Deliverables

Each group must provide the following as a **single** PDF file:

1. A completed cover page (template on Canvas)
2. A brief (~2-3 sentences) summary of your project. Many of your TAs are managing multiple projects so this will help them remember details about your project.
3. The ER diagram you are basing your item #3 (below) on. This ER diagram may be the same as your milestone 1 submission or it might be different. If you have made changes from the version submitted in milestone 1, attach a note indicating what changes have been made and why.

   If you have decided not to implement the suggestions given by your project mentor, please be sure to leave a note stating why. This is **not** to say that you must do everything that your project mentor says. In many instances, there are trade-offs between design choices and your decision may be influenced by different factors. That being said, your TAs will often leave suggestions that are meant to help massage your project into a form that will fit with the requirements in future project milestones. If you choose not to take their advice, it would be helpful for them to know why in order to better assist the group moving forward.
4. The schema derived from your ER diagram (above). For the translation of the ER diagram to the relational model, follow the same instructions as in your lectures. The process should be reasonably straightforward. For each table:
   a. List the table definition (e.g., Table1(attr1: domain1, attr2: domain2, ...)). Make sure to include the domains for each attribute.
   b. Specify the primary key (PK), candidate key, (CK) foreign keys (FK), and other constraints (e.g., not null, unique, etc.) that the table must maintain.
5. Functional Dependencies (FDs)
   a. Identify the functional dependencies in your relations, including the ones involving all candidate keys (including the primary key).

      PKs and CKs are considered functional dependencies and should be included in the list of FDs. You do not need to include trivial FDs such as A → A.

      Note: In your list of FDs, there must be some kind of valid FD other those identified by a PK or CK. If you observe that no relations have FDs other than the PK and CK(s), then you will have to intentionally add some (meaningful) attributes to show valid FDs. We want you to get a good normalization exercise. Your design must go through a normalization process.

6. Normalization
    a. Normalize each of your tables to be in 3NF or BCNF. Give the list of tables, their primary keys, their candidate keys, and their foreign keys after normalization.

       You should show the steps taken for the decomposition. Should there be errors, and no work is shown, no partial credit can be awarded without steps shown. The format should be the same as Step 3, with tables listed similar to Table1(attr1:domain1, attr2:domain2, …). ALL Tables must be listed, not only the ones post normalization.
7. The SQL DDL statements required to create all the tables from item #6. The statements should use the appropriate foreign keys, primary keys, UNIQUE constraints, etc.

   Unless you know that you will always have exactly x characters for a given character, it is better to use the VARCHAR data type as opposed to a CHAR(Y). For example, UBC courses always use four characters to represent which department offers a course. In that case, you will want to use CHAR(4) for the department attribute in your SQL DDL statement. If you are trying to represent the name of a UBC course, you will want to use VARCHAR as the number of characters in a course name can vary greatly.
8. INSERT statements to populate each table with at least 5 tuples. You will likely want to have more than 5 tuples so that you can have meaningful queries later.

Note: Be consistent with the names used in your ER diagram, schema, and FDs. Make a note if the name has been intentionally changed.

Note: As you start analyzing these requirements, you may notice that certain details are missing. In this case, you may make any reasonable assumptions about them; but, if there is any uncertainty about some requirements, you should ask your project TA before proceeding further (or if it's more general in nature, post your question on Piazza). Furthermore, it is acceptable to modify your design from your original project proposal, because as you progress and start thinking more about the data and the queries that you want to answer from your application, it is normal to find that you need to modify the design (but don't go back and re-do or re-submit your project proposal).

WARNING: Do not start on the implementation portion of the project until you complete tutorial 6 /7/8. Tutorial 6/7/8 will offer a chance for you to try Oracle/Java, Oracle/PHP, and Oracle/Javascript. It is likely that you will not know what you enjoy working with until you finish these tutorials.

Check the milestone 2 assignment on Canvas for the grading rubric. Refer to the syllabus for information on late submission/penalty rules.

FAQ

1. **I want to test my SQL DDL CREATE and INSERT statements, what is an efficient way to do this?**
   Use SQLFiddle. Later in the course, we will be introducing SQL Plus and how to use it run SQL statements from a file. At that point, you can put all your DDL statements into a single file and run it.

2. **My relational schemas do not have functional dependencies beyond what is specified by primary keys and candidate keys. What should I do?**

   A good way to go about this would be to identify what constraints you need in your data (and potential sources of user error on data entry). As a reminder, not all constraints can be represented through the ER diagram. For example, we discussed an Address entity in class where the Address entity had House#, Street, City, Province, and Postal Code as attributes. In this case, you would want a FD that says Postal Code determines province (i.e., if I know the postal code, I know what province the postal code is in). This would mean that you prohibit data entries such as "*<1234, West Broadway, Vancouver, BC, H3C 2Q0>*". H3C, while a beautiful neighborhood in Montreal, is not a BC postal code, and therefore wouldn't make much sense in this tuple.

3. **How many non-primary/candidate key functional dependencies do I need to have?**
   The number of FDs your project will have will heavily depend on the domain and direction of your project. There are some domains where there are lots of FDs present due to the semantics of the data. However, there will also be other domains where there are less of these constraints.

   You should come up with as many FDs as the real application dictates. These will make your normalization more interesting. The objective of Milestone 2 is to help you learn normalization in a "real world" setting, not just the academic settings you encounter in exams and practice questions. You may do well in the latter; but when you become a relational database designer, the former is much more important.

   We warmly recommend that you talk with your project mentor about specifics. Please note that your project mentor will not pre-grade your work for you (as in, they will not tell you if you have enough FDs or if you are missing any) but if you talk about your overall process and reasoning, they can tell you if are heading in the right direction.

4. **Are FDs limited to a single relation?**
   Functional dependencies are statements about how values in your data are related. If the association is spread across several relations, that is fine.

5. **Does normalization happen in the real world?**
   It depends on various factors. There is significant trade-off between having many tables with minimal or no redundancy and very few tables with significant data duplication; there are production level databases at both extremes. For example, there is an open source contact management system that uses over 150 tables— though we have not looked at it close enough to see how normalized it is. Some of your TAs have also been in a position where they worked on a single table even though it meant a lot of duplication.

   Normalizing reduces duplication and saves space but increases complexity of queries and the need for joins (which can be computationally expensive). On the other hand, less normalizing means having to deal with potential duplication, which is sometimes preferred. For example, if duplicated attributes rarely change but are read frequently, it can be better to update each of them when necessary than join multiple tables frequently for many reads. Think of it almost like caching frequently accessed data in a separate attribute.

   All in all, it depends. The size of the tables and the storage space required, the frequency of reads vs writes, hardware factors such as storage IO speed and memory availability, number of concurrent accesses, etc. can all be contributing factors. Consequently, knowing what normalization is and how it works is beneficial when making those decisions, or understanding why prior decisions were made.

6. **Oracle doesn't support ON UPDATE. What do I do?**
   As mentioned in class, Oracle does not support on update cascade.

   For your project, feel free to do something else that makes sense to you (your choice). In your milestone, mention that you know that it should be "on update cascade", but Oracle won't support it.