

STEVENS INSTITUTE OF TECHNOLOGY
FE 513 - Database Design
Instructor: Cheng Lu

Final Exam

Student: Aidana Bekboeva

Contents

1	SQL	2
1.1	Task 1	2
1.2	Task 2	2
1.3	Task 3	2
1.4	Task 4	3
1.5	Task 5	3
1.6	Task 6	4
1.7	Task 7	5
1.8	Task 8	5
1.9	Task 9	5
2	User-defined Function in R	6
2.1	Task 1	6
2.2	Task 2	6
2.3	Task 3	6
2.4	Task 4	6
2.5	Task 5	7
2.6	Task 6	8
3	PostgreSQL API in R	9
3.1	Task 1	9
3.2	Task 2	9
3.3	Task 3	9
3.4	Task 4	9

1 SQL

1.1 Task 1

Import given bank data into PostgreSQL database.

```
1 --- Creating table bank_data and filling it with data
2 drop table bank_data
3 CREATE TABLE IF NOT EXISTS public . bank_data
4 (
5 id integer ,
6 date date ,
7 asset integer ,
8 liability integer ,
9 idx integer
10 )
11
12 COPY bank_data (id, date, asset, liability, idx)
13 FROM 'C:\Users\Public\bank_data-1.csv'
14 DELIMITER ','
15 CSV HEADER
16
17 SELECT * FROM bank_data
```

1.2 Task 2

Create a primary key for the import table.

```
1 --- Create a primary key for the import table
2 CREATE INDEX key_id ON bank_data(id);
```

1.3 Task 3

Find the highest asset observation for each bank.(i.e. There are 4 observations for each bank in a year. If there're 100 unique banks in total, then the result contains 100 observations. Each observation may belong to different quarters.) Sort the resulting table according to asset value. Report the first 10 observations of output table


```
1 SELECT MAX (asset) obs FROM bank_data
2 GROUP BY id
3 ORDER BY obs
4 LIMIT 10;
```

Data output		Messages	Notifications
	obs integer		
1	571		
2	1190		
3	1749		
4	2261		
5	2387		
6	2539		
7	2663		
8	2694		
9	2915		
10	2974		

1.4 Task 4

Show the query plan for question 1.3 using EXPLAIN tool.

```
1 EXPLAIN ANALYZE
2 SELECT MAX (asset) obs FROM bank_data
3 GROUP BY id
4 ORDER BY obs
5 LIMIT 10;
```

Data output	Messages	Notifications
		
QUERY PLAN		
text		
1	Limit (cost=1...	
2	-> Sort (cost=...	
3	Sort Key: (ma...	
4	Sort Method: ...	
5	-> HashAggre...	
6	Group Key: id	
7	Batches: 1 M...	
8	-> Seq Scan o...	
9	Planning Tim...	
10	Execution Ti...	

```
1 1. Limit (cost=1111.80..1111.83 rows=10 width=8) (actual time=17.809..17.813 rows=10 loops
   =1)"
2 2. -> Sort (cost=1111.80..1135.81 rows=9602 width=8) (actual time=17.808..17.809 rows=10
   loops=1)"
3 3. Sort Key: (max(asset))"
4 4. Sort Method: top-N heapsort Memory: 25kB"
5 5. -> HashAggregate (cost=808.29..904.31 rows=9602 width=8) (actual time=13.266..16.452
   rows=9614 loops=1)"
6 6. Group Key: id"
7 7. Batches: 1 Memory Usage: 1169kB"
8 8. -> Seq Scan on bank_data (cost=0.00..619.19 rows=37819 width=8) (actual time
   =0.012..2.569 rows=37819 loops=1)"
9 9. "Planning Time: 0.086 ms"
10 10. "Execution Time: 17.868 ms"
```

Listing 1: SQL output

1.5 Task 5

Given the highest asset table from question 1.3, count how many observations are there for each quarter.

In this task, we create a temporary table "tempor" to store observations for each quarter. By using the command SELECT COUNT, we can obtain the exact number of such observations

```
1 DROP TABLE tempor
2
3 CREATE TABLE tempor AS (
4 SELECT bank_data.id , bank_data.date , bank_data.asset ,
5        bank_data.liability , bank_data.idx
6 FROM bank_data
7 INNER JOIN (SELECT MAX (asset) obs, id FROM bank_data qdata
8             GROUP BY id) qdata ON qdata.obs=bank_data.asset
9 AND bank_data.id=qdata.id
10 WHERE bank_data.date = '3/31/2002'
```

```

11 ORDER BY bank_data.id);
12
13 SELECT COUNT(tempor.id) FROM tempor

```

First quarter: 1203 observations.

```

1 DROP TABLE tempor
2
3 CREATE TABLE tempor AS (
4 SELECT bank_data.id , bank_data.date, bank_data.asset ,
5        bank_data.liability, bank_data.idx
6 FROM bank_data
7      INNER JOIN (SELECT MAX (asset) obs, id FROM bank_data qdata
8                   GROUP BY id) qdata ON qdata.obs=bank_data.asset
9                   AND bank_data.id=qdata.id
10 WHERE bank_data.date = '6/30/2002'
11 ORDER BY bank_data.id);
12
13 SELECT COUNT(tempor.id) FROM tempor

```

Second quarter: 763 observations.

```

1 DROP TABLE tempor
2
3 CREATE TABLE tempor AS (
4 SELECT bank_data.id , bank_data.date, bank_data.asset ,
5        bank_data.liability, bank_data.idx
6 FROM bank_data
7      INNER JOIN (SELECT MAX (asset) obs, id FROM bank_data qdata
8                   GROUP BY id) qdata ON qdata.obs=bank_data.asset
9                   AND bank_data.id=qdata.id
10 WHERE bank_data.date = '9/30/2002'
11 ORDER BY bank_data.id);
12
13 SELECT COUNT(tempor.id) FROM tempor

```

Third quarter: 1747 observations.

```

1 DROP TABLE tempor
2
3 CREATE TABLE tempor AS (
4 SELECT bank_data.id , bank_data.date, bank_data.asset ,
5        bank_data.liability, bank_data.idx
6 FROM bank_data
7      INNER JOIN (SELECT MAX (asset) obs, id FROM bank_data qdata
8                   GROUP BY id) qdata ON qdata.obs=bank_data.asset
9                   AND bank_data.id=qdata.id
10 WHERE bank_data.date = '12/31/2002'
11 ORDER BY bank_data.id);
12
13 SELECT COUNT(tempor.id) FROM tempor

```

Fourth quarter: 5947 observations.

1.6 Task 6

For the whole sample data, how many observations have asset value higher than 100,000 and liability value smaller than 100,000.

```

1 SELECT COUNT (*) FROM bank_data
2 WHERE asset > 100000 AND liability < 100000;

```

Answer: 1411 observations.

1.7 Task 7

Each observation was given an 'idx' number. Find the average liability of observation with odd 'idx' number.

```
1 SELECT AVG (liability) FROM bank_data
2 WHERE ((idx % 2) <> 0) = false
3 GROUP BY ((idx % 2)) ;
```

Answer: 787029.208260616638.

1.8 Task 8

Find the average liability of observation with even 'idx' number. What's the difference between these two average numbers?

```
1 SELECT AVG (liability) even FROM bank_data
2 WHERE ((idx % 2) <> 0) = TRUE
3 GROUP BY ((idx % 2)) ;
```

Answer: 778839.890163934426.

Difference: 8189.318097.

1.9 Task 9

For each bank find all records with increased asset. The record with increase asset means one record's asset value is larger than the one of previous quarter. (For instance a bank (id: 123) has asset 30,000 in 3/31/02, asset 20,000 in 6/30/02 and asset 25,000 in 9/30/02. Then the record with bank id (123), asset value (25,000) and date (9/30/02) is recorded. Because its asset value is larger than asset value in 6/30/02.) Report the first 10 observation of output table.

For this task, we use LAG() to access previous rows at a specified offset value. Since our date values are already divided into quarters, we just simply use date as order for partition.

```
1 SELECT bank_data.*
2 FROM (SELECT bank_data.*,
3             LAG(asset) OVER (partition by id ORDER BY date) as prev_asset
4       FROM bank_data
5       ) bank_data
6 WHERE asset > prev_asset
7 LIMIT 10 ;
```

Data output							Messages	Notifications
	id integer	date date	asset integer	liability integer	idx integer	prev_asset integer		
1	9	2002-06-30	361953	332900	20913	348727		
2	9	2002-09-30	383246	352456	20914	361953		
3	14	2002-06-30	73600000	69200000	27335	68600000		
4	14	2002-12-31	79600000	74500000	27333	72800000		
5	28	2002-09-30	12474	5543	3939	12049		
6	35	2002-06-30	492046	457116	12624	471056		
7	35	2002-09-30	503401	467080	12625	492046		
8	39	2002-06-30	203754	182287	14697	201681		
9	39	2002-09-30	205211	182742	14698	203754		
10	39	2002-12-31	206140	183912	14695	205211		

2 User-defined Function in R

2.1 Task 1

Download daily stock data using a given stock ticker for a given time period (set starting time, ending time and stock ticker as input variable)

For this task we retrieve data using quantmod library and getSymbols.

```
1 library(quantmod)
2 library(tidyverse)
3
4 #1) Download daily stock data
5
6 stocks <- "NFLX"
7 getSymbols(stocks, from="2021-12-01", to = "2022-12-01")
```

2.2 Task 2

Get the adjusted close price and consider this price data only for following tasks.

```
1 #2) Get the adjusted close
2
3 adjPrice <- NFLX$NFLX.Adjusted
```

2.3 Task 3

Perform a rolling window estimation on stock price vector to calculate the mean and standard deviation. For each time, you keep using a fixed size of data to calculate the mean and the standard deviation. (e.g. The whole sample contains 5 data points. You can perform a rolling window estimation with a window size of 2. In the first window (subset), you calculate the mean and standard deviation (std) using first two data points. Then you will roll the window one step ahead and calculate the mean and std using the second and the third data points. You repeat the same procedure using the fixed size of data until the end of whole sample.) The window size needs to be set as a input variable.

For rolling windows estimation, we firstly (lines 3, 4) declare the window size and the quantity of windows (row pairs). Then, we use a for-loop with index i that iterates from 1 to n_windows and calculates mean and standard deviation for a chosen (at each window) pair of datapoints. We then store results for means and deviations into one dataframe.

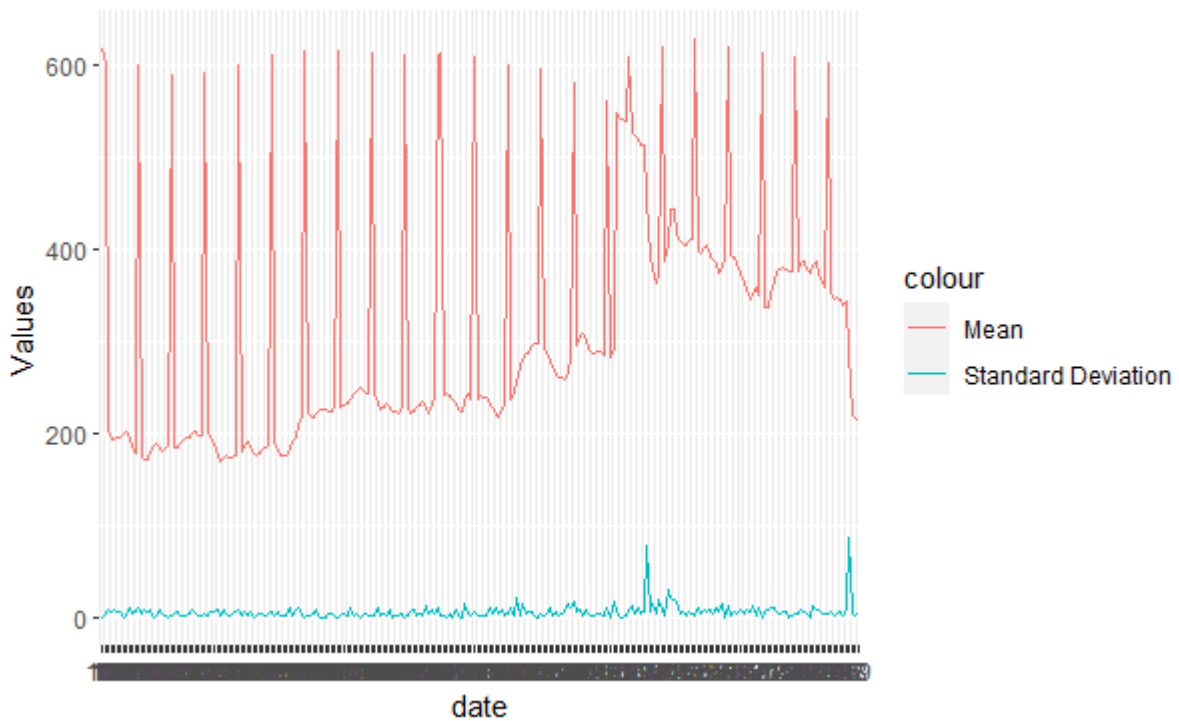
```
1 #3) Perform a rolling window estimation
2
3 size = 2
4 n_windows = nrow(adjPrice) - 2
5 results <- c()
6
7 for(i in 1:n_windows){
8
9   subs <- adjPrice[i:(i+1), ]
10
11   mean[i] <- mean(subs)
12   sdv[i] <- sd(subs)
13 }
14
15 results <- data.frame(adjPrice[2:252, 0], mean, sdv)
```

2.4 Task 4

Store the statistical result of Q 2.3 into a dataframe. Plot this statistical dataframe using scatter plot. In the plot, x axis represents the index for each rolling window and y axis represents the statistical values.

(Make sure you use different colors to differentiate between mean value and std value. Don't forget to include an legend.)

```
1 #4) Plot this statistical dataframe using scatter plot.
2
3 results <- data.frame(mean, sdv)
4
5 results <- setNames(cbind(rownames(results), results, row.names = NULL),
6                     c("Index", "Mean", "SD"))
7
8 library(ggplot2)
9
10 colors <- c("Mean" = "red", "Standard Deviation" = "blue")
11
12 ggplot() +
13   geom_line(aes(x=results$Index, y=results$Mean, group = 1, color = "Mean")) +
14   geom_line(aes(x=results$Index, y=results$SD, group = 2, color = "Standard Deviation")) +
15   ylab('Values')+xlab('date')
```



2.5 Task 5

Return the statistical dataframe.

```
1 summary(results)
```

```
1 > summary(results)
2   Index      Mean      SD
3 Length:251   Min.   :168.6   Min.   : 0.106
4 Class :character 1st Qu.:218.5 1st Qu.: 2.372
5 Mode  :character Median :256.6 Median : 4.448
6          Mean   :310.9 Mean   : 6.050
7          3rd Qu.:378.7 3rd Qu.: 7.598
8          Max.   :626.8 Max.   :86.564
```

Listing 2: R output

2.6 Task 6

Test your function with suitable parameters.(1 or 2-year data and a rolling window size of 20). If you got a huge dataframe for the output, do NOT print the whole sample. Showing a part of it is enough.

```
1 stocks <- "AAPL"
2 getSymbols(stocks, from="2020-12-01", to = "2022-12-01")
3
4 adjPrice <- AAPL$AAPL.Adjusted
5
6 size = 20
7 n_windows = nrow(adjPrice) - 2
8 results <- c()
9
10 for(i in 1:n_windows){
11
12     subs <- adjPrice[i:(i+1), ]
13
14     mean[i] <- mean(subs)
15     sdv[i] <- sd(subs)
16 }
17 results <- data.frame(mean, sdv)
18
19 results <- setNames(cbind(rownames(results), results, row.names = NULL),
20                     c("Index", "Mean", "SD"))
21 head(results)
22 tail(results)
```

Here, we used new data for Apple, and changed the window size to 20. As we can see, the function works well, the total quantity of observations was 502 (for 2 years). Below, see the first and the last 6 values of the resulting dataframe.

```
1 > head(results)
2   Index      Mean      SD
3 1      1 121.4422 0.25154334
4 2      2 121.5509 0.09782398
5 3      3 121.1408 0.48212237
6 4      4 121.5410 1.04808498
7 5      5 122.5933 0.44017751
8 6      6 121.6200 1.81666480
9 > tail(results)
10   Index      Mean      SD
11 497    497 149.650 2.3193088
12 498    498 149.095 1.5344203
13 499    499 150.625 0.6293349
14 500    500 149.590 2.0930403
15 501    501 146.165 2.7506454
16 502    502 142.695 2.1566778
17 >
```

Listing 3: R output

3 PostgreSQL API in R

3.1 Task 1

Make a connection to your local PostgreSQL database.

```
1 library(DBI)
2
3 con <- dbConnect(RPostgres::Postgres(), dbname = 'postgres',
4                 host = "localhost",
5                 port = 5432,
6                 user = 'postgres',
7                 password = '221928ABb')
```

3.2 Task 2

Query the PostgreSQL database via API to get the original bank data. (The bank data that you import to PostgreSQL database in Q 1.1) Store the data into a dataframe.

```
1 bank_data <- dbSendQuery(con,
2                          "SELECT id, date, asset, liability, idx FROM bank_data;")
3 bank_data <- as.data.frame(dbFetch(bank_data))
```

3.3 Task 3

Calculate asset growth rate for each quarter and each bank. (asset growth rate = (current quarter value - previous quarter value) / previous quarter value). The result start from second quarter, since we don't have all necessary data for first quarter calculation. Store the calculation result in a data frame.

```
1 asset_growth <- c()
2
3 for(i in 2:37819){
4   if (bank_data$id[i] == bank_data$id[i-1]) {
5     gr = (bank_data[i, 3] - bank_data[i-1, 3]) / bank_data[i-1, 3]
6     asset_growth <- append(asset_growth, gr)
7   } else asset_growth <- append(asset_growth, NA)
8 }
9 asset_growth
10
11 bank_data$asset_growth[2:37819] <- asset_growth
```

3.4 Task 4

Export the dataframe of Q 3.3 to the PostgreSQL database via API.

```
1 dbWriteTable(con, "bank_data_new", bank_data, overwrite = T)
```

Back to PostgreSQL:

```
1 SELECT * FROM bank_data_new
```

	id integer	date date	asset integer	liability integer	idx integer	asset_growth double precision
1	23373	2002-09-30	95914	87304	1	[null]
2	23376	2002-12-31	95937	87453	2	[null]
3	23376	2002-03-31	83335	75939	3	-0.13135703638846
4	23376	2002-06-30	84988	77125	4	0.01983560328793
5	23376	2002-09-30	90501	82248	5	0.06486798136207