Jessica Rieger 09/11/2016
Lab 2

1. A super key is the most general of the key categories. It is essentially one or more attributes that collectively identify a row in a table as unique. You can have many super keys that have different attributes and each succeed in identifying a row as unique. However though this may sound like a candidate key, it is different because you can remove any of the attributes in the group and the row will still be unique. For candidate keys, you must have all the attributes that you started with to ensure that a row is unique. If you remove an attribute, then the row is no longer unique. You can have multiple candidate keys that combine different attributes to make a key. Of the many candidate keys, the primary key is simply the one that you have chosen to be the main key. You can only have one primary key and the other candidate keys leftover are simply labeled as alternate keys.

2. Data fields are very important in that they allows us to identify the type of the data/information that we are viewing. There are several different types that can be useful, but some of the common ones are integer, decimal, char(), and varchar(). Each of these data types has unique constraints associated with them that limit what type of information we can put as content for that column. For example, a char(n) data type limits the length of the string to n characters. By using data types, we can add another layer of checking to ensure that we don't end up putting an integer where a string should be. As well there is also the declaration NULL which, though it isn't a data type, can be a place holder for a data value and indicates that we either don't know the value that should be there or that the value for that specific row is not relevant.

To give an example, we can assume that we have a table named Pets that keeps track of all the pets in a pet shop. This table will have several fields, each with an associated data type. The fields and their data types are as follows: PetID int, PetName varchar(), PetType varchar(), WeightINPounds decimal, Color char(15), AllergyFood varchar(). The fields that are nullable are WeightINPounds, Color, and AllergyFood. The reason that these are nullable is that they do not relate to all animals. For example, though a dog has a deathly allergy to grapes, other animals may not have one particular thing they can't eat, thus NULL would be appropriate. The other fields are not nullable because they are required to identify the animal or the pet shop decided they wanted to require these fields.

Below I have included an example of what the table would look like to demonstrate the things described above. As you can see, there are values that are left NULL, and all the values in each column are of the same data type.

| PetID | PetName | PetType | WeightINPounds | Color | AllergyFood |
|---|---|---|---|---|---|
| 1 | Poppy | Cat | 15.5lbs | Brown/white | Milk |
| 2 | Tweety | Canary | NULL | yellow | NULL |
| 3 | Chad | Dog | 20lbs | Gray | Grapes |

3. The first rule is the "first form normal" rule. This rule states that all values/fields must be atomic. This means that they are in the smallest possible unit and they have no internal structure. If all the "fields" in the table follow the rule, we say that the table is in first normal form. An example of this would be that the number 2 would be an ok field, however the equation 2+3=5 has clear internal structure and thus would be a violation of the rule. To fix this we would need to have columns for the values to be added and the total in order to make the table free of violations. This rule is important because it helps eliminate repeating groups (or columns that have the same or similar value) by simply creating a whole new relation that connects to the original relation but stores the similar column values in their own table. Going back to the equation example, we would have a table of things to add and maybe a table of totals, which related back to the original relation via a primary/foreign key relationship. As well, it allows us to access each piece of data more specifically. For example, if we have multiple strings in one field, there is no way to simply access one as the result of a query. In a database where the main purpose is to store data to be accessed, this seems like a major problem.

The second rule is best described as "what, not where." This means that we are never allowed to access data based on its location in a table (meaning by row and column.) For example, instead of querying for the field at row 3, column 4, we would say to "return all orders associated with the customer Chris." There are two reasons this is important. First, by asking for a field by its location it deprives the user of the context that is included with the correct way to write a query (essentially who the data retrieved is related to and what it represents.) Second, the tables in a database are sets of information so the order of the rows does not matter. This means that the 3rd row could be different every time you query the database. Just because one field was located in row 3 when you queried the first time, that doesn't mean changes to the database haven't been made which changed the location of your desired field.

The final rule states that all rows must be unique. This means that at least 1 column value must be different between all the rows. Or to put it another way, no two rows may have the same values for all of their columns. For example, if we have a table with the columns name and customerID, we can never have two rows that have John for the name and 5 for the ID number. This rule is important because it eliminates duplication of data and allows us to keep more accurate records.

One thing I would like to note about all of these rules is that, according to the actual database program, any of them can be broken legally. Essentially, they are just practices that we should follow in order to maintain a clean, accurate, and usable database. However, it is up to the designers of the databases to enforce these theoretical rules.