University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: ___4___

Date: __Nov 24, 2025_

Group Number: _55__

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Windy Huang	68306661	c1y3i	windyuniapp@gmail.com
Kylie Wang	69075752	o8s0u	kyliewang025@gmail.com
Zhengyang Xu	65484644	v0u7d	xuzhengyang2023@gmail.com

University of British Columbia, Vancouver
Department of Computer Science

Project summary:

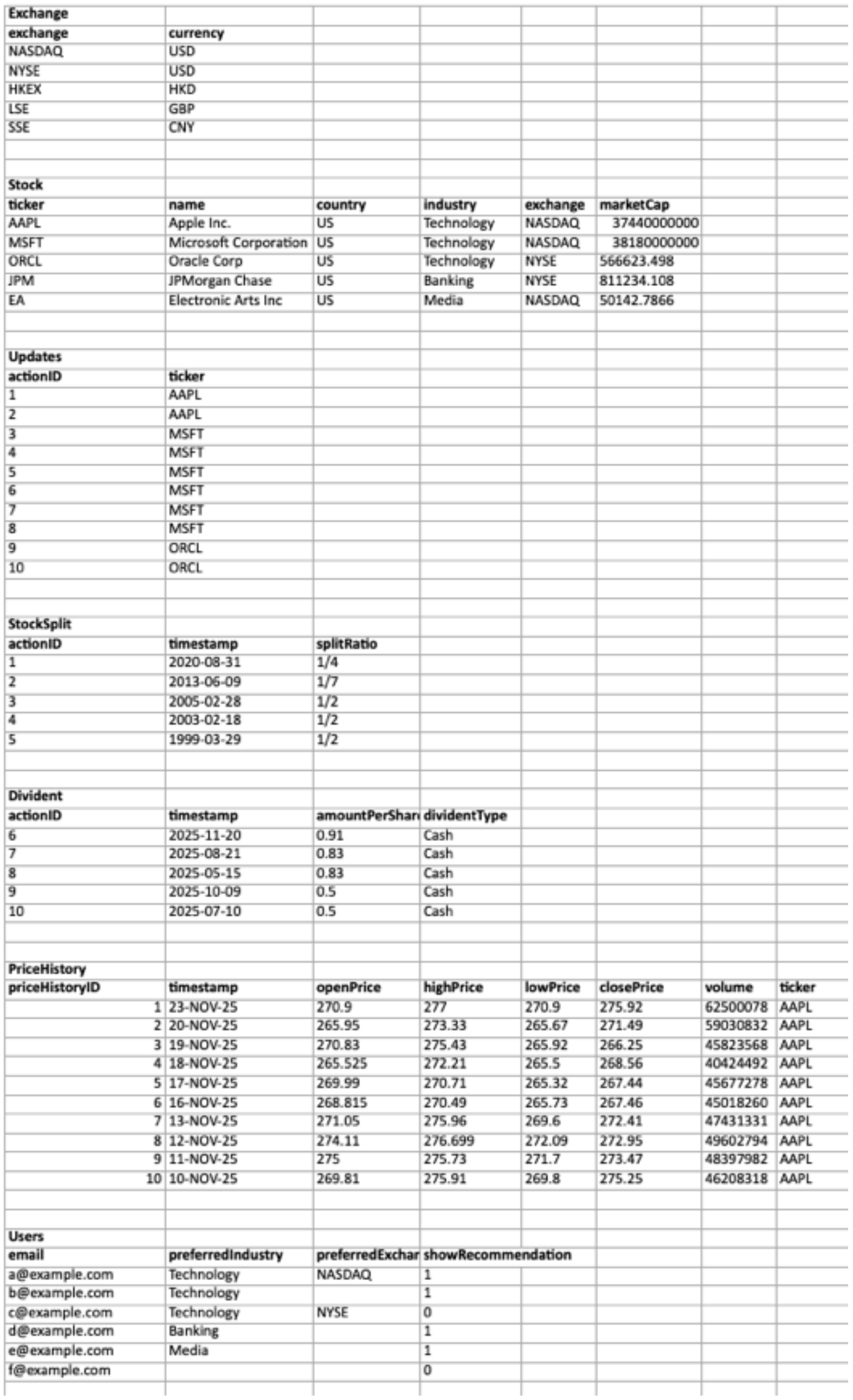
Our project provides recommendations to people on which stocks to sell or buy. People using this application would be able to make more informed decisions when manipulating their stock market portfolio.

Schema differences:

We modified our final schema to better align with our implemented features.

1. Removed the Company table: Based on Milestone 2 feedback, we learned that the 1 to 1 relationship between Company and Stock allows us to merge the 2 tables together.
2. Modified attributes of Users table: The updated attributes reflect the actual information required by our queries and our UI.
3. Added an attribute in Holds table: We added a new attribute to support potential time-based features.
4. Simplified attributes of AnalystRating table: We simplified the representation from multiple percentage fields and a target price to a single numeric recommendation, because computing and using exact buy/hold/sell percentages turned out to be unnecessarily complex for our purposes.

Mocked results:



Exchange

exchange	currency
NASDAQ	USD
NYSE	USD
HKEX	HKD
LSE	GBP
SSE	CNY

Stock

ticker	name	country	industry	exchange	marketCap
AAPL	Apple Inc.	US	Technology	NASDAQ	37440000000
MSFT	Microsoft Corporation	US	Technology	NASDAQ	38180000000
ORCL	Oracle Corp	US	Technology	NYSE	566623.498
JPM	JPMorgan Chase	US	Banking	NYSE	811234.108
EA	Electronic Arts Inc	US	Media	NASDAQ	50142.7866

Updates

actionID	ticker
1	AAPL
2	AAPL
3	MSFT
4	MSFT
5	MSFT
6	MSFT
7	MSFT
8	MSFT
9	ORCL
10	ORCL

StockSplit

actionID	timestamp	splitRatio
1	2020-08-31	1/4
2	2013-06-09	1/7
3	2005-02-28	1/2
4	2003-02-18	1/2
5	1999-03-29	1/2

Divident

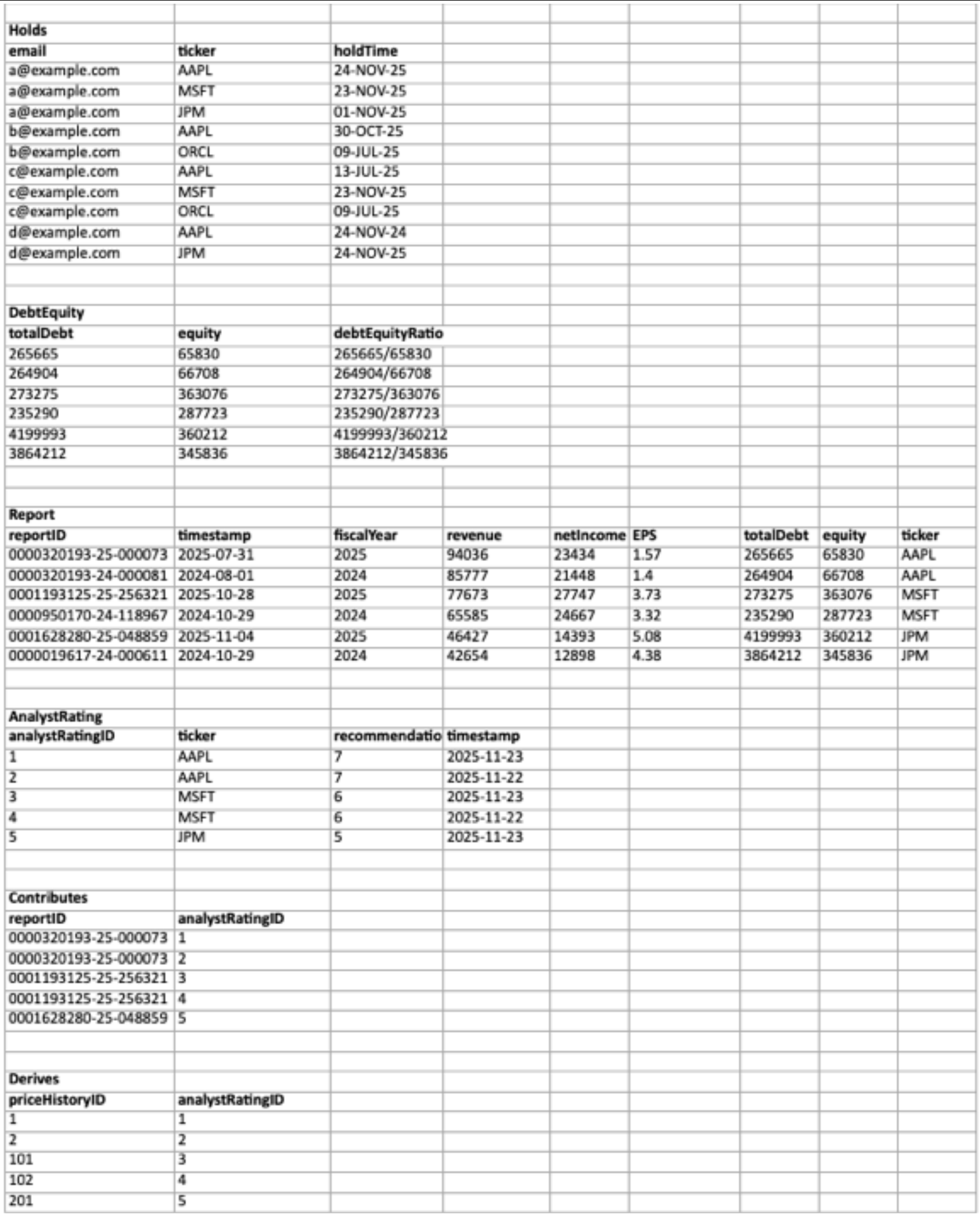
actionID	timestamp	amountPerShar	dividentType
6	2025-11-20	0.91	Cash
7	2025-08-21	0.83	Cash
8	2025-05-15	0.83	Cash
9	2025-10-09	0.5	Cash
10	2025-07-10	0.5	Cash

PriceHistory

priceHistoryID	timestamp	openPrice	highPrice	lowPrice	closePrice	volume	ticker
1	23-NOV-25	270.9	277	270.9	275.92	62500078	AAPL
2	20-NOV-25	265.95	273.33	265.67	271.49	59030832	AAPL
3	19-NOV-25	270.83	275.43	265.92	266.25	45823568	AAPL
4	18-NOV-25	265.525	272.21	265.5	268.56	40424492	AAPL
5	17-NOV-25	269.99	270.71	265.32	267.44	45677278	AAPL
6	16-NOV-25	268.815	270.49	265.73	267.46	45018260	AAPL
7	13-NOV-25	271.05	275.96	269.6	272.41	47431331	AAPL
8	12-NOV-25	274.11	276.699	272.09	272.95	49602794	AAPL
9	11-NOV-25	275	275.73	271.7	273.47	48397982	AAPL
10	10-NOV-25	269.81	275.91	269.8	275.25	46208318	AAPL

Users

email	preferredIndustry	preferredExchar	showRecommendation
a@example.com	Technology	NASDAQ	1
b@example.com	Technology		1
c@example.com	Technology	NYSE	0
d@example.com	Banking		1
e@example.com	Media		1
f@example.com			0



Holds

email	ticker	holdTime
a@example.com	AAPL	24-NOV-25
a@example.com	MSFT	23-NOV-25
a@example.com	JPM	01-NOV-25
b@example.com	AAPL	30-OCT-25
b@example.com	ORCL	09-JUL-25
c@example.com	AAPL	13-JUL-25
c@example.com	MSFT	23-NOV-25
c@example.com	ORCL	09-JUL-25
d@example.com	AAPL	24-NOV-24
d@example.com	JPM	24-NOV-25

DebtEquity

totalDebt	equity	debtEquityRatio
265665	65830	265665/65830
264904	66708	264904/66708
273275	363076	273275/363076
235290	287723	235290/287723
4199993	360212	4199993/360212
3864212	345836	3864212/345836

Report

reportID	timestamp	fiscalYear	revenue	netIncome	EPS	totalDebt	equity	ticker
0000320193-25-000073	2025-07-31	2025	94036	23434	1.57	265665	65830	AAPL
0000320193-24-000081	2024-08-01	2024	85777	21448	1.4	264904	66708	AAPL
0001193125-25-256321	2025-10-28	2025	77673	27747	3.73	273275	363076	MSFT
0000950170-24-118967	2024-10-29	2024	65585	24667	3.32	235290	287723	MSFT
0001628280-25-048859	2025-11-04	2025	46427	14393	5.08	4199993	360212	JPM
0000019617-24-000611	2024-10-29	2024	42654	12898	4.38	3864212	345836	JPM

AnalystRating

analystRatingID	ticker	recommendatio	timestamp
1	AAPL	7	2025-11-23
2	AAPL	7	2025-11-22
3	MSFT	6	2025-11-23
4	MSFT	6	2025-11-22
5	JPM	5	2025-11-23

Contributes

reportID	analystRatingID
0000320193-25-000073	1
0000320193-25-000073	2
0001193125-25-256321	3
0001193125-25-256321	4
0001628280-25-048859	5

Derives

priceHistoryID	analystRatingID
1	1
2	2
101	3
102	4
201	5

Query 1:

```
INSERT INTO Report
VALUES (:1, :2, :3, :4, :5, :6, :7, :8, :9)
```

In file: appService.cjs, function: insertReportPerCompany, lines: 509-510

- Allows a user to insert a new report by specifying an access number (PK)
- If the system can auto-parse the report (for appropriate attributes), it is inserted directly into the database
- If the report cannot be parsed automatically, the UI
    - Displays the relevant attributes
    - Prompts the user to manually provide the missing values before insertion
- Handles error that PK already exist, or FK does not exist

Query 2:

```
UPDATE Users
SET preferredIndustry = :industry, preferredExchange = :exchange,
showRecommendation = :rec
WHERE email = :email
```

In file: appService.cjs, function: updateUser, lines: 579-581

- Users log in by specifying their email (PK)
- If the email does not exist, a new User row is created with that email and NULL attributes
- A logged-in user can update any of their preference attributes (Ex: preferred industry, preferred exchange, show recommendation flag)
- In our design, Users referencing Exchange, so in a DBMS that supports update cascade, if an exchange updates, all corresponding Users values would automatically be updated

Query 3:

```
DELETE FROM Users
WHERE email = :1`
```

In file: appService.cjs, function: delUser, lines: 599-600

- Once logged in, the user can delete their user account

- Implemented by deleting from the Users table, and their Holds entries will be deleted on cascade

Query 4:

```
SELECT * FROM Stock WHERE ${where}
```

In file: appService.cjs, function: filterStock, lines: 621

- Adds a search bar with:
    - A dropdown to choose which attribute to search on
    - A text input for the search term
- The backend performs a selection on rows whose chosen attribute contains the input substring
- Results are dynamically populated below the search bar
- Users can append multiple conditions using the AND/OR dropdown

Query 5:

```
SELECT ${fields}
FROM PriceHistory
WHERE ticker = :1
ORDER BY timestamp DESC
FETCH FIRST 1 ROW ONLY
```

In file: appService.cjs, function: fetchRecentPriceHistory, lines: 870-874

- When the user selects a stock, the attributes of the most recent priceHistory entry for that stock are rendered
- The UI allows the user to hide individual attributes (via an X button) and requery the database, so they only see what they care about

Query 6:

```
SELECT h.ticker, s.name
FROM Holds h
JOIN Stock s ON h.ticker = s.ticker
WHERE h.email = :email
ORDER BY s.ticker
```

In file: appService.cjs, function: getUserHeldStocks, lines: 720-724

- When a logged-in user opens their portfolio:

  - The backend joins the Holds and Stock tables on ticker
  - The user's email is used in the WHERE clause to select only their holdings
- Resulting rows include both holding and stock details, displayed by the portfolio UI

Our project provides recommendations to people on which stocks to sell or buy. People using this application would be able to make more informed decisions when manipulating their stock market portfolio.

Query 7:

```
`SELECT ${type}, COUNT(*) FROM Stock GROUP BY ${type}`
```

In file: appService.cjs, function: fetchSettingDropdown, lines: 547

- When a user chooses an exchange or industry to watch, the backend runs a GROUP BY query that counts the number of tickers in each selected category

Query 8:

```
SELECT h.ticker, s.name, ROUND(SYSDATE - h.holdTime, 2) as holdDays
FROM Holds h
JOIN Stock s ON h.ticker = s.ticker
WHERE h.email = :1
GROUP BY h.ticker, s.name, h.holdTime
HAVING (SYSDATE - h.holdTime) >= :2
ORDER BY holdDays DESC
```

In file: appService.cjs, function: getUserHeldStocksByDuration, lines: 846-852

- Adds a dropdown for users to choose a minimum holding duration (Ex: 1 day)
- When the user selects a duration and applies the filter:
  - The backend groups holdings by ticker and calculates the holding duration
  - A HAVING clause filters to keep only stocks held longer than the selected threshold
- The filtered list of stocks is displayed, so users can quickly see which positions they've held long-term

Query 9:

```
SELECT s.ticker
FROM Stock s LEFT JOIN Holds h ON h.ticker = s.ticker
WHERE s.industry = :1
GROUP BY s.ticker
```

```
HAVING COUNT(*) <= ALL (
    SELECT COUNT(*)
    FROM Stock s1 LEFT JOIN Holds h1 ON h1.ticker = s1.ticker
    WHERE s1.industry = :1
    GROUP BY s1.ticker
)
```

In file: appService.cjs, function: fetchLeastPopularStock, lines: 652-660

- Implements the least popular recommendation:
    - Least popular is defined as the stock(s) within the user's preferred industry that are held by the fewest number of users
- Users can choose to watch or unwatch this recommendation in user settings

Query 10:

```
SELECT DISTINCT h.ticker
FROM Holds h
WHERE NOT EXISTS (
    (SELECT u.email FROM Users u)
    MINUS
    (SELECT h1.email FROM Holds h1 WHERE h1.ticker = h.ticker)
)
```

In file: appService.cjs, function: fetchPopularStock, lines: 633-639

- Implements the most popular recommendation:
    - Most popular is defined as the stock(s) that are held by all users of the application
- Users can choose to watch or unwatch this recommendation in user settings