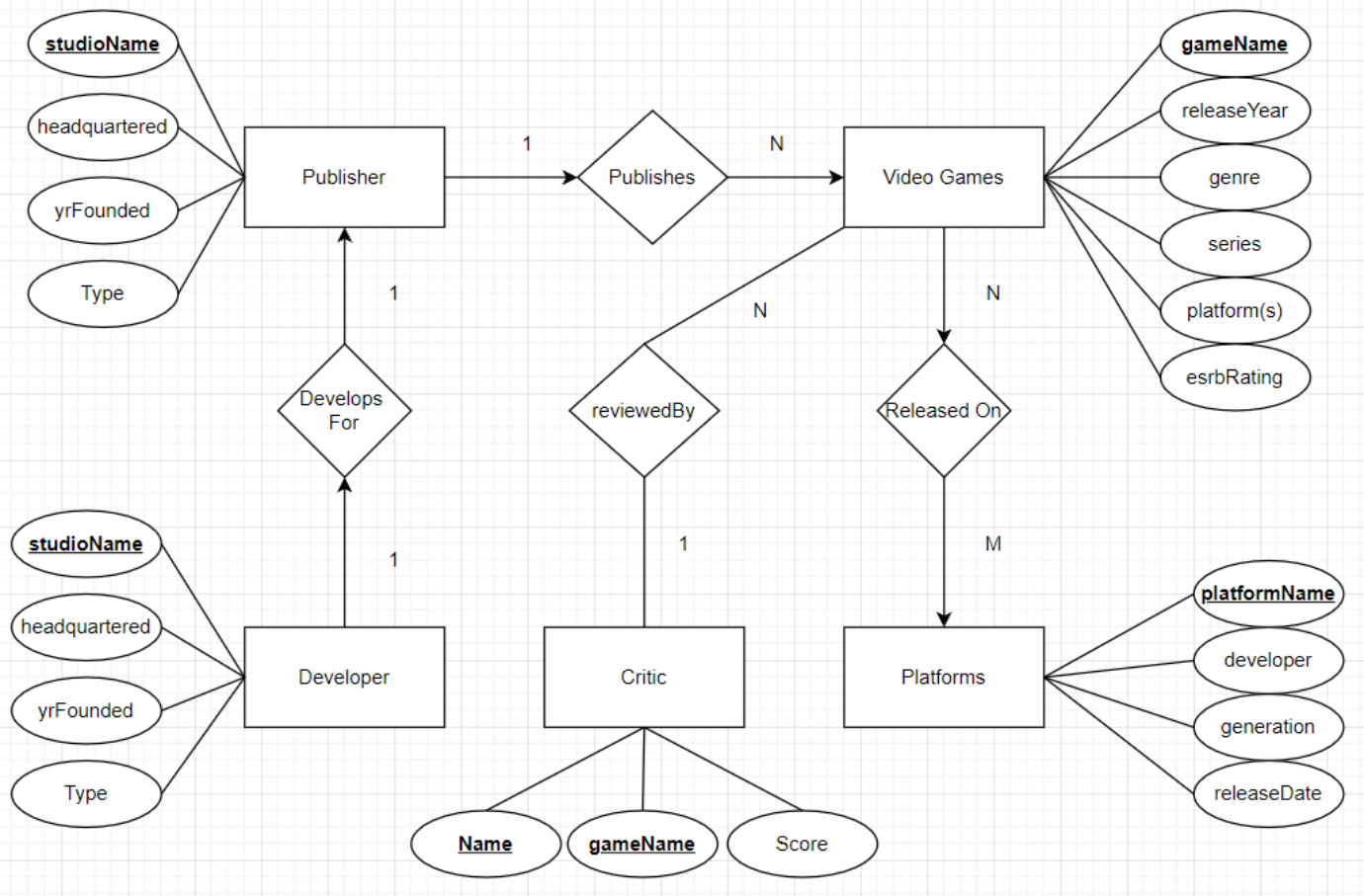
Megan Hoeksema

Comp 3421 Assignment 7

Diagram:



- Developer (**devStudioName**, headquartered, yrFounded, Type)
- Publisher (**pubStudioName**, headquartered, yrFounded, Type)
- VideoGames (**gameName**, releaseYear, genre, series, platform(s), esrbRating)
- Platforms (**platformName**, developer, generation, releaseDate)
- Critic (**Name, gameName**, score)
- DevelopsFor (**devStudioName**, **pubStudioName**)
- Publishes (**pubStudioName, gameName**)
- ReleasedOn (**gameName, platformName**)

1. Make sure you have foreign key constraints in your database. If you do not, add them. Show the create table statement that defines these foreign key constraints. Create an update that violates the foreign key constraint. List the update command, and then show the output of running this update.

Shows create table statement including a foreign key constraint

```
mysql> create table releasedOn(gameName varchar(50),
    ->     platformName varchar(50),
    ->     Primary key(gameName, platformName),
    ->     Foreign Key (gameName) references VideoGames(gameName));
```

Shows an insert statement which should violate the foreign key constraint, then shows the resulting error of running the insert command.

```
mysql> insert into releasedOn values ('game_26001', 'Playstation 5');
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails
 (`videogames`.`releasedon`, CONSTRAINT `releasedon_ibfk_1` FOREIGN KEY (`gameName`)
 REFERENCES `videogames` (`gameName`))
```

Shows an update statement which should violate the foreign key constraint, then shows the resulting error of running the insert command.

```
mysql> update releasedOn
    -> set gameName = 'game_001'
    -> where gameName = 'game_1';
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`videogames
`.`releasedon`, CONSTRAINT `releasedon_ibfk_1` FOREIGN KEY (`gameName`) REFERENCES `videogames` (
`gameName`))
```

2. Write a MySQL procedure or function (your choice) and script that calls this procedure/function. This procedure/function should be logical for your database and should include one or more inputs and an output. After calling the procedure, you should print the results of the output. If your procedure/function makes multiple database state changes, document the changes with before/after select commands showing a (small) set of data that is changed. Include prose (English) that describes what the procedure/function does.

Procedure:

```
mysql> delimiter //
mysql> create procedure proc_inOut(IN inScore INT, IN inYear INT, OUT outCount INT)
    -> begin
    -> select count(*) INTO outCount
    -> from videoGames V, critic C
    -> where V.gameName = C.gameName and v.releaseYear = inYear and C.score = inScore;
    -> end
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
```

Script to call the procedure:

```
mysql> source proc_inOut.txt
Query OK, 0 rows affected (0.01 sec)
```

The procedure takes in a game release year and a critic score and returns the count of games with the input score released in the input year.

```
mysql> set @theNum = 0 ; # variable for "output"
Query OK, 0 rows affected (0.00 sec)

mysql> set @varYear = 1999 ; #var for input
Query OK, 0 rows affected (0.00 sec)

mysql> set @varScore = 97;
Query OK, 0 rows affected (0.00 sec)

mysql> call proc_inOut( @varScore, @varYear, @theNum);
Query OK, 1 row affected (0.02 sec)

mysql> select @theNum as 'Number of Games';
+-----------------+
| Number of Games |
+-----------------+
|               2 |
+-----------------+
1 row in set (0.00 sec)
```

For the above query, we are inputting the Score as 97 and the Year as 1999 which results in 2 games being released in the year 1999 that obtained a score of 97. We can confirm the results using the following query:

```
mysql> Select v.gameName, C.score, V.releaseYear
    -> From videoGames V, critic C
    -> where V.releaseYear = 1999 and C.score = 97 and V.gameName = C.gameName;
+------------+-------+-------------+
| gameName   | score | releaseYear |
+------------+-------+-------------+
| game_12044 |    97 |        1999 |
| game_15723 |    97 |        1999 |
+------------+-------+-------------+
2 rows in set (0.01 sec)
```

3. Create an index and show how it speeds up two types of queries: 1) a selection on a single relation, and 2) a selection that involves a join. Note: To get times that will be measurably different, you will likely need to populate your databases with a larger dataset.

First populated videoGames table with 5 million different records.

```
mysql> select count(*) from videoGames;
+----------+
| count(*) |
+----------+
|  5000000 |
+----------+
1 row in set (2.74 sec)
```

1. Ran a query on releaseYear (with no Index)

```
mysql> select Count(*)
    -> from videoGames V
    -> where V.releaseYear = 1999;
+----------+
| Count(*) |
+----------+
|    69279 |
+----------+
1 row in set (4.14 sec)
```

2. Ran a query containing a Join (with No Index)

```
mysql> Select v.genre, AVG(c.score)
    -> From videoGames V, critic C
    -> where V.releaseYear = 1999 and V.gameName = C.gameName
    -> group by v.genre;
+-----------------------+--------------+
| genre                 | AVG(c.score) |
+-----------------------+--------------+
| Puzzlers and Party    |      55.1026 |
| RPG                   |      44.6061 |
| Survival and Horror   |      50.4194 |
| MOBA                  |      56.9737 |
| Strategy              |      55.7429 |
| Sandbox               |      50.5357 |
| Platformer            |      54.6061 |
| Shooter               |      47.4872 |
| Simulation and Sports |      58.4848 |
| Action-Adventure      |      51.2174 |
+-----------------------+--------------+
10 rows in set (6.34 sec)
```

Created an index on releaseYear

```
mysql> create table VideoGames(
    ->     gameName varchar(50) primary key,
    ->     releaseYear int not null,
    ->     genre varchar(50),
    ->     series varchar(20),
    ->     platform varchar(50) not null,
    ->     esrbRating varchar(20),
    ->     index (releaseYear));
Query OK, 0 rows affected (0.03 sec)
```

1. ***Ran a query on releaseYear (with Index)***

```
mysql> select Count(*)
    -> from videoGames V
    -> where V.releaseYear = 1999;
+----------+
| Count(*) |
+----------+
|    69279 |
+----------+
1 row in set (0.05 sec)
```

2. ***Ran a query containing a Join (with Index)***

```
mysql> Select v.genre, AVG(c.score)
    -> From videoGames V, critic C
    -> where V.releaseYear = 1999 and V.gameName = C.gameName
    -> group by v.genre;
+----------------------+--------------+
| genre                | AVG(c.score) |
+----------------------+--------------+
|  Puzzlers and Party  |      55.1026 |
|  RPG                 |      44.6061 |
|  Survival and Horror |      50.4194 |
|  MOBA                |      56.9737 |
|  Strategy            |      55.7429 |
|  Sandbox             |      50.5357 |
|  Platformer          |      54.6061 |
|  Shooter             |      47.4872 |
|  Simulation and Sports |      58.4848 |
|  Action-Adventure    |      51.2174 |
+----------------------+--------------+
10 rows in set (3.05 sec)
```

Conclusions:

1. ***A selection on a single relation yielded a result in 4.14s without an index but lowered to 0.03s with an index.***
2. ***A selection containing a join yielded a result in 6.34s without an index but lowered to 3.05s with an index.***