Project 4, Microsoft Access Capstone

CS 0131, Software for Personal Computing
Timothy J Parenti

Assigned: Monday 15 April 2013 **Due:** Sunday 21 April 2013, 11:59pm EDT

Introduction

This is the fourth and final project of the term, covering the features of **Microsoft Access 2010**. You only need to make one database file for this project, but there are several components corresponding to the various types of objects in Access, each beginning with an 'S':

1. **Structuring the Database** — Designing tables to hold customer and account information.
2. **Simplifying the Tellers' Job** — Creating a simple query to look up accounts, as well as a form to modify account data.
3. **Seeming Productive with Reports** — Running reports to summarize the data and glean useful information from it.

As we discussed in class, databases are very useful for storing large amounts of data. On a day-to-day basis, you probably interact (indirectly) with many databases without even realizing it! In fact, your favorite shopping site, search engine, and social network are all probably running giant *distributed* databases that are spread across hundreds or thousands of servers. When you think about the grand scale on which many of these sites operate, it gives you an appreciation for how much work goes into keeping everything running... and perhaps puts things in perspective when, occasionally, they don't!

Another set of databases you probably rely on regularly are the databases your bank keeps to make sure all of your transactions are processed in an efficient (and just as importantly, correct) manner. A database is the perfect thing to help banks manage their customers and their customers' accounts. Today, you'll be placed into the shoes of the employee who has been tasked with creating the humble beginnings of the database of accounts at everyone's favorite bank, Banko Bank.

Submission instructions are at the end of this document. All submissions are due **Sunday 21 April 2013, 11:59pm EDT**.

1 Structuring the Database

Let's start by creating a blank database called `banko.accdb`. Save it in a known location so you won't lose track of the database file.

The first thing we need for a database is tables! A big bank like Banko Bank will ultimately end up having lots of tables for their employees, branches, fee structures, ATM transactions, and so on... but hey, it's your first day on the job, and you just want to impress your ~~instructor~~ **boss** with your knowledge of Microsoft Access 2010 so that he'll keep trusting you with important stuff. You decide that being able to manage some basic information about customers and their accounts would be a good start.

Let's pretend that Banko Bank offers two types of accounts: checking accounts and savings accounts. Each customer can have an unlimited amount of accounts, although most people will probably only have one of each type.

1. The bank needs to keep track of several bits of required customer information. While you're designing the table, you ask yourself some questions to make sure you're designing the best possible database you can to keep Banko Bank going strong for years to come. For each customer, you'll need some way of storing:

   a. Customer ID number — This can be automatically generated by the database system.
   b. Customer name — Does it make sense to put this in one field or two?
   c. Customer address — What if we wanted to offer a special to customers in a certain state or city?
   d. Customer social security number (SSN) — This is always nine digits long. How can you make sure it's never longer than nine digits? What if it starts with a zero?
   e. How long they've been a customer — So you can give preferred customers special benefits, including Banko brand pens and Banko brand stationary! — But how should we store this data?
   f. The customer's preferred check design — Some people like their checks to have pictures of kittens, others like beach pictures, still others prefer formal Banko checks. How can we make sure customers choose a valid check type that's available? (Hint: We might need another table for that.) Make the default value be for this field correspond to the Banko-branded checks.

   Finally, you consider the different data types that Access offers for fields (text, number, currency, autonumber, etc.) and decide which is best for each. You apply appropriate validation rules to the fields that require them and make certain fields required as makes sense.

2. Next, you shift your attention to the individual accounts. For each account, you need some way of storing:

   a. Account number — How can we automatically generate this so we don't have to worry about duplicate account numbers?
   b. Account type — We only offer checking and savings accounts; how can we make sure these are the only types used by our database?
   c. Account balance — What's the right data type for this?
   d. Interest rate — You should set the default value to be 0 if a non-zero rate isn't otherwise entered. Some accounts (like savings accounts) do have an interest rate that is non-zero; preferred customers may be given higher rates than usual. You should express rates as a decimal (e.g., for 2.25%, you would store 0.0225), and up to four decimal places should be allowed (you'll need to set Decimal Places *and* Scale).

   Again, you consider the different data types available and set validation rules to make sure, for example, that there isn't a negative interest rate on an account. You also make certain fields required.

3. It makes sense to store the customers and accounts in two separate (clearly named) tables. We can then use a third table to link them together. This third table will probably only have two fields in it, since it will only link customers to accounts. Think about which fields in the other tables you'll be linking, and specify the data types of your fields in this "join table" accordingly. Give it a name that clearly reflects its purpose.

4. Once you've designed the fields of your three tables (and the ancillary tables you decided you needed to store other information), you'll need to establish primary keys for each. Think about what must be uniquely identifiable in each of your tables. In particular, we'd like to be able to uniquely identify customers and accounts, but we might have more than one account associated with the same customer (checking and savings, perhaps), or more than one customer associated with the same account (in the case of spouses, perhaps). Think about what this means, in particular, for your join table that links accounts to customers. What should be the primary key(s) there? (Join tables like this are a very common database construct, and thus an important and useful concept to understand.)

5. Now it's time to link up your tables. In relationship view, you can simply drag a field in one table to the corresponding field in another table to create a relationship, then double-click on the join line to enforce referential integrity.

6. Finally, we want to know that our database design is going to work once it hits prime time, so we need some sample data. If you've set up the proper referential integrity constraints, it will probably make the most sense to proceed in this order:

    a. There are only a few check designs, and the database should not allow a customer or bank teller to input an invalid type. Specify at least three different check design types, each with an ID number, in the table that holds them. (One of these should be Banko-branded checks.) Make sure that table is capable of adding new designs when they become available.

    b. We only offer two types of accounts: checking and savings. Specify an ID number for each in the table that holds information on account types. Make sure that table is capable of adding new account types at a later date (once Banko Bank grows big enough to start offering money market accounts).

    c. Enter sample data for at least five customers and at least five accounts.

    d. Enter customer-account links into your join table. Every customer must linked to at least one account, and every account must be linked to at least one customer. To be really sure everything is going to work correctly, make at least one customer be linked with multiple accounts, and make at least one account be linked with multiple customers.

2 Simplifying the Tellers' Job

The bank tellers want to be more productive at work, and furthermore, they're tired of looking at table after table of cells of data. They want color and they only want to see what is relevant to what they're currently doing. Most of what they're doing is handling deposits and withdrawals.

To that end, the tellers have requested a workflow that asks for the account number and then it lets them edit the balance (so that they can handle the customer's transaction). Ideally, we'd be able to do entire transactions in a single form, but since we're just getting started, we'll be satisfied with a two-step process for now: a query followed by a form.

1. First, let's create a query that requests a given account number:

    a. Using the Simple Query Wizard, create a query that displays only the account number, balance, interest rate, and name of the customer (so the bank teller can verify the photo identification of the person conducting the transaction).

    b. Then, in Query Design view, set the criteria of the account number field so that when the query is run, a pop-up box appears asking the user for the desired account number. You can accomplish this by setting the "Criteria" for the account number field to a prompt enclosed in square brackets, e.g., "`[Account number?]`".

    c. Test our your query by running it with F5. When run, a message box should pop up with your prompt asking you for the account number. After entering that, it should display the information for that account **only**. Remember, each account may have multiple people associated with it, so you may get two records returned as a result, but each record should be for the same account number. Press F5 to re-run your query on different accounts to make sure it's working properly.

2. With the results of your account query open, select Create » Form to create a form for account editing using the fields from your query. Using the Layout and Design views, customize the form as follows:

    a. The form should feature in big red letters the words "Account Edit Mode" so that the bank teller is aware of the fact that this is a dangerous form and should be used carefully. After all, any error could result in a very unhappy customer (or a very unhappy bank).

b. Re-word your labels, if necessary, to make the form more user-friendly. For example, if you have a label that says "`account_id`" you should change it to something like "`Account Number:`". Do this for all the fields, using proper capitalization throughout.

c. Download the BANKO BANK logo from the "Projects" page of the course website and use it in the header of your form.

d. Lastly, take the text boxes which display each field from the query and rearrange them in some way. Some possible ideas include just reordering them, or displaying them side by side. Accomplish this using the tools on Form Layout Tools » Arrange and in the right-click context menu when editing the form. You should arrange the fields in a way that makes sense for the data that's in the form.

e. Test your form by editing a balance and then pressing ⇧Shift+Enter to save the associated query record. Open your table of accounts to see the change reflected in the underlying data.

3 Seeming Productive with Reports

As an employee of BANKO, you are expected to seem productive (seeming productive is not always the same as actually *being* productive). And what better way is there to seem productive than by generating large reports of seemingly useful data? You hope that by producing large stacks of paper filled with numbers, your boss will be so impressed that he'll consider you for a swift promotion.

1. Make a report that uses all of the information from both the customer and account tables. Be sure and sort the data based on some kind of criteria. For example, you might sort based on the customer names, the customer sign up date, or even balance. You'll also want the report to contain some summary information (e.g., total balances, minimum balances, etc.).

2. Don't forget to add the official BANKO BANK logo to your report. Make sure there is an automatically updating date somewhere on the report, and that some text like "`SENSITIVE DATA - DO NOT DISTRIBUTE`" appears in the page footer.

3. In Layout view, notice the Conditional Formatting tool under Report Layout Tools » Format, which is very similar to the Conditional Formatting tool in Excel. Cause the following to happen in the report:

   a. If an account balance is negative (i.e., $< \$0$) it should be red

   b. If an account is not negative, but is less than $100, it should be yellow

   c. If an interest rate is less than 1%, it should be underlined. This way, we can notice this account and try to convince the customer to put their money into a high-interest savings account.[1]

   d. If a customer's preferred check design is **not** the BANKO-branded design, it should be made bold. The BANKO tellers will try to convince these customers to use BANKO brand checks, which effectively advertise BANKO BANK wherever the checks are used.

4. Lastly, be sure to change any goofy looking field labels (e.g., "`account_interest`" into something more presentable, such as "`Interest Rate:`").

Whew! A busy first day of work at BANKO BANK. Save all your open objects, and make sure they all have descriptive (non-default) names, then close your database file. Hopefully the ~~boss~~ **instructor** is impressed!

Submission

Use the "Assignment Submission" section in CourseWeb to submit your `banko.accdb` file for **Project 4** by the deadline. All submissions are due **Sunday 21 April 2013, 11:59pm EDT**.

[1] Why would BANKO do this? In part, because their high-interest savings accounts have many restrictions and huge penalties if the rules aren't followed. Typically, this means BANKO makes lots of money on these accounts.