Lecture 6 Notes - Monday 10/03/16

Reading Quiz

Question 1: Ans = D
Question 2: Ans = C
Question 3: Ans = B
Question 4: Ans = A
Question 5: Ans = C

Notes

Note that the due dates for Labs 2 and 3 (which you can find on the website) have been pushed back to give you an extra day in class for setup and lab work

We recommend that you do the exercises from the *Learning SQL* book chapters to get familiar with SQL commands

Bank ERD: note that this model does not allow for joint ownership of an account -- to do this, we would need a many to many relationship between account and customer, which would require a junction table -> the junction table would probably have a composite primary key composed of *account_id* and *cust_id* ; the relationship might have some other attributes in the junction table, like *primary_account, primary_owner*, etc.

Concept Question 1: D. We should populate the *customer* table first because it is the only table that does not have a foreign key referring to another table. If we had tried to populate another table first (say, *account)*, MySQL would get mad at us because *cust_id* is supposed to point at a record in the *customer* table, but no records exist yet in *customer*! Using the same reasoning, we would populate *account* next, and then *transaction, business,* and *individual* could be populated in any order; since *D* is the only option that starts with "Customer, Account,..." that must be the right answer. (Note, we could also populate *business* and *individual* before *account*, since they only depend on the *customer* table. The point is just that *account* must be populated before *transaction*.) These constraints that require foreign keys to have something to point to are part of the idea of *data consistency*.

Insert Statements: Note that Option 2 is basically just a shortcut version of Option 1. To do this, you have to have values for every column, and these have to be in the same order as the table's columns. Option 3 is for if you have data in a table already, but want to insert it into a new table -- in a sense, you query the old table and copy/paste the rows resulting from the query into the new table.

Concept Question 2: E. The special characters should be fine as long as you are using a Unicode character set, which MySQL is fine with. (Just don't use a single quote (') without escaping it.) So *A* isn't the right answer. You don't necessarily need an auto-generated sequence -- you could be doing this in the program you use to insert records (like a Python program). As far as C goes -- maybe not great but not really an issue. Everything will work out fine with the table. *Product* is perfectly descriptive, so not *D.* So the only answer left is *E.*

But there *is* something wrong with this table -- it's not in 1st Normal Form. The attribute *contact_id* contains a list, which is not cool. This is problematic. Imagine if you want to do any queries with *contact_id*; that's gonna get tricky. And even removing a *contact_id* from a row would be tough.

Concept Question 3: C. You can see in the CREATE TABLE statement for *Product* that we are auto incrementing the primary key, *id*. So if we insert a custom *id* value for one row, but let MySQL auto-increment the next, it'll ignore whatever *id* values you might have skipped over when you chose your own.

A is wrong because we just saw a couple of slides ago that you can do this.

B is on the right track, because the *id* column is problematic, but in fact the latest version of MySQL won't "overlap" *id* values when it auto-increments. (Other Database engines might have this issue, though)

D -- nope, MySQL will in fact let you choose your own value for an (otherwise) auto-incremented column

Update Statements: kind of the same idea here as with inserts, except you don't have to include every column because you are specifying which values you want to set. Option 2 is useful because we don't necessarily want to update the whole table -- so we use a WHERE clause to filter what rows we want to change. Note with Option 3 that we have to do a query that returns a single value, otherwise we'd be storing basically a list inside of one "cell"