Thomas O’Brien and Anthony Keba

CMSI 486

9/20/18

Homework Assignment #1

**1.13** One example would be some small application that runs locally, does not have multiple users hitting it, and does not expect to change the data at all. While databases can be used on any size scale, they’re generally favorable when dozens of rows are being generated and manipulated. Not only that, but when only one user is using a system, the necessity for a database is not nearly as strong. Another example would be an application used for compressing files. Files themselves are being manipulated here, there is no need to store them or their pieces into a database. Chances are the user would want their files to be simply compressed and nothing more. Creating a database for a case like this could be considered to be over engineering the application.

**1.14 a)** The following columns would need to be updated: STUDENT.Major, COURSE.Course\_number, COURSE.Department, SECTION.Course\_number, PREREQUISITE.Course\_number, and PREREQUISITE.Prerequisite\_number.

**1.14 b)** Unique course ids could be implemented into the various **Course\_number** and **Prerequisite\_number** columns. By using a unique course id, they will not have to be up to date with the department name. A unique course id will always refer to a specific course.

**2.3** The database schema is a description of a database that contains the information of what the title of a table, the various columns and their expected types, and so on. Generally, this is not expected to change and is determined during the initial design of a database. The database state is the data currently in a database during a given instance. This includes all of the information regarding the current data living within the database and the specific rows and columns it pertains to. The main difference between the two is that a database schema is more like the design of what kind of data you would expect from a given table and column, while the database state is the actual data itself stored in the rows and columns at a given instance.

**2.7** One type of user-friendly interface is a menu-based interface for web clients or browsing. This interface relies on dropdown menus to both present the data to the user and allow the user to be manipulate the data in the database. These tend to be pointed towards users who are unfamiliar with query languages. Another type is a form-based interface. This utilizes forms for users to fill out that can vary from data insertion to manipulating existing data. Similar to a user-friendly interface, a form-based interface is also catered towards naïve users with minimal programming experience. A graphical user interface displays a schema to any user in a diagrammatic form. In order to modify data in a GUI, the user would manipulate the diagram somehow. While these are designed for any kind of user to use, GUIs are used by a wide range of users. Another category, natural language interfaces, attempt to understand some sentence in a natural language into a query. These have not developed significantly overtime due to the difficultly in parsing out natural languages. The target audience for these were intended to be anyone with an understanding in a natural language. Another interface would be the speech input and output interface. These require the user to know the expected vocabulary that the interface is programmed to know and use these phrases to manipulated the database. These are designed for users who do not want to learn a querying language but have the ability to learn keywords. Another interface would be interface for parametric users. These are designed so that certain query functions are designed to be setup as hotkeys for the user to use. These are designed for users that perform repetitive tasks over and over again. Lastly, there are interfaces for the DBA. These are interfaces that have specific commands only available to staff users such as setting system parameters, granting account authorization, changing a schema, and so on. The kind of users here would be for organization that have multiple users without the same permissions.

**2.14** The best DBMS architecture for creating a web-based system to make airline reservations and sell airline tickets would be the three-tier architecture for web applications. Here, the client is a GUI that the user would interact with. When interacting with the GUI, the web interface is able to call upon functions that are from the web server that the GUI lives on. This web server is then able to interact with the database which contains all of the data regarding any airline reservation. The Centralized DBMS Architecture would not work due to users not being able to be on the same machine as the database at an industrial scale. The Basic Client/Server Architecture would work, but put a heavy load on the server itself. This is due to both the application logic and the GUI having to live on the same machine. This issue also applies to the Two-Tier Client/Server Architecture.

**3.2** When looking at the definition itself, a relation is a set of tuples. Mathematically, part of a set’s definition states that it will not contain an order of any kind. Thus, this means that a relation is not sensitive to the ordering of the tuples it contains.

**3.5** It’s important to designate one of the candidate keys of a relation as the primary key for various reasons. The main reason is to create a standard for deciding which candidate key to use consistently across other tables when targeting a specific row in a table. Another reason for picking a primary key is that it is normally the key will the least number of attributes, making it the easiest to identify when compared to the other candidate keys.

**3.9** A foreign key is a set of attributes in a relational schema of some kind that contains a reference towards the primary key of another relational schema. The foreign key itself must be equal to a primary key that exists in another relational schema or must be null. This is useful for cross referencing multiple relational schemas together and joining data from multiple tables to do some kind of data manipulation.

**3.13** There are several difference options for what candidate key to use for the relation CLASS. One option is simply the attribute **Univ\_Section#**, provided that this attribute is unique across multiple semester. The keys **Course#, Univ\_Section#, and Semester** would work as the candidate keys if the attribute Univ\_Section# is not unique across multiple semesters. Another set of candidate keys could be **Semester, Building\_code, Room#, Time\_period, and Weekdays**, provided that there is only one class per room**.**

**3.20 c)** The advantages of using generated keys is that it will guarantee the keys to be unique and will not having issues references the proper rows when trying to grab it. The disadvantages of using generated keys is that they must be generated, which takes some time. Not only that, but it is possible to run out of generated keys.