Marist College

Lab 2

(Part 2 - Essays)

Dayna Eidle
Database Management
Professor Alan Labouseur
Due Date: September 18, 2017

Keys

A **primary key** is a set of one or more columns used to define a specific row of a table. Each table can have only one primary key. The main goal of the primary key is to maintain the uniqueness of each table. An example of a primary key might be *user_ID* because that would be unique to other tables. A **candidate key** is the column (or set of columns) that make a row unique to other rows. Candidate keys can have multiple fields combined, but they use the least amount of keys to ensure uniqueness. For example, *firstname* could not be a candidate key on its own because there's a good chance that more than one person could have the same first name so it is not unique. However, if *lastname* was added to *firstname* and no one in the table had the same full name, then that would work as a candidate key because it makes each row unique to the other rows. However, if another key was added to *firstname* and *lastname*, like *age*, this would be a **superkey**. A superkey is any key that takes a candidate key and adds another existing attribute to it, even if it is not necessary. In the previous example, *age* is not needed to ensure uniqueness, but it also doesn't take away from the row being unique, so it classifies as a superkey. As long as uniqueness is achieved with the superkey, it can have as many fields as desired.

Data Types

A data type, when dealing with databases, is the type of value being put into each column for a table. These values can be integers, booleans, strings, etc. When I have meetings for lacrosse, we create a sign-in sheet which is a database of the people

who come to the meeting. On the sign-in sheet, the columns for demographics are first name, last name, CWID, email address, and year(in school). The columns for positions played are defense, attack, midfielder, and goalie. For first name, last name, email address, year in school, and all of the positions played, they would be string values. For positions played, we have them write yes or no underneath each position so that we know which positions each person can potentially play However, this could also be a boolean value and be true or false depending on if they can play each position or not. Finally, CWID would be an integer value because those have to be an eight digit number for each player from the school. For this specific example, none of the fields could be nullable because for the demographics, everyone has a first and last name, CWID, email address, and current grade at the school. Then, for the positions played, yes or no needs to be filled in so there is no need for it to be blank.

Relational Rules

1. **First Normal Form (1NF) Rule -** The First Normal Form rule states that every point in a table where the rows and columns intersect (each box in the table), the values inside must be broken down into the smallest bits of information. In other words, they must be in their atomic form. An example of this would be a table that had *name* as a field. If I were to enter my name into that box, it would be “Dayna Eidle”. However, this violates the 1NF rule. My name can be broken down further than just my full name. To satisfy the 1NF rule, the *name* field would have to be broken down into *firstname* and *lastname*. That way I have “Dayna” in

one box and “Eidle” in another. This rule is important because it organizes the table better and makes looking values up more efficient.

2. **Access Rows by Content Only Rule -** The Access Rows by Contents Only rule states that you should address the contents of a table by what is there, not where the content is in the table. For example, say there is a table of demographic information about people with the first column being *firstname*. If I wanted to see my information in the third row, I wouldn’t access it by addressing the row number; I would address it by saying “WHERE firstname = ‘Dayna’”. Accessing it by where it is in the table leaves room for error if the table is modified. If I addressed it by row number, and another name was added above mine, then suddenly, my name is not in the third row anymore and I will not get the correct information that I want.
3. **All Rows Must Be Unique Rule -** The All Rows Must Be Unique rule is just as it sounds - all rows in a table must achieve uniqueness, being that no two rows are exactly the same. If the columns of a table are *firstname*, *shirt size*, *favorite color*, and *favorite animal*, the first row could have the values “Brooke”, “medium”, “green”, “dolphin”. Then, the second row could have the values “Brooke”, “medium”, “green”, “dog”. That satisfies this rule because even though the first three fields are the same for the two rows, the last field, *favorite animal*, is different which makes the rows unique from one another. Just one value has to be different for rows to be unique from other rows. Rows must be unique from one another because if they’re not, then it essentially means that there are one or

more duplicate rows in the table. If this were the case, it leaves room for error if a change were made to the table. One of the duplicate rows may get changed and the other may not and then the information becomes inaccurate.