CS 1555/2055 – DATABASE MANAGEMENT SYSTEMS (SPRING 2017)
DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF PITTSBURGH

Project: BetterFuture

Release: March 16, 2017

Due: Milestone 1 - March 31, 2017 @ 8:00pm
Milestone 2 - April 14, 2017 @8:00pm
Milestone 3 - April 22, 2017 @8:00pm

Project Goal

The goal of this project is to design and implement an electronic investing system of 401(k) retirement accounts. The core of such a system is a database system and a set ACID transactions. Our system, which we call "BetterFuture", allows personal investors to buy shares of mutual funds (of stocks, bonds, or mixed of both), exchange-traded shares, and keep track of their BetterFuture investments.

Milestone 1: The BetterFuture Database Schema

The BetterFuture Database includes information about the mutual funds, customers and their investments, the transaction histories and the current time and date (we do NOT use system time, rather, we maintain the "pseudo" current time in separate relation called MUTUALDATE). The latter will allow us to test different scenarios.

- Mutual funds
  The BetterFuture offers a variety of mutual funds such as money-market, real-estate, short-term-bonds, long-term-bonds, balance-bonds-stocks, social-responsibility-bonds-stocks, general-stocks, aggressive-stocks and international-markets-stocks. Each mutual fund has a name (e.g., money-market), is identified by its symbol and belongs to one or more categories: fixed, bonds, stocks and mixed. For example, money-market and real-estate are fixed, while balance-bonds-stocks and social-responsibility-bonds-stocks are mixed. The database also maintains a description for each fund and the date when the fund was created. In addition, the database keeps track of the closing price of each mutual fund which is used in the following day for purchases, exchanges-traded shares and calculation of investments.

- Customer data
  For every customer, we store his/her name, address, email address, a unique login-name and password. Each customer may own some funds and also has his/her allocation preferences based on which assets are allocated: Every time a customer deposits an amount to his/her 401(k) account, this amount is invested in shares based on the allocation preferences expressed as percentages. Customers can change their allocation preferences once a month. A customer can distribute his/her investment to as many funds as he/she wants. Each fund will have a corresponding percentage and the sum of the percentages should be 100%.

- Investment transactions
  Finally, the history of trading transactions is maintained in the TRXLOG table. Each transaction records the customer, the mutual fund involved in the transaction, the number of shares, the price of the mutual fund and the total amount. The date when the transaction was processed and the action taken are also stored.

- Relational schemas: The following schemas should be used.
  - MUTUALFUND(symbol, name, description, category, c_date)
    PK (symbol)
    Datatype
    * symbol: varchar2(20)
    * name: varchar2(30)
    * description: varchar2(100)
    * category: varchar2(10)
    * c_date: date
  - CLOSINGPRICE(symbol, price, p_date)
    PK (symbol, p_date)
    FK (symbol) $\rightarrow$ MUTUALFUND(symbol)
    Datatype
    * symbol: varchar2(20)
    * price: float
    * p_date: date
  - CUSTOMER(login, name, email, address, password, balance)
    PK (login)
    Datatype
    * login varchar2(10)
    * name varchar2(20)
    * email varchar2(30)
    * address varchar2(30)
    * password varchar2(10)
    * balance float
  - ADMINISTRATOR(login, name, email, address, password)
    PK (login)
    Datatype
    * login varchar2(10)
    * name varchar2(20)
    * email varchar2(30)
    * address varchar2(30)
    * password varchar2(10)
  - ALLOCATION(allocation_no, login, p_date)
    PK (allocation_no)
    FK (login) $\rightarrow$ CUSTOMER(login)
    Datatype
    * allocation_no: int
    * login: varchar2(10)
    * p_date: date

- PREFERS(allocation_no, symbol, percentage)
  PK (allocation_no, symbol)
  FK (allocation_no) → ALLOCATION(allocation_no)
  FK (symbol) → MUTUALFUND(symbol)
  Datatype
  - allocation_no: int
  - symbol: varchar2(20)
  - percentage: float

- TRXLOG(trans_id, login, symbol, t_date, action, num_shares, price, amount)
  PK (trans_id)
  FK (login) → CUSTOMER(login)
  FK (symbol) → MUTUALFUND(symbol)
  Datatype
  - trans_id: int
  - login varchar2(10)
  - symbol: varchar2(20)
  - t_date: date
  - action: varchar2(10) //see comment + below
  - num_shares: int
  - price: float
  - amount: float

  + action is either 'deposit' or 'sell' or 'buy'
  A deposit transaction should trigger a set of buy transactions showing that the money deposited into a customer's account will automatically be invested to mutual funds based on their preferences.

- OWNS(login, symbol, shares)
  PK (login, symbol)
  FK (login) → CUSTOMER(login)
  FK (symbol) → MUTUALFUND(symbol)
  Datatype
  - login varchar2(10)
  - symbol: varchar2(20)
  - shares: int

- MUTUALDATE (c_date)
  PK (c_date)
  Datatype
  - c_date: date

- You are required to define all of the structural and semantic integrity constraints and their modes of evaluation.

Milestone 2: the BetterFuture interface and Functions

For this project, you need to design two easy-to-use interfaces for two different user groups: administrators and customers (note that an administrator could not be a customer under our BetterFuture company rules). You are expected to implement your project in Java. You may design nice graphic interfaces, but it is not required and it carries *NO bonus points*. A simple text-oriented menu with different options is sufficient enough for this project: a two level menu such that the higher level menu provides options for administrators login and normal user login. The lower level menu provides different options for different users, depending on whether she is an administrator or he is a customer.

You can develop your project on either unixs, class3 or Windows environments, but we will only test your code in unixs or class3 machine. So even though your code works in a windows environment, you should make sure it works on unixs or class3 before you submit your project.

You need to design the **SQL transactions** appropriately and when necessary, use the concurrency control mechanisms supported by Oracle (e.g., isolation level, locking modes) to make sure that inconsistent state will not occur. There are two situations that you should handle:

- If the same user logs in and runs functions on two different terminals concurrently, the application semantics should remain still intact. For example, if the customer attempts to buy shares from two different terminals for a sum higher that his/her balance, one of the transactions should fail.
- If the administrator updates the share price of a mutual fund and/or updates today's date while a customer is buying/selling shares of the same mutual fund, the state of the database should stay consistent. For example, the administrator updates the per share price of stock XYZ from $10 to $15 while a customer is trying to buy 1 share for XYZ. If the customer's balance is $10, then the system cannot record the purchase of 1 shares at $15.

The objective of this project is to familiarize yourself with all the powerful features of SQL which include functions, procedures and triggers. Recall that triggers and stored procedures can be used to make your code more efficient besides enforcing integrity constraints.

Each interface is described as a set of functions. Many of which should be processed in an "all or nothing" fashion. The functions are described as following:

- Customer interface
  - Browsing mutual funds
    Customers are allowed to look at the list of all mutual funds or those in a category of their choice; they can also ask for funds to be sorted on the highest price for a given date, or alphabetically by fund name.
  - Searching mutual fund by text
    Customers can specify up to two keywords; our system has to return the products that contain ALL these keywords in their mutual fund description.
  - Investing
    Customers may submit (deposit) an amount for investment, which is used to buy shares based on their allocation preference. *Note that a 'deposit' transaction triggers a set of 'buy' transactions and this should be automated by a trigger defined in your database.* Other implementations will result in 0 points for this task. Also, Investing should either result in buying shares of all mutual funds as specified in the allocation or none. Buying some is not an option. Thus, when an amount is deposited and the new balance in the account is not sufficient to buy all the shares as specified in the allocation, that amount is just deposited in the account. In addition, any remaining amount after an investment becomes the new balance in the account.

- Selling shares
  A customer may sell a number of shares of a mutual fund by providing its symbol and the number of shares. The resulting amount from the sell remains in the customers account as a balance which can be used for buying shares of another mutual fund. *Note that the balance should be automatically updated by a trigger defined in your database.*
- Buying shares
  A customer may buy a number of shares of a mutual fund by providing its symbol and the number of shares or by providing its symbol and the amount to be used in the trade. In both cases, the required amount should not exceed the balance in the customer's account.

  Note, when a customer buys shares by specifying an amount which is equal or less the balance in his/her account. The customer buys the maximum number of (whole) shares based on the specified amount and the remaining amount remains in the balance. That this is not the same when you specify the number of shares to be bought. If the balance is not sufficient to buy all the shares, no share is bought. Also, the price of a share on a given day is the closing price of the most recent trading day, e.g., on a Monday will be the closing price of the previous Friday.
- Conditional investment
  There are two kinds of condition. First, a customer may set up a date, so that on that date, certain investment action (invest/buy/sell) will be executed. Second, a customer may set up price condition for buy and sell. For example, buy certain amount of mutual fund if its price drops under certain point or sell some if the price rises above certain point. The conditional investment should be realized through triggers.
- Changing the allocation preference
  A customer may change his/her allocation preference.
- Customer portfolio
  This function generates a performance report of a customer portfolio. For a specific date, the report will list the mutual funds the customer owns shares of: their symbols, their prices, the number of shares, their current_values on the specific date, the cost value and the yield as well as the total value of the portfolio. We define price to be the amount of money needed to buy one share of a stock. This price changes every day as discussed above. The current_value of a given stock owned by a customer is defined as the product of the number of shares and the price on the specific date. The cost value of a stock is the total amount paid to purchase the shares. The adjusted_cost is the cost value minus the sum of all the sales of the given stock. Thus, the yield is the current_value minus the adjusted_cost. We define the total value of the portfolio to be the sum of the current_value of the stocks owned by the customer.

- Administrator interface
  - New customer registration
    Each new customer has to provide his/her name, address, email address, preferred login name and password. You should also specify whether the new user is an administrator or not. Your database should prevent two customers from selecting the same login name.
  - Updating share quotes for a day
    Update the price of each share of a mutual fund.
  - Adding new mutual fund.
    Add a new mutual fund.
  - Updating the time and date.

- Statistics
  For the past x months, we want to know
  1. the top k highest volume categories (highest count of shares sold)
  2. the top k most investors (highest invested amount)

  k should be an input parameter that can be specified by the users of your system.

You must verify the login name and password of all users at the beginning. By default, we assume a pre-existing administrator with a login name "admin" and a password "root". This information can be inserted into your table after you create the database. You can add yourselves as the first customers.

Milestone 3: Bringing it all together

The primary task for this phase is to create a Java driver program to demonstrate the correctness of your BetterFuture backend by calling all of the above functions. It may prove quite handy to write this driver as you develop the functions as a way to test them. Your driver should also include a benchmark feature that stress the stability of your system buy calling each function automatically for multiple time with reasonably large amount of data and display corresponding results each time a function is being called.

Now this may not seem like a lot for a third milestone (especially since it is stated that you may want to do this work alongside Milestone 2). The reasoning for this is to allow you to focus on incorporating feedback from the TA regarding Milestones 1 and 2 into your project as part of Milestone 3. This will be the primary focus of Milestone 3.

How to proceed

Before implementing the required SQL queries within your application program, you should test them using sqlplus (after you populate the database with test data). Only after you are confident that you have the correct query should you start implementing it in JDBC.

The tasks have been ordered so that you continue building on previous tasks as you move along. Therefore, it is advisable to implement them in order, starting from administrative task #1.

All errors should be captured "gracefully" by your application. That is, for all tasks, you are expected to check for and properly react to any errors reported by Oracle (DBMS), providing appropriate success or failure feedback to the user. Also, your application must check the input data from the user. You need to create your own test data starting with the example data above.

Attention must be paid in defining transactions appropriately. Specifically, you need to design the **SQL transactions** appropriately and when necessary, use the concurrency control mechanisms supported by Oracle (e.g., isolation level, locking modes) to make sure that inconsistent state will not occur.

Finally, you can develop your project on either unixs, class3 or Windows environments (Lab PC or your laptop), but we will only test your code in Unix. So even though your code works in Windows environment, you should make sure that it also works on unixs or class3 and does not uses any specialized libraries before you submit your project. **It is your responsibility to submit code that works**.

Project submission

Milestone 1 of the project is due **8:00pm, Friday, March 31, 2017**. The first milestone should contain only the SQL part of the project. Specifically, the scripts to create the database schema along with the definition of the triggers, and any stored procedures or functions that you designed. If you wish to receive more feedback, feel free to include any or all of your SQL queries in your repository.

Note that after the First Phase submission, you should continue working on your project without waiting for our feedback. Furthermore, you should feel free to correct and enhance your SQL part with new views, functions, procedures etc.

Milestone 2 of the project is due **8:00pm, Friday, April 14, 2017**. The second milestone should contain, in addition to the SQL part, the Java code.

Milestone 3 of the project is due **8:00pm, Saturday, April 22, 2017**. The third milestone should contain, in addition to the second milestone, the driver and benchmark. *NO late submission is allowed.*

When you submit your milestones you need to **email us your GitHub commit ID** together with the full web link to your repository each time you submit a milestone. For the project, each group can **only submit once per milestone per group**.

Group Rules

- You are asked to form groups of two. You need to notify the instructor and the TA by emailing group information to cs1555-staff@cs.pitt.edu (Remember that you need to send the email from your pitt email address, otherwise the email will not go through). The email should include the names of the team members and should be CC-ed to both team members (otherwise the team will not be accepted).
- The deadline to declare teams is **Tuesday, March 21, 2017** . If no email was sent before this time, you will be assigned a team member by the instructor. Each group will be assigned a unique number which will be sent by email upon receiving the notification email.
- It is expected that both members of a group will be involved in all aspects of the project development. Division of labor to data engineering (db component) and software engineering (java component) is not acceptable since each member of the group will be evaluated on both components.
- The work in this assignment is to be done *independently* by each group. Discussions with other groups on the assignment should be limited to understanding the statement of the problem. Cheating in any way, including giving your work to someone else will result in an F for the course and a report to the appropriate University authority.

Grading

The project code (both Java source code and SQL code) will be graded on correctness (which includes integrity constraint specification), robustness (error-checking i.e., it produces user-friendly and meaningful error messages) and readability. You will not be graded on efficient code with respect to speed although bad programming will certainly lead to incorrect programs.

Enjoy your class project!