CPSC 304 Project Cover Page

Milestone #: 4

Date: 29 November, 2024

Group Number: 44

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Kshitij Gomber	18521526	n4i6m	kshitijgomber@gmail.com
Luke Nathan	38946877	a2x9g	lnathan80@gmail.com
Apoorva Devarakonda	66647223	z1o3g	devarka403@gmail.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

Project Summary

Our final project is a database representation of a website that allows users to log and review games, track their development, follow other users, and track in-game item purchases. The database UI shows four tables: players, items, games, and the joined table of games and reviews. The three individual tables are the three most central relations to our project, and the operations available for each of them are the ones that make the most sense for each relation.

We managed to accomplish not only a model of what this website's database might look like, but also a number of useful functions that we hadn't previously envisioned using the aggregation and division queries. We accomplished a database that's easy to use even for those with no computer science background, and one that runs smoothly on the back end. Our queries are non-trivial and user friendly.

Description of Schema Differences and Reasons for Changes

We implemented several changes to our schema. Firstly, we simplified the foreign keys, like in the Comments relation, where the table now references both Player and Review. This was done to reduce complexity. Next, we replaced the TEXT and TIME types to CLOB and TIMESTAMP respectively, as they work better with oracle and provide greater versatility. Next, we added NOT NULL constraints to several attributes we felt needed to be present for the relation to make sense, like Review text and time. Finally, we added the Player_Achievement table to represent the many-to-many relationship between players and achievements.

QUERIES

2.1.1 INSERT

```
`INSERT INTO Player (username, followers, following, reviews, achievements)
            VALUES (:username, :followers, :following, :reviews, :achievements)`
[username, followers, following, reviews, achievements],
            { autoCommit: true }
```

On line 202-205 on appServic.js of function insertPlayerToDb(username, followers, following, reviews, achievements) on line 198.

2.1.2 UPDATE

```
`UPDATE ITEM SET name=:newName, price=:newPrice, function=:newFunction where
name=:oldName AND price=:oldPrice AND function=:oldFunction`,
[newName,newPrice,newFunction, oldName, oldPrice, oldFunction],
        { autoCommit: true }
```

On line 219-221 on appService.js of function updateItem(oldName, newName, oldPrice, newPrice, oldFunction, newFunction) on line 215.

<u>2.1.3 DELETE</u>

```
 `DELETE FROM Player WHERE Trim(username) = :username`, [username],
 { autoCommit: true }
```

On line 252-254 on appService.js of function deletePlayer(username) on line 248.

<u>2.1.4 SELECTION</u>

```
`SELECT Price, Name, Genre, Platform, release_year FROM Game WHERE ${conditions}`
```

On line 374 of function getGamesByConditions(conditions) on line 372.

<u>2.1.5 Projection</u>

```
`SELECT ${columnList} FROM Player`;
```

On line 170 of appService.js of function projectPlayerFromDb(columns) on line 166

<u>2.1.6 JOIN</u>

```
`SELECT G.price, G.name, G.genre, G.platform, G.release_year,
        R.rating, R.text, R.time, R.author_username
        FROM Game G
        JOIN Review R
        ON G.name = R.game_name AND G.developing_company =
R.game_developing_company
        WHERE ${where}`;
```

On line 388-393 of appService.js of function joinTable(where) on line 386.

<u>2.1.7 Aggregation with GROUP BY</u>

```
SELECT G.name AS game_name, G.developing_company,  AVG(R.rating) AS average_rating
FROM Game G
JOIN Review R ON G.name = R.game_name AND G.developing_company =
R.game_developing_company GROUP BY G.name, G.developing_company
```

This query finds each game’s average rating
On line 355-361 on appService.js of function getGamesWithAverageRating() on line 352.

2.1.8 Aggregation with HAVING

```
SELECT * FROM Player
WHERE followers >= 50
```

This query finds all players with at least 50 followers
On line 269-271 on appService.js of function getPlayersWithAtLeast50Followers() on line 266.

2.1.9 Nested aggregation with GROUP BY

```
SELECT A.username, COUNT(*) AS num_achievements
FROM Achievement A
GROUP BY A.username
HAVING COUNT(*) > (
   SELECT COUNT(*)
   FROM Achievement A2
   WHERE A2.username = ‘Player1’
);
```

This query finds all players with more achievements than the given player
On line 355-362 on appService.js of function getGamesWithAverageRating() on line 352.

2.1.10 Division

```
SELECT G.name, G.developing_company
FROM Game G
WHERE NOT EXISTS (
   SELECT R2.author_username
   FROM Review R2
   WHERE NOT EXISTS (
      SELECT R1.author_username
      FROM Review R1
      WHERE R1.game_name = G.name
      AND R1.game_developing_company = G.developing_company
      AND R1.author_username = R2.author_username
   )
);
```

This query finds all games that have been reviewed by all players with at least one submitted review.
On line 331-344 on appService.js of function getGamesReviewedByAllUsers() on line 328.