CPSC 304 Project Cover Page

Milestone #: 4

Date: Nov. 29, 2024

Group Number: 84

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Connor Won	27314798	c6c6o	cywon2@gmail.com
Jonathan Cai	37435609	g3f5e	jonathan.y.cai@outlook.com
Harry Zhu	98024730	t8i3b	harrylakers242424@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 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 DESCRIPTION

Pickuple is a sports and recreation app designed to connect pickleball players of all skill levels. Our application allows users to create, organize and join local pickleball games with a simple click of a button. Upon creating an account, users will be taken to a dashboard of games available to join, which can be filtered by location, postal code, city and province. Once users find a game that suits them, they can register and engage with other participants through a unique comment section attached to each game invite. Registered games will show up in the Upcoming Games tab, and after completion they will show up in the History tab. Additionally, Pickuple offers profile customization, enabling users to update their email, password, and profile picture. When finished, users can log out securely and later log back in using their registered email.

SCHEMA DIFFERENCES

Change 1: The first change we made is we got rid of the Clubs entity in our table. Unfortunately, we were unable to find the time to implement this and as it was an additional relation we didn't need to fulfill the rubric, we ultimately decided not to include it.

Change 2: The second change we made was to get rid of SinglesStatus and DoublesStatus. This was initially created to normalize the Singles and Doubles table, but as it did not have a use in our app we decided to get rid of it as well.

Minor Changes:

- Added default value for isActive column in Singles and Doubles
- Added default value for status column in GameInvite
- Added an assertion to ensure currentlyEnrolled <= capacity for Singles and Doubles

SQL QUERIES

INSERT: Line 253 in appService.js - createUser()

UPDATE: Line 419 in appService.js - updateUserInfo()

DELETE: Line 52 in gameService.js - deleteGame()

SELECTION: Line 99 in gameService.js - getGamesToJoin()

- Performs a selection on the Location + CityLocation + ProvinceLocation tables. Note that the selection is a part of a larger joined table and that

the values the user provides can only come from a dynamically generated dropdown.

PROJECTION: Line 31 in gameInviteService.js - getRegisteredUsers()

- Note password column isn't included as one of the possible projected columns because of security reasons

JOIN: Line 318 in appService.js - getCommentsByUser()

AGGREGATION WITH GROUP BY: Line 346 in appService.js - getReservationByPostalCode():

- This query returns the number of Reservations currently associated with each postal code and the corresponding postal code.

```
export async function getReservationByPostalCode() {
    try {
        return await withOracleDB(async (connection) => {
            const result = await connection.execute(
                `
                SELECT
                    r.postalCode,
                    COUNT(*)
                FROM
                    Reservation r
                GROUP BY
                    r.postalCode
                `
            );
            console.log(result.rows);
            return result.rows;
        })
    } catch (error) {
        console.error("Error fetching number of reservations by postal code", error);
        throw new Error("Failed to number of reservations by postal code");
    }
}
```

AGGREGATION WITH HAVING: Line 369 in appService.js - getUserCreateMultGame()

- This query will return all email addresses of users who have created more than one game as well as the number of games each corresponding user has created.

```
export async function getUserCreateMultGame() {
    try {
        return await withOracleDB(async (connection) => {
            const result = await connection.execute(
                `
                SELECT
                U.email,
                COUNT(*) AS num_invites
                FROM GameInvite G
                JOIN UserInfo U ON G.creator = U.userID
                GROUP BY U.email
                HAVING COUNT(*) > 1
                `
            );
            console.log(result.rows);
            return result.rows;
        })
    } catch (error) {
        console.error("Error fetching users who've created more than one game", error);
        throw new Error("Failed to fetch users who've created more than one game");
    }
}
```

NESTED AGGREGATION WITH GROUP BY: Line 392 in appService.js - getUserRegMoreThanAvg()

- This query returns the users who are currently registered to more games than the average number of games every user is currently registered to as well as the number of games the corresponding user is currently enrolled in.

```
export async function getUserRegMoreThanAvg() {
    try {
        return await withOracleDB(async (connection) => {
            const result = await connection.execute(
                `
                SELECT U.email AS "EMAIL", COUNT(R.inviteID)
                FROM UserInfo U
                JOIN registers R ON U.userID = R.userID
                GROUP BY U.email
                HAVING COUNT(R.inviteID) > (
                    SELECT AVG(reg_count) FROM (
                        SELECT COUNT(*) AS reg_count
                        FROM registers
                        GROUP BY userID
                    ) reg_counts
                )
                `
            );
            console.log(result.rows);
            return result.rows;
        });
    } catch (error) {
        console.error("Error fetching users who've registered more than average", error);
        throw new Error("Failed to fetch users who've registered more than average");
    }
}
```

DIVISION: Line 291 in appService.js - getAllRegUsers()

- This query gets all users who are currently enrolled in every single available game. It will return the userID.

```
export async function getAllRegUsers() {
    try {
        return await withOracleDB(async (connection) => {
            const result = await connection.execute(
                `
                SELECT u.userID
                FROM UserInfo u
                WHERE NOT EXISTS (
                    SELECT g.inviteID
                    FROM GameInvite g
                    MINUS
                    SELECT r.inviteID
                    FROM registers r
                    WHERE r.userID = u.userID
                )
                `
            );
            console.log("Query Result", result.rows);
            return result.rows;
        })
    } catch (error) {
        console.error("Error fetching unregistered users", error);
        throw new Error("Failed to get unregistered users");
    }
}
```

ASSERTIONS:

- Line 119 & 131 in setupdb.sql
- Line 169 in setupdb.sql
- Line 201 in setupdb.sql