University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: ___4____

Date: ____2025-04-01_____

Group Number: ____77_____

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Amy Xiong	97765291	m8o7v	amyxiongg@gmail.com
Matthew Haryanto	24695686	i7q9t	matthewanh10@gmail.com
Sadra Khosravi	90431511	o3v1b	sadrakh@outlook.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

University of British Columbia, Vancouver
Department of Computer Science

Final Project Description

Our project, Accelera, is a startup support platform built with Next.js and Supabase. The goal was to simulate a realistic startup ecosystem that allows startups to connect with investors, resources, accelerators, and find events. We implemented a clean and consistent UI, and each of the queries required by the milestone have a dedicated page that interacts with our database. Users can sign up, log in securely, manage their profiles, explore and search for events, and interact with various features. We implemented 10 core SQL queries as required.

Schema Differences

Our final schema evolved from the one we submitted in Milestone 2 in a few ways. Initially, we planned to normalize the startup table into BCNF, but we decided against this for simplicity and practicality. This was especially true when implementing queries involving accesses, attends, and managed_by.

SQL Queries

Number	Type	Description	File Name	Line Numbers
1	Insert	Signup: Inserts new user into users table	/src/actions/users/signup-action.ts	64-67
2	Update	Allows users to update their email	/src/actions/users/update-user.ts	26-29
3	Delete	Deletes a user's account by email	/src/actions/users/delete-user-action.ts	14-16
4	Selection	Filter events by type or location	/src/actions/events/filter-events.ts	50-53

5	Projection	Choose which event attributes to view		
6	Join	Join Startup and Event relation to see which startup attended the given event	/src/actions/join/join-event.ts	21-25
7	Aggregation with Group By	Group Investor Group to see the order of the average capital amongst its investors from highest to lowest	/src/actions/group/group-capital.ts	14-20
8	Aggregation With Having	Finds average capital of investors in descending order	/src/actions/group/group-investor.ts	24-28
9	Nested Aggregation With Group By	Find startups that accessed more resources than the average	/src/actions/nested-aggregation/nested-aggregation.ts	17-28
10	Division	Find the startup that attended all events that are present in the database	/src/actions/division/get-startup-all-events.ts	13-17

Queries 7-10:

7. Aggregation with GROUP BY

This query finds the average capital of investors grouped by their investor group and orders the results from highest to lowest average capital.

```
SELECT ig.name, AVG(i.capital)
FROM investors i, investor_group ig, belongs_to b
WHERE i.user_id = b.user_id AND ig.invest_group_id = b.invest_group_id
GROUP BY ig.name
ORDER BY AVG(i.capital)
DESC
```

8. Aggregation with HAVING

This query counts how many funding rounds each user has participated in and only shows users who meet or exceed a specified minimum number.

```
SELECT u.name, COUNT(*)
FROM users u, investors i, funding_round f, startup s
WHERE u.user_id = i.user_id AND i.user_id = f.user_id AND f.startup_id = s.startup_id
GROUP BY u.name
HAVING COUNT(*) >= ${minimum}
```

9. Nested Aggregation with GROUP BY

This query finds startups that have accessed more resources than the average number of accesses across all startups. It uses nested aggregation with GROUP BY and HAVING.

```
SELECT s.startup_id, s.name, COUNT(a.resource_id) AS access_count
FROM startup s, accesses a
WHERE s.startup_id = a.startup_id
GROUP BY s.startup_id, s.name
HAVING COUNT(a.resource_id) > (
  SELECT AVG(total_accesses)
  FROM (
    SELECT COUNT(*) AS total_accesses
    FROM accesses
    GROUP BY startup_id
  ) AS access_counts
);
```

10. Division

This query finds startups that have attended all events. It checks that there is no event the startup has not attended.

```
SELECT s.name
FROM startup s
WHERE NOT EXISTS ((SELECT e.event_id FROM event e)
                  EXCEPT
                  SELECT a.event_id FROM attends a WHERE s.startup_id = a.startup_id)
```