GROUP-6 PS2: Documentation

Url: https://web06.cs.ait.ac.th/ (offline)
Gitlab repo: https://gitlab.com/ait-fsad-2022/web6/PS2

Selecting and creating new categories

In order to select existing category names belonging to a Quotation model, we use the shortcut “pluck”. On the view we use it this way:
<%= form.select :category, Quotation.distinct.pluck(:category) %>

To add a new category, we create a field with a new parameter e.g “new_category”
<%= text_field_tag(:new_category) %>

In the controller, we check for the presence of that parameter, if true, we replace category parameter with new_category one

```
if(params[:new_category].present?)
        new_quote_params = quotation_params
        new_quote_params[:category] = params[:new_category]
      end
      @quotation = Quotation.new(new_quote_params)
```

With new_quote_params being a method in which we perform some validation

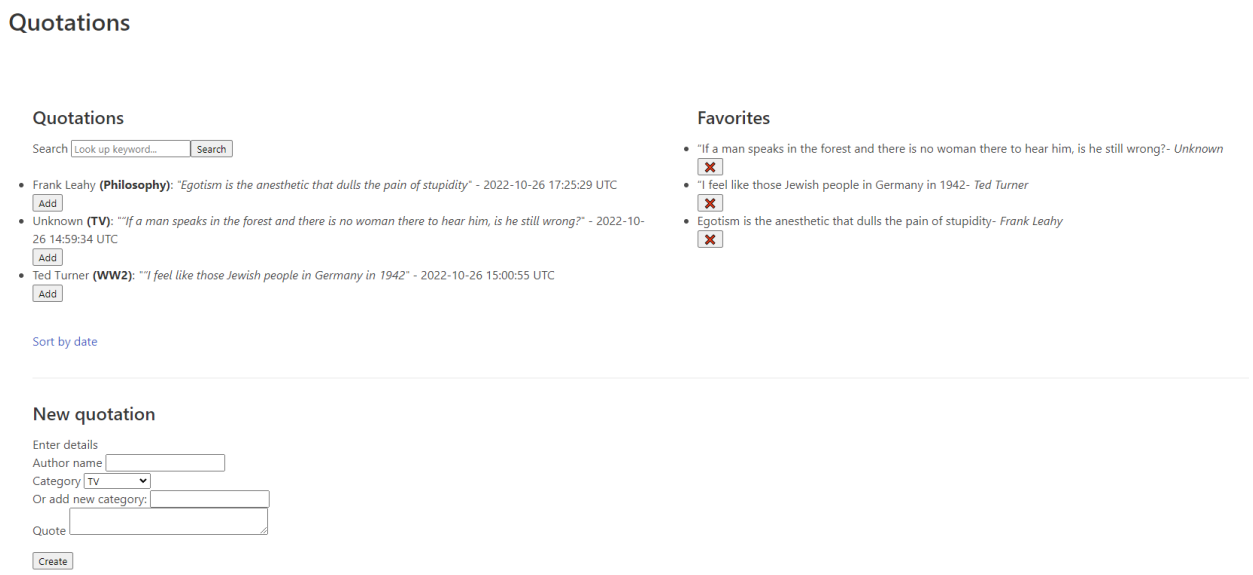
Filter for keywords

Simply put, in the controller we check if a param name of keyword is present, which is sent from view form and use the ActiveRecord query .where in which we pass the parameter value

```
if params[:keyword]
      @quotations = Quotation.where("quote like ?",
"%#{params[:keyword]}%")
    end
```

Saving quotes in browser as ‘favorite quotes’ using session

The user is able to click a button if they would like to save that quote as favorite



How we achieved it:

We set a session hash as empty array in a helper method called current_favorites

```
def current_favorites
    session[:favorites] ||= []
  end
```

On the view, a form sends a parameter named *favorite* containing the id of the quote we clicked on. On the controller, we find the quotation with that id and add it to our session

```
def add_to_favorites
    @favorite = Quotation.find(params[:favorite])
    if current_favorites.all? { |fav| fav["id"].to_s != params[:favorite]
}
      current_favorites << @favorite
    end
  end
```

To remove, we do similar action where we filter the array for ids not equal to the one that was sent in params

```
session[:favorites] = session[:favorites].filter { |fav| fav["id"].to_s !=
params[:remove_favorite] }
```

Finally, to clear the session, we just set it as empty array

```
session[:favorites] = []
```

Exporting favorites as XML/Json

We then allow the user to export the saved quotations into XML or JSON file by checking the presence of form parameters and use rails helpers **to_xml** and **to_json** and **send_data** to allow the user to save the files.

```
if params[:export_xml]
      xmlfile = current_favorites.to_xml
      send_data xmlfile, :filename => "quotes.xml", :type =>
"application/xml"
    end

    if params[:export_json]
      jsonfile = current_favorites.to_json
      send_data jsonfile, :filename => "quotes.json", :type =>
"application/json"
    end
```

Importing Favorites

We allow the user to import quotations saved by someone else using the same Quotation model by using **file_field** element on the view which allows the user to upload a file.
In the controller, we check if the file type is either text/xml or json then Hash it before saving it into the database

```
file_data = params[:attachment]
      if file_data.respond_to?(:read)
        contents = file_data.read
        if params[:attachment].content_type == 'text/xml'
          hash = Hash.from_xml(contents)['objects']
        else
          hash = JSON.parse(contents)
        end
```

Future improvements

The way we built it doesn't allow saving into the database if the existing quotation ID already exists. What if the server the XML originated from has an id of 1 and

we already have a quotation using that id, although quotations saved are different?
We didn't implement a file sanitization function either, which makes the app vulnerable for any malicious uploads.

SQL (Database Session)

7 solution:

After creating the postgresql database in rails, use that database using the following psql command.

psql -d ps2_development

Created a table **my_stocks** with the following parameters:

```
create table my_stocks (
symbol varchar(20) not null,
n_shares integer not null,
date_acquired date not null
);
```

After the creation of table, created a text file with the name file containing the following data and the fields separated by a tab.

```
Ibm     400     08-28-2008
tesla   800     04-22-2020
google  15000   08-17-2004
oracle  2100    10-26-2022
intel   10000   01-19-2010
```

Loaded the data from the text file into the database using the following command :

\COPY my_stocks FROM '/home/conqueror/FSAD/file';

Output after loading the data into **my_stocks** table.

```
ps2_development=# SELECT * FROM my_stocks;
 symbol | n_shares | date_acquired
--------+----------+---------------
 ibm    |      400 | 2008-08-28
 tesla  |      800 | 2020-04-22
 google |    15000 | 2004-08-17
 oracle |     2100 | 2022-10-26
 intel  |    10000 | 2010-01-19
 nasdaq |     4000 | 2015-04-23
(6 rows)
```

8 Solution:

Created a table **stock_prices** with the following parameters:

```
create table stock_prices (
symbol varchar(20) not null,
quote_date date not null,
price float not null
);
```

The table **stock_prices** is filled with symbol from **my_stocks** table and we have filled the other two columns with the current_date and 31.415 as specified in the problem set.

```
insert into stock_prices
select symbol,current_date as quote_date, 31.415 as price from
my_stocks;
```

Output after inserting the data into **stock_prices** table.

```
ps2_development=# SELECT * FROM stock_prices;
 symbol | quote_date | price
--------+------------+--------
 ibm    | 2022-10-26 | 31.415
 tesla  | 2022-10-26 | 31.415
 google | 2022-10-26 | 31.415
 oracle | 2022-10-26 | 31.415
 intel  | 2022-10-26 | 31.415
(5 rows)
```

Created a new table **newly_acquired_stocks** with the following parameters:

create table newly_acquired_stocks (
symbol varchar(20) not null,
n_shares integer not null,
date_acquired date not null
);

The table **newly_acquired_stocks** is filled with the data from the **my_stocks** table using the condition that the number of stocks are less than 10000

insert into newly_acquired_stocks select symbol,n_shares,date_acquired from my_stocks where n_shares < 10000

Output after inserting the data into **newly_acquired_stocks** table.

```
ps2_development=# SELECT * FROM newly_acquired_stocks;
 symbol | n_shares | date_acquired
--------+----------+---------------
 ibm    |      400 | 2008-08-28
 tesla  |      800 | 2020-04-22
 oracle |     2100 | 2022-10-26
(3 rows)
```

9 Solution:

To show the symbol, number of shares, price per share and the current value using INNER JOIN, the following command is used.

*select my_stocks.symbol as Symbol,my_stocks.n_shares as "Number of Shares",stock_prices.price as "Price Per Share", my_stocks.n_shares*stock_prices.price as "Current Value" from my_stocks INNER JOIN stock_prices ON my_stocks.symbol=stock_prices.symbol;*

Output after running the JOIN command:

```
ps2_development=# select my_stocks.symbol as Symbol,my_stocks.n_shares as "Number of Shares",stock_prices.price as "Price Per Share", my_s
tocks.n_shares*stock_prices.price as "Current Value" from my_stocks INNER JOIN stock_prices ON my_stocks.symbol=stock_prices.symbol;
 symbol | Number of Shares | Price Per Share | Current Value
--------+------------------+-----------------+---------------
 ibm    |              400 |          31.415 |         12566
 tesla  |              800 |          31.415 |         25132
 google |            15000 |          31.415 |        471225
 oracle |             2100 |          31.415 |       65971.5
 intel  |            10000 |          31.415 |        314150
(5 rows)
```

10 Solution:

Inserted another row into the **my_stocks** table

insert into my_stocks(symbol,n_shares,date_acquired) VALUES ('nasdaq','4000','04-23-2015');

Output after inserting the row into the table

```
ps2_development=# select * from my_stocks;
 symbol | n_shares | date_acquired
--------+----------+---------------
 ibm    |      400 | 2008-08-28
 tesla  |      800 | 2020-04-22
 google |    15000 | 2004-08-17
 oracle |     2100 | 2022-10-26
 intel  |    10000 | 2010-01-19
 nasdaq |     4000 | 2015-04-23
(6 rows)
```

To show the symbol, number of shares, price per share if available and the current value using OUTER JOIN, the following command is used.

*select my_stocks.symbol as Symbol,my_stocks.n_shares as "Number of Shares",stock_prices.price as "Price Per Share", my_stocks.n_shares*stock_prices.price as "Current Value" from my_stocks FULL OUTER JOIN stock_prices ON my_stocks.symbol=stock_prices.symbol;*

Output after running the Outer Join command:

```
ps2_development=# select my_stocks.symbol as Symbol,my_stocks.n_shares as "Number of Shares",stock_prices.price as "Price Per Share", my_s
tocks.n_shares*stock_prices.price as "Current Value" from my_stocks FULL OUTER JOIN stock_prices ON my_stocks.symbol=stock_prices.symbol;
 symbol | Number of Shares | Price Per Share | Current Value
--------+------------------+-----------------+---------------
 ibm    |              400 |          31.415 |         12566
 tesla  |              800 |          31.415 |         25132
 google |            15000 |          31.415 |        471225
 oracle |             2100 |          31.415 |       65971.5
 intel  |            10000 |          31.415 |        314150
 nasdaq |             4000 |                 |
(6 rows)
```

11 Solution:

Created a PL/SQL function that takes a trading symbol as argument and returns the stock value. This is done using the ASCII function which converts the characters into equivalent values.

```
create or replace FUNCTION ascii_value_count(symbol varchar(20))
returns int
LANGUAGE plpgsql
as
$$
declare
        total_count integer;
        temp_letter VARCHAR(2);
begin
        total_count := 0;
        for i IN 1..length(symbol)
        loop
                temp_letter := Substr(symbol, i, 1);
                total_count := total_count + ascii(temp_letter);
```

```
        end loop;
Return total_count;
end;
$$;
```

We have updated the stock_prices table to set each stock's value returned by the ascii_value_count function

```
UPDATE stock_prices SET price = ascii_value_count(symbol);
```

Output after updating the rows using the ascii_value_count function

```
ps2_development=# UPDATE stock_prices SET price = ascii_value_count(symbol);
UPDATE 5
ps2_development=# SELECT * FROM stock_prices;
 symbol | quote_date | price
--------+------------+-------
 ibm    | 2022-10-26 |   312
 tesla  | 2022-10-26 |   537
 google | 2022-10-26 |   637
 oracle | 2022-10-26 |   630
 intel  | 2022-10-26 |   540
(5 rows)
```

We have defined a function **portfolio_value()** which takes no arguments and returns the aggregate value of the portfolio.

```
create or replace FUNCTION portfolio_value()
returns SETOF text
LANGUAGE plpgsql
as
$$
declare
        answer text;
        share record;
        curs1 CURSOR FOR select my_stocks.symbol
symbol,my_stocks.n_shares shares,stock_prices.price price,
my_stocks.n_shares*stock_prices.price value from my_stocks INNER JOIN
stock_prices ON my_stocks.symbol=stock_prices.symbol;
```

```
begin
        OPEN curs1;
        loop
                fetch curs1 into share;
                exit when not found;
                answer:= share.symbol || ',          ' || share.shares || ',
' || share.price || ',                ' || share.value;
                return NEXT answer;
        end loop;
        close curs1;
return;
end;
$$;
```

Output after executing the portfolio_value function

```
ps2_development=# SELECT portfolio_value();
                    portfolio_value
------------------------------------------------------------
 ibm,            400,            312,            124800
 tesla,          800,            537,            429600
 google,         15000,          637,            9555000
 oracle,         2100,           630,            1323000
 intel,          10000,          540,            5400000
(5 rows)
```

12 Solution:

Query to find out the average price of our holdings

```
SELECT AVG(price) FROM stock_prices;
```

Output after executing the query

```
ps2_development=# SELECT AVG(price) FROM stock_prices;
   avg
-------
 531.2
(1 row)
```

Using the Insert statement, we have doubled our holdings in all the stocks whose price is higher than average i.e, 531.2

*INSERT INTO my_stocks SELECT my_stocks.symbol,n_shares*2 as n_shares,my_stocks.date_acquired from my_stocks INNER JOIN stock_prices ON my_stocks.symbol = stock_prices.symbol WHERE stock_prices.price > 531.2;*

Output after executing the query

```
ps2_development=# INSERT INTO my_stocks SELECT my_stocks.symbol,n_shares*2 as n_shares,my_stocks.date_acquired from my_stocks INNER JOIN s
tock_prices ON my_stocks.symbol = stock_prices.symbol WHERE stock_prices.price > 531.2;
INSERT 0 4
ps2_development=# SELECT * FROM my_stocks;
 symbol | n_shares | date_acquired
--------+----------+--------------
 ibm    |      400 | 2008-08-28
 google |    15000 | 2004-08-17
 tesla  |      800 | 2020-04-22
 nasdaq |     4000 | 2015-04-23
 intel  |    10000 | 2010-01-19
 oracle |     2100 | 2022-10-26
 google |    30000 | 2004-08-17
 tesla  |     1600 | 2020-04-22
 intel  |    20000 | 2010-01-19
 oracle |     4200 | 2022-10-26
(10 rows)

ps2_development=# SELECT * FROM stock_prices;
 symbol | quote_date | price
--------+------------+-------
 ibm    | 2022-10-26 |   312
 tesla  | 2022-10-26 |   537
 google | 2022-10-26 |   637
 oracle | 2022-10-26 |   630
 intel  | 2022-10-26 |   540
(5 rows)
```

After running the query in the exercise 10, we get the following

```
ps2_development=# select my_stocks.symbol as Symbol,my_stocks.n_shares as "Number of Shares",stock_prices.price as "Price Per Share", my_s
tocks.n_shares*stock_prices.price as "Current Value" from my_stocks FULL OUTER JOIN stock_prices ON my_stocks.symbol=stock_prices.symbol;
 symbol | Number of Shares | Price Per Share | Current Value
--------+------------------+-----------------+---------------
 ibm    |              400 |             312 |        124800
 google |            15000 |             637 |       9555000
 tesla  |              800 |             537 |        429600
 nasdaq |             4000 |                 |
 intel  |            10000 |             540 |       5400000
 oracle |             2100 |             630 |       1323000
 google |            30000 |             637 |      19110000
 tesla  |             1600 |             537 |        859200
 intel  |            20000 |             540 |      10800000
 oracle |             4200 |             630 |       2646000
(10 rows)
```

Query to generate a report of symbols and total number of shares held
SELECT symbol,n_shares from my_stocks group by symbol, n_shares;

Output after executing the query

```
ps2_development=# SELECT symbol,n_shares from my_stocks group by symbol, n_shares;
 symbol | n_shares
--------+----------
 google |    15000
 oracle |     4200
 nasdaq |     4000
 ibm    |      400
 google |    30000
 intel  |    20000
 oracle |     2100
 intel  |    10000
 tesla  |     1600
 tesla  |      800
(10 rows)
```

Query to generate a report which contains the symbol and its corresponding total value using join and group by

*SELECT my_stocks.symbol,my_stocks.n_shares * stock_prices.price as "Total Value" from my_stocks INNER JOIN stock_prices ON my_stocks.symbol = stock_prices.symbol GROUP BY my_stocks.symbol, my_stocks.n_shares, stock_prices.price*;

Output after executing the query

```
ps2_development=# SELECT my_stocks.symbol,my_stocks.n_shares * stock_prices.price as "Total Value" from my_stocks INNER JOIN stock_prices
ON my_stocks.symbol = stock_prices.symbol GROUP BY my_stocks.symbol, my_stocks.n_shares, stock_prices.price;
 symbol | Total Value
--------+-------------
 intel  |     5400000
 tesla  |      429600
 google |    19110000
 google |     9555000
 oracle |     2646000
 tesla  |      859200
 ibm    |      124800
 oracle |     1323000
 intel  |    10800000
(9 rows)
```

Query to generate a report which contains the symbol, number of shares and its corresponding total value while having atleast two blocks of shares using group by and having

*SELECT my_stocks.symbol,my_stocks.n_shares,my_stocks.n_shares * stock_prices.price as "Total Value" FROM my_stocks INNER JOIN stock_prices ON my_stocks.symbol = stock_prices.symbol GROUP BY my_stocks.symbol, my_stocks.n_shares,stock_prices.price HAVING COUNT(my_stocks.symbol) > 1;*

Output after executing the query

```
ps2_development=# SELECT my_stocks.symbol,my_stocks.n_shares,my_stocks.n_shares * stock_prices.price as "Total Value" FROM my_stocks INNER
 JOIN stock_prices ON my_stocks.symbol = stock_prices.symbol GROUP BY my_stocks.symbol, my_stocks.n_shares,stock_prices.price HAVING COUNT
(my_stocks.symbol) >= 1;
 symbol | n_shares | Total Value
--------+----------+-------------
 intel  |    10000 |     5400000
 tesla  |      800 |      429600
 google |    30000 |    19110000
 google |    15000 |     9555000
 oracle |     4200 |     2646000
 tesla  |     1600 |      859200
 ibm    |      400 |      124800
 oracle |     2100 |     1323000
 intel  |    20000 |    10800000
(9 rows)
```

13 Solution:

Created a view called stocks_i_like and encapsulated the above query

*CREATE VIEW stocks_i_like AS SELECT my_stocks.symbol,my_stocks.n_shares,my_stocks.n_shares * stock_prices.price as "Total Value" FROM my_stocks INNER JOIN stock_prices ON my_stocks.symbol = stock_prices.symbol GROUP BY my_stocks.symbol, my_stocks.n_shares,stock_prices.price HAVING COUNT(my_stocks.symbol) > 1;*

Output after executing the query

```
ps2_development=# CREATE VIEW stocks_i_like AS SELECT my_stocks.symbol,my_stocks.n_shares,my_stocks.n_shares * stock_prices.price as "Tota
l Value" FROM my_stocks INNER JOIN stock_prices ON my_stocks.symbol = stock_prices.symbol GROUP BY my_stocks.symbol, my_stocks.n_shares,st
ock_prices.price HAVING COUNT(my_stocks.symbol) >= 1;
CREATE VIEW
ps2_development=# SELECT * FROM stocks_i_like;
 symbol | n_shares | Total Value
--------+----------+-------------
 intel  |    10000 |     5400000
 tesla  |      800 |      429600
 google |    30000 |    19110000
 google |    15000 |     9555000
 oracle |     4200 |     2646000
 tesla  |     1600 |      859200
 ibm    |      400 |      124800
 oracle |     2100 |     1323000
 intel  |    20000 |    10800000
(9 rows)
```