Homework #1

Complete By: Monday, June 22nd @ 11:59pm
Assignment: reading + completion of following exercise
Policy: Individual work only, late work *is* accepted
Submission: electronic via Blackboard

Reading

Read chapters 1-3 in the required text, "Database Design for Mere Mortals".

Software Setup

If you haven't already, you need to install Visual Studio 2013; the Community Edition is freely available, though you can also install the Professional or Ultimate editions if you have access to those. The Community Edition can be found via an internet search.

Visual Studio requires Windows. If you are not running a Windows-based machine, you have two options: (1) setup a Virtual Machine to run Windows, or (2) if you have a Mac, use Apple's BootCamp to dual-boot OS X and Windows. If you go with a VM, I'd recommend VirtualBox as your VM software (it's free and cross-platform); be sure to allocate at least 4GB of RAM (more if you have it) to the VM. If you go with BootCamp, allocate at least 60GB of disk space to the Windows partition (more if you have it). Copies of Windows 7 are available here for download:

32-bit: https://uofi.box.com/s/cv0v5rpvewax9etv05al
64-bit: https://uofi.box.com/s/ey00rglkgham3ac9bqge

You will need a Windows key to successfully activate Windows; email Sean (sdeitz2@uic.edu) for a unique, valid key.

The uiTunes Database

First, before you get started, make sure Visual Studio is updated to the latest version: Tools menu, Extensions and Updates, and then expand the "Updates" section. Look in "Product Updates" for updates to Visual Studio, and the Database tooling. Install those updates.

Now it's time to design your database for the fictitious "uiTunes" web site. Recall that we started this process in class on Thursday, 06-18; lecture notes and SQL scripts are available on the course web site (http://www.joehummel.net/cs480.html, see the Lectures folder), and a recording of the class is available on Blackboard (Tools, Echo360). You are free to design the database as you see fit, as long as it stores the following information:

Song: name, year released, duration, price, genre(s), artist(s), role of each artist (lead, "with", …)
Album: name, year released, price, songs(s), artist(s), role of each artist (lead, "with", …)
Artist: name, year born, song(s), album(s)

User: first name, last name, email, password, account balance (positive => credit, negative => owes)
Playlist: name, user, song(s) and album(s)
Review: song or album or playlist, user, rating (1..5), comment (optional)

Purchase: items (songs and albums), user, amount paid, date

Consider the above as "minimum" requirements; you can certainly store more data if you wish. In your design, you are required to follow the basic principles of relational database design, namely:

1. Store a value only once
2. Store a single value in a single field (e.g. no lists in a field)
3. Store related data together (one entity == one row)
4. Assign a unique key to every entity

For example, consider a user's music preference. We could store the genre(s) that a particular user likes, or we can compute his/her preference from their purchase history. Likewise for the genre(s) of an album or an artist --- we can store, or is it better to compute from the individual songs? And should we store the duration of an album, or compute from the individual songs?

After you design your database, think about any additional indexes you might need…

Feel free to talk to your partner(s) and classmates about their design, and incorporate their ideas as appropriate; a number of good suggestions have already been mentioned in class. But ultimately this is your design, and you need to write the SQL yourself. When you're ready, convert your design into an SQL script to create the necessary tables and indexes. Test your script by creating a "uiTunes.mdf" database using Visual Studio and Microsoft SQL Server; follow the steps discussed in class on Thursday, 06-18. If you need help, note that a sample SQL script file is available on the course web page: Lectures\06-18\uiTunes\uiTunes-Create.txt. This creates the 4-table database discussed in class.

To further test your database, write a SQL script to insert some records into your database. Some initial data and Insert queries are available on the course web page: Lectures\06-18\uiTunes\uiTunes-Insert.txt. You'll need to adapt as needed for your particular database design. Keep an eye on Piazza for posting of additional sample data you can use; feel free to post your own data as well.

When you are done and happy with your design, create an ER diagram of your database. Here are the steps in Visual Studio:

1. Make sure your database appears in the Server Explorer, under Data Connections
2. File >> New, select Project…
3. Under Templates, expand Visual C#, and select "Windows Desktop"
4. In the list to the right, select "Windows Forms Application"
5. Confirm the name and location at bottom of window, and click OK
6. Give Visual Studio a few seconds to create --- eventually a small "Form1" design surface appears

7. Project menu, select Add New Item…
8. Select Data in the left-most list, then to the right select "ADO.NET Entity Data Model"
9. Click Add
10. Select "EF Designer from Database", click Next
11. Select "uiTunes.mdf" from drop-down, click Next
12. Answer "Yes" to make a local copy…
13. Use Entity Framework 6.x, click Next
14. Click the checkbox for "Tables", and click Finish

15. In a few seconds Visual Studio will generate it's version of an ER diagram
16. You can drag the entities around to adjust diagram…
17. To export, right-click on diagram's background, Diagram >> Export as Image…
18. For submission purposes, export as .jpg

That's it!

Have a question? Use Piazza, not email

Use Piazza for questions: http://piazza.com/uic/summer2015/cs480/home. As discussed in the syllabus, questions via email will be ignored, so post to our Piazza site. Remember the guidelines for using Piazza:

1. *Look before you post* — the main advantage of Piazza is that common questions are already answered, so search for an existing answer before you post. Posts are categorized to help you search, e.g. "Pre-class" or "HW".
2. Post publicly — only post privately when asked to by the staff, or when it's absolutely necessary (e.g. the question is of a personal nature). Private posts defeat the purpose of piazza, which is answering questions to the benefit of everyone.
3. Ask pointed questions — do not post a big chunk of code and then ask "help, please fix this". Staff and other students are willing to help, but we aren't going to type in that chunk of code to find the error. You need to narrow down the problem, and ask a pointed question, e.g. "on the 3rd line I get this error, I don't understand what that means…".
4. Don't post your entire answer — you just gave away the answer to the ENTIRE CLASS. Sometimes you will need to post code when asking a question, but then post only the fragment that denotes your question, and omit whatever details you can. If we can't answer your question, we will ask you to post your entire code privately — i.e. "visible to staff only". This is an option when you post.

Electronic Submission

Using Blackboard (http://uic.blackboard.com/), submit 3 files: your SQL create script, your SQL insert script, and your ER diagram (.jpg preferred). There will be a separate Blackboard assignment link for each file. At the top of your script files, add a header comment with your name and other information. For example:

```
--
-- SQL script to create uiTunes database
--
-- NAME and NetID
-- U. of Illinois, Chicago
-- CS 480, Summer 2015
-- Homework 1
--
```

You may submit as many times as you want before the due date, but we grade the last version submitted. This implies that if you submit a version before the due date and then another after the due date, we will grade the version submitted after the due date — we will *not* grade both and then give you the better grade. We grade the last one submitted. In general, do not submit after the due date unless you had a non-working script before the due date.

In terms of grading, we will be focusing on (1) completeness, i.e. does your database store the required information, and (2) relational, i.e. does your database follow standard relational design principles.

Policy

Late work *is* accepted. You may submit as late as 24 hours after the deadline for a penalty of 25%. After 24 hours, no submissions will be accepted.

Unless stated otherwise, all work submitted for grading *must* be done individually. While I encourage you to talk to your peers and learn from them, you cannot submit their work — whether partial or complete — as your own. The University's policy is described here: http://www.uic.edu/depts/dos/docs/Student%20Disciplinary%20Policy.pdf . In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. It is also considered academic dishonesty if you click someone else's iClicker with the intent of answering for that student, whether for a quiz, exam, or class participation. Academic dishonesty is unacceptable, and penalties range from failure to expulsion from the university; cases may be submitted to the official student conduct process: http://www.uic.edu/depts/dos/studentconductprocess.shtml