**INF1343, Winter 2011, Database Modeling and Database Design**

# The Final Project

One deliverable on March 7, another on March 28 (see below for details).

This assignment is worth 300 points, 30% of your course grade. Of those 300 points:

- 50 points are for the initial design, due on March 7, at the beginning of class.
- 250 points are for the final report, due on March 28, at the beginning of class.

The assignment can be done individually or in groups of 2 or 3. You can choose your own group. However, no group can include students who worked together on Assignment 2. If you are having trouble finding a group then contact me ***immediately***.

# The Problem

For this assignment you will build a prototype of an information system for a small online bank. Your bank will offer its customers a choice of deposit accounts, some of which will pay interest and charge fees. Your customers would be able to deposits and withdraw money using simulated ABMs and also to transfer funds between the accounts, including accounts in other banks. They will have a web interface that will allow them to see their balances and the recent activity on their accounts, and also to perform transactions.

Your bank will have a special partnership with a larger bank, the Imperial Bank of Spadina (IBS). IBS will handle all of your interactions with the external world.

Below are some of the specific features that you should offer, together with a number of out-of-scope items. (Out-of-scope means you specifically do *not* need to do it.) You should ideally implement all of those features. If you cannot, then do what you can. Please remember, however, that this is a course on *data modeling and database design*. You will therefore be primarily evaluated on the underlying database. A team that designs and builds an excellent database but fails to implement a working website will likely get a higher grade than a team that provides a beautiful web interface to a broken database. Also, your website can be ugly. Your SQL should be beautiful.

As you design the system, avoid making up features that are not asked for. (Or, at least, make sure you implement the required features before you start adding extra ones.) When in doubt, ask if a particular feature is expected. Your design should probably have around 7 entities.

All web components should be implemented using Python CGI, which will be introduced in class on Week 7. If your team would prefer to use a different approach to web development, *please contact me to discuss this*. Be advised that permission to use a different method will only be granted in cases where *all* members of the team are familiar with it. ***Your web interface does not need to handle authentication.***

**Out of scope items (overall)**

- Your bank will not allow overdrafts. Transactions that would result in a negative balance can simply be denied.
- You can pay interest at the end of each day and credit the accounts immediately. (What banks usually do, I believe, is assess interest frequently – perhaps every hour – and then credit your account at the end of the month. That's too complicated for our purposes.)
- Equally, you can make charge all of the account fees *daily* rather than monthly.

- All transactions will be in Canadian dollars only.
- There does not need to be an administrative interface apart from the specific things described below.

## 1. Online Banking

The customer should be offered online access to their account. They should be able use the website to see their balances and past transactions. They should also be able to transfer money between their accounts, to other customers accounts in your bank, or to an account at another bank.

Some specific things to consider:

- Your online banking site should be safe from SQL injection attacks by the customer. (We will cover this in week 10.)

**Out of scope items**

- ***Your online banking prototype does not need to handle customer authentication.*** It should work as if a particular customer has already been successfully logged in.
    - This means that your website will only work for *one* customer, Obi-Wan Kenobi. Obi-Wan should have accounts numbered 1001 and 1002. At least one of those accounts should be paying interest and should charge a fee if the balance goes below some threshold.
    - Your *database*, however, should handle multiple customers and you should enter a few of them into the database. In particular, Obi-Wan should be able to transfer money to other customer's accounts.
- You website does not have to be pretty. It should be functional, though.

## 2. Simulated Automatic Banking Machines (ABMs)

You should build a separate web page to simulate ABMs. You should have multiple AMBs, but you don't need many (2-3 is fine). Obi-Wan should be able to select an ABM, select the account and choose to either deposit cash into that account or withdraw cash from it. The simulated ABM will then either do the transaction or will tell Obi-Wan why the transaction could not be completed. Possible reasons for failure should include Obi-Wan exceeding his daily withdrawal limit, the account being out of funds, or the machine being out of cash.

**Out of scope items**

- As with online banking, the simulated ABMs should assume that the customer's identity is known. Don't worry about PINs, etc.
    - It should be the same customer who is authorized to use the online banking — Obi-Wan. If I go and withdraw $100 from one of Obi-Wan's accounts using the simulated ABM, this withdrawal should the appear when I go to online banking site.
- You machines will only deal with $20 bills. Assume that your ABMs can store any amount of cash.
- Your ABMs never have mechanical problems or make mistakes.

## 3. The Executive Dashboard

You should create another page that would show the bank's executive what is happening at the high level, including:

- The total amount of money you owe to customers, by account type.

- The total amount of cash your bank has.
- Your balance at the Imperial Bank of Spadina (IBS).
- Your overall balance (cash + money you are owed – money you owe).
- Your profit for the last day.

## 4. The Passage of Time

Customers need to be paid interest and charged fees. You will simulate this by creating a page that can be used to advance time. You can think of it as God's interface to your system. The page should show the "current date" and have a button saying "next day." When the button is clicked, the date should advance and all the necessary nightly events should occur, including:

- Interest should be paid and fees should be charged, where applicable. (This should be done to all accounts, not just Obi-Wan's!)
- The ABMs should be replenished with cash. If you need additional cash, you will be getting it from the Imperial Bank of Spadina, which will debit your account for the cash they give you. Make sure to keep track of the cash you receive from them – IBS is known to make mistakes in their accounting. If some ABMs have too much cash at the end of the day then the excess cash should be transferred either to other ABMs or to IBS.
- IBS will pay you 0.03% daily interest on your balance with them. (That is, they will give you $3 for every $10,000 of your balance.) If you balance is negative, they'll charge you 0.03% interest.

## 5. Inter-Bank Transfers

Your bank should be able to handle sending money to other banks or accepting payments from those bank. For simplicity we will assume that all payments are cleared through the Imperial Bank of Spadina (IBS), so you won't need to deal with other banks directly.

For incoming payments IBS will send you an XML file at the end of each day. The file will state how much money needs to be deposited into each customer's account and who it is coming from. It will also state the total amount of money that will be credited to account at IBS. (E.g., if you accept two $100 deposits then $200 will be credited to your IBS account.) Your system should have some way of handling this XML file.

For outgoing payments you will generate an XML file that tells IBS how much money to send to accounts in other banks and the total value that you expect to be withdrawn from your IBS account. Do remember that IBS is known for making mistakes. You should keep enough records to rectify things later.

I will provide you with sample IBS XML files. For now, just assume that they will include the following information about each transaction: (1) the account number in your bank, (2) the amount to be transferred, (3) the bank number of the counterpart bank, (4) the account number in the counterpart bank, (5) the name of the payer, (6) an optional comment.

**Out of scope items**

- Your customers cannot ask you to withdraw money from another bank. They can only *send* money. Equally, you will not need to deal with external requests to transfer money out of your customer's accounts.
- You do not need to worry about how the XML files will actually be transferred from IBS to your bank or vice versa.

# Deliverable 1: The Preliminary Design (50 points)

Your first deliverable is the preliminary design. It should be submitted on paper and should include:

1. **The list of team members.** Include the email address of each team member.
2. **An ER diagram and data dictionary.** The data dictionary should consist of a brief explanation of each entity and a table documenting and explaining the attributes of each entity. Explain any assumptions that you make and any non-obvious design choices.
3. **The implementation plan.** Describe the schedule for implementing and testing of the system. State clearly who is responsible for each piece and by when they will need to finish it.

# Deliverable 2: The Final Report and the Implementation (250 points)

Your final deliverable is the final report and the actual implementation.

The implementation should include:

1. **The database.**
2. **A web interface.** The web interface should simulate online banking, ABMs, the dashboard, and and the passage of time interface.
3. **Handling XML files.** Some way of creating the XML files for outgoing transfers and some way of handling XML files for incoming transfers.

The final report should include the following elements:

1. **The list of team members.** If this list is different from the one in your proposal, attach a note explaining why. (If you expect a change in team composition, please discuss this with the instructor.)
2. **The name of the database.** (I need to be able to locate your database.)
3. **URLs of the implemented system.** State the URLs for each of the required web components.
4. **Instructions for inter-bank transfers.** Provide instructions on how to test the inter-bank transfers. In particular, I will generate my own file that would request deposits into accounts 1001 and 1002. Your instructions should make it clear what I would need to do to import those XML files. Similarly, I should be able to use the website to request transfers to accounts in other banks. You should also provide clear instructions on how to generate an XML file for the requested outgoing transfers.
5. **The final ER diagram and data dictionary.** Include an ER diagram and data dictionary for your final system. *The diagram should match your implementation.* The final data dictionary should include and identify all necessary primary and foreign keys and should satisfy the conditions for 1NF, 2NF, and 3NF. Provide an explanation for any non-obvious decisions.
6. **SQL.** Include in your report *all* SQL statements that you used to create the database (CREATE and, if applicable, ALTER TABLE) and all the SQL queries used by your system. The latter should include all the SELECT, UPDATE, and INSERT queries, including those used inside Python files. Explain any non-obvious decisions.
7. **Additional Comments.** You can optionally provide additional explanations. For example, if you did not manage to implement some of the features, explain what you tried and why you think it didn't work.
8. **Statement of Responsibilities.** Describe briefly the contribution of each team member.
9. **Credits.** The ER diagram and the SQL used in this assignment should be written exclusively by your team. For other components (e.g., images, CSS, HTML, Python functions) you can use what's available on the web, if you wish. If you do so, state what you got where, including specific URLs. You can only use components that are publicly available and they cannot be custom-made for your project. (In other words, if you find some CSS you want to use, go for it. But do not *ask* someone to write CSS for you.)