University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: ____4____

Date: ___2024-11-29___

Group Number: ____70____

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
April Cao	72028764	g4q6w	aprilcao2002@gmail.com
Sherry He	94345741	r8k8g	sherryhe1107@gmail.com
Xinya Lu	88957790	e1e1a	xinyalu13@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

---

2. A short description of the final project, and what it accomplished.

The final project allows users to view a list of hikers, hiking clubs, and trails. Users can add a new hiker or update an existing one. They can also delete a hiker or search for a hiker that satisfies certain conditions specified by the user. Users can choose what information they want to see for all hikers (e.g., email and names). Additionally, they can find hikers who have hiked a specific trail. Users can also add a new hiking club or delete an existing one.

Users can view the average number of trails completed by hikers in each experience level, the maximum number of trails completed by hikers in large hiking clubs, a list of hikers whose number of completed trails exceeds the average for their experience level, and a list of hikers who have hiked all mountains.

3. A description of how your final schema differed from the schema you turned in. If the final schema differed, explain why.

There is only one minor change: Oracle does not support the TIME data type, so we changed the data type of the EstimatedTime attribute in the Have_Trails1 table and the CompletionTime attribute in the Hike table to INTERVAL DAY TO SECOND.

4. A list of all SQL queries used to satisfy the rubric items and where each query can be found in the code (file name and line number(s)).
5. For SQL queries 2.1.7 through 2.1.10 inclusive, include a copy of your SQL query and a maximum of 1-2 sentences describing what that query does. You can embed this in your above list of queries. You don't need to include the output of the query. The purpose of this requirement is to allow your TAs time outside of your presentation to verify these more complex queries are well formed.

The single SQL script file, which includes all DROP TABLE statements, tables, and data, is named sql_script_for_testing.

We did not implement the following constraints, but they should have been implemented as described:

- ON UPDATE CASCADE
- In the Have_Organizers1 relation (OrganizerEmail, Name, NumOfEventsOrganized, ClubEmail), we should have implemented an assertion to enforce the total participation constraint on the 'one' side (Hiking Clubs). However, Oracle does not support CREATE ASSERTION that we learned in class:
- -- assertion that enforces every hiking club must have at least one organizer

```
CREATE ASSERTION CheckClubHasOrganizer
CHECK
(NOT EXISTS (SELECT ClubEmail FROM HikingClubs)
        EXCEPT
        (SELECT ClubEmail FROM Have_Organizers1));
```

- In the Join_Hikers1 relation (HikerEmail, Name, **NumofTrailsCompleted**, **ClubEmail**), we should have implemented an assertion to enforce the total participation constraint on the 'one' side (Hiking Clubs). However, Oracle does not support CREATE ASSERTION that we learned in class:

```
-- assertion that enforces every hiking club must have at least one hiker
CREATE ASSERTION CheckClubHasHiker
CHECK
(NOT EXISTS (SELECT ClubEmail FROM HikingClubs)
        EXCEPT
        (SELECT ClubEmail FROM Join_Hikers1));
```

Non-hardcoded:

2.1.1 INSERT
- File name: appService.js
- Line numbers: 253-254 (insert hikers), 330-331 (insert hiking clubs)

2.1.2 UPDATE
- File name: appService.js
- Line numbers: 433-435 (update hikers), 460-463 (update hiking clubs)

2.1.3 DELETE
- File name: appService.js
- Line numbers: 488 (delete hikers), 512 (delete hiking clubs)

2.1.4 Selection
- File name: appService.js
- Line numbers: 558, 587

2.1.5 Projection
- File name: appService.js
- Line numbers: 610

2.1.6 Join
- File name: appService.js
- Line numbers: 639-644

Hardcoded: (These queries can also be found in the file named sql_queries.sql.)

2.1.7 Aggregation with GROUP BY
- File name: appService.js
- Line numbers: 672-675

```
-- Find the average number of trails completed by hikers for each experience level
```

```
SELECT ExperienceLevel, AVG(NumofTrailsCompleted) AS AVGNumofTrailsCompleted
FROM Join_Hikers2
GROUP BY ExperienceLevel
ORDER BY ExperienceLevel;
```

2.1.8 Aggregation with HAVING

- File name: appService.js
- Line numbers: 701-706

```
-- For each club having more than 50 members (big clubs), find the club email
--     and the max number of trails completed by hikers in that club.
SELECT c.ClubEmail, MAX(h.NumofTrailsCompleted) AS MaxTrails
FROM HikingClubs c, Join_Hikers1 h
Where c.ClubEmail = h.ClubEmail
GROUP BY c.ClubEmail
HAVING SUM(c.NumofMembers) > 50
ORDER BY c.ClubEmail ASC;
```

2.1.9 Nested aggregation with GROUP BY

- File name: appService.js
- Line numbers: 732-741

```
-- Find the hiker for each experience level whose number of completed trails is
--     greater than the average number of completed trails for that experience level.
SELECT h.HikerEmail, h.Name, h1.ExperienceLevel
FROM Join_Hikers2 h1, Join_Hikers1 h
WHERE h1.NumofTrailsCompleted = h.NumofTrailsCompleted AND
   h1.NumofTrailsCompleted >= ALL (
   SELECT AVG(h2.NumofTrailsCompleted)
   FROM Join_Hikers2 h2
   WHERE h2.ExperienceLevel = h1.ExperienceLevel
   GROUP BY h2.ExperienceLevel)  -- find the avg NumofTrailsCompleted per ExperienceLevel
ORDER BY h1.ExperienceLevel;
```

2.1.10 Division

- File name: appService.js
- Line numbers: 767-776

```
-- Find hikers who have hiked all the mountains.
SELECT UNIQUE h.HikerEmail, j.Name
FROM Hike h, Join_Hikers1 j
WHERE h.HikerEmail = j.HikerEmail AND NOT EXISTS (
      (SELECT m.Latitude, m.Longitude
       FROM Mountains m)  -- 1. all mountains
      MINUS  -- 3. all mountains that have not been hiked by hiker
      (SELECT h2.Latitude, h2.Longitude
       FROM Hike h2
       WHERE h2.HikerEmail = h.HikerEmail)  -- 2. all mountains hiked by hiker
```

```
);
```