CPSC 304 2015W2
Project Final

Group Members

Name	Student No.	Unix ID	Email
Albert Xing	40640104	z6k8	albert.xing@alumni.ubc.ca
Calvin Cheng	36090132	o7x8	calvin.cheng@alumni.ubc.ca
Kyle Stadnyk	52749025	b5p5	b5p5@ugrad.cs.ubc.ca

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 Code

All the code required is in the project-final directory.

Script Used to Create Tables

All the data to seed the database can be found in project-final/db/migrate/.
The code to insert the data is in project-final/db/seeds.rb.

Project Description

We have built a financial portfolio management application that allows investors and financial advisors track and manage portfolios of stocks. In addition to tracking portfolio holdings and performance, the application enables financial advisors to add and remove investors and stocks.

Changes in Database Schema

All primary keys in all tables were replaced with a unique “id” code. This change was made to improve efficiency; looking up unique integers is faster than looking up a combination of strings. Additional changes are listed in Functional Dependencies and Database Schema below.

List of SQL Queries Used

1. **Selection:** select portfolios with principal or cash > or < [X]
   SELECT * from portfolio
   WHERE principal > [user input]

   E.g. show portfolios with cash < $200 AND principal >= $10,000 AND principal <= $500,000:

   SELECT "portfolios".* FROM "portfolios" WHERE (cash < 200 AND principal >= 10000 AND principal <= 500000)

2. **Projection**: Show/hide all attributes.

   SELECT [attributes checked by user] from [current page]

   E.g.
   SELECT "portfolios"."id", "portfolios"."purpose", "portfolios"."creation_date", "portfolios"."principal", "portfolios"."cash", "portfolios"."manager_id" FROM "portfolios" WHERE (cash < 200 AND principal >= 10000 AND principal <= 500000)

3. **Join**: show all companies a user is invested in.

   SELECT S.id, S.symbol, S.exchange_id, S.name
   FROM users U
   INNER JOIN portfolios P
   ON U.id = P.owner_id
   INNER JOIN holdings H
   ON H.portfolio_id = P.id
   INNER JOIN Stocks S
   ON S.id = H.stock_id
   WHERE U.id = ?

   E.g. Show all companies a Calvin is invested in:
   SELECT DISTINCT S.id, S.symbol, S.exchange_id, S.name
   FROM users U
   INNER JOIN portfolios P
   ON U.id = P.owner_id
   INNER JOIN holdings H
   ON H.portfolio_id = P.id
   INNER JOIN Stocks S

```
ON S.id = H.stock_id
WHERE U.id = 1"
```

4. **Division**: Show all portfolios whose holdings consist of all of some specified list of stocks.

```
SELECT P.id
FROM portfolios P
WHERE NOT EXISTS (SELECT S.id
FROM stocks S
WHERE symbol IN ?
EXCEPT
SELECT H.stock_id
FROM holdings H
WHERE H.portfolio_id = P.id);
```

E.g. Show all portfolios whose holdings consist of all of "AAPL, FB, MSFT, GOOGL”

```
SELECT P.id
FROM portfolios P
WHERE NOT EXISTS (
SELECT S.id
FROM stocks S
WHERE symbol IN ('AAPL','FB','MSFT','GOOGL')
EXCEPT
SELECT H.stock_id
FROM holdings H
WHERE H.portfolio_id = P.id);
```

5. **Aggregation:** Show min, max, average, or count.

E.g.
Show total number of users (including admins and advisors):

```
SELECT COUNT(*) FROM "users"
```

Show number of admins:

```
SELECT COUNT(*) FROM "users" WHERE "users"."role" = $1  [["role", 2]]
```

Show number of advisors:

```
SELECT COUNT(*) FROM "users" WHERE "users"."role" = $1  [["role", 1]]
```

Show average portfolio principal:
SELECT AVG("portfolios"."principal") FROM "portfolios"

Show average portfolio cash:
SELECT AVG("portfolios"."cash") FROM "portfolios"

Show portfolio with minimum principal:
SELECT MIN("portfolios"."principal") FROM "portfolios"

Show portfolio with max principal:
SELECT MAX("portfolios"."principal") FROM "portfolios"

6. **Nested aggregation with group-by:** Minimum/maximum of average portfolio value in each country.

```
SELECT country, AVG(#{attr})
  FROM users U
  INNER JOIN portfolios P
    ON U.id = P.owner_id
  INNER JOIN addresses A
    ON A.id = U.address_id
  GROUP BY country
  ORDER BY AVG(#{attr}) #{order}
```

E.g. show average principal in each country:

```
SELECT country, AVG(principal)
FROM users U
INNER JOIN portfolios P
ON U.id = P.owner_id
ON A.id = U.address_id
INNER JOIN addresses A
GROUP BY country
ORDER BY AVG(principal) DESC
```

7. **Delete**: Delete user.

```
DELETE FROM user
WHERE name = ‘Smith’
```

E.g. Delete Serena Nelson:
DELETE FROM "users" WHERE "users"."id" = $1 [["id", 21]]

Functional Dependencies and Database Schema

stocks(id: INT, symbol: CHAR(6), exchange_id: INT, name: VARCHAR(64))

Primary key: (id)
Foreign key: exchange_id references exchanges
Functional dependencies: id → symbol, exchange_id, name
Each stock has a unique ID that determines the stock symbol, exchange and name.

symbol, exchange_id → name
The combination of stock symbol and exchange name determine stock name.

Changes: replaced “Exchange” field with “exchange_id” which now references the exchanges table.

holdings(id: INT, porfolio_id: INT, stock_id: INT, num_shares, INT, purchase_date: DATE, price: INT)

Primary key: (id)
Foreign key: porfolio_id references portfolios
Foreign key: stock_id references stocks
Functional dependencies: id → portfolio_id, stock_id, num_shares, purchase_date, price
Each holding has a unique ID that determines what the number of shares of a stock inside each portfolio with a purchase date and price (in USD).

Changes: replaced “Symbol” field with “stock_id” field. Replaced “Exchange” field with “exchange_id” which now references the exchanges table.

portfolios(id: INT, purpose: VARCHAR(64), creation_date: DATE, principal: INT, cash: INT, owner_id: INT, manager_id: INT)

Primary key: (id)
Foreign key: owner_id references users
Foreign key: manager_id references users
Functional dependencies: id →purpose, creation_date, principal, cash, owner_id, manager_id
Each portfolio has a unique ID that determines its purpose (why the portfolio was made), its creation date, the principal invested, the cash in the portfolio, the owner and the manager.

users(id: INT, email: VARCHAR(64), name: VARCHAR(64), role: INT, phone: CHAR(10), address_id: INT)

Primary key: (id)

Foreign key: address_id references addresses
Functional dependencies: id → email, name, role, phone, address_id
Each user has a unique ID that determines the user's email address, name, role (user or manager), phone number and address.

Changes: added "role" field to identify investors vs advisors.

addresses(id: INT, street_address: VARCHAR(64), city: VARCHAR(64), country: VARCHAR(64), postal_code: VARCHAR(64))
Primary key: (id)
Functional dependencies: id → street_address, city, country, postal_code
Each address has a unique ID that determines its street address (includes street number, P.O. box and apartment number), city, country and postal code. Note postal code field also includes zip codes if the resident is from the US.

Changes: "Number" and "Street" fields merged into 1 field titled "street_address" to account for apartments and P.O. Boxes.

exchanges(id: INT, code: VARCHAR(64), name: VARCHAR(64))
Primary key: (id)
Functional dependencies: id → code, name
Each exchange has a unique ID that determines the exchange name and the exchange code (short form for its name).

Changes: this table was added so we can include the full exchange name (e.g. "New York Stock Exchange") as well as the exchange code.