CS 325 - Exam 2 Review Suggestions

last modified: 2016-11-02

- You are responsible for material covered in lectures and labs, and especially anything that's been on a homework or lab exercise; BUT, here's a quick overview of especially important material.
  - It is strongly advised that you study posted examples and notes, and make sure you can do exercises such as those on homeworks and lab exercises;
  - (Note that if your understanding of the material is not strong enough, you may have difficulty completing the exam within its time limit.)
- You are permitted to bring into the exam a single piece of paper (8.5" by 11") on which you have handwritten whatever you wish on one or both sides. This paper must include your name, it must be handwritten by you, and it will not be returned.
  - Other than this piece of paper, the exam is closed-note, closed-book, and closed-computer.
- Note that you are responsible for all material covered through **Homework 7** and through **part 1 of converting a database model to a database design/schema**; you are not responsible for part 2 of converting a database model to a database design/schema, nor for material discussed during Week 11 (although all of that material **will** be included in the material for the Final Exam).
- This will be a pencil-and-paper exam; you will be reading and writing SQL*Plus commands and SQL statements in this format, as well as answering questions about concepts we have discussed.
- Note that you are also responsible for knowing -- and following -- the course SQL style standards and the course ERD notation.
- With a few exceptions, the **focus** of **most** of the Exam 2 questions will be on material covered near the end of the Exam 1 content period and on the **new** material covered since Exam 1.
  - However, since much of the new material is "built" upon previous material, that previous material will still be involved; much of this is cumulative.
  - Also, note that you **will** be asked to write at least one query involving a join.

More on Modeling

- [This was on the Exam 1 Review, also, BUT since it was covered near the "end" of the Exam 1 content window, I am including it here, also, and considering it to be "fair game" for Exam 2.]
- What is a recursive relationship? How is it depicted in an ER diagram?
- What is a weak entity class? How is it depicted in an ER diagram?
- What is a supertype entity class? What is a subtype entity class?
  - How is a supertype/subtype relationship depicted in an ER diagram? (Remember to follow class style standards for these)
  - What is meant by having a `d` in the circle in depicting supertype/subtypes entity classes? ...having an `o` in that circle? ...having a `u` in that circle?

Normalization

- What is **normalization**? What is its purpose?
- What are modification anomalies?
  - ...deletion anomalies?
  - ...insertion anomalies?
  - How does normalization reduce/get rid of some of these?
- Be aware of the tradeoffs involved in normalization;
- Remember: normalization does not eliminate data redundancies;
  - It tries to reduce/eliminate UNNECESSARY data redundancies(although reducing modification anomalies is a more important goal, I think);
  - It produces controlled data redundancy that lets us link/join database tables as needed.
- Also note: when you normalize, you generally introduce referential integrity constraints.
  - (What are referential integrity constraints? how can they be handled/implemented in Oracle SQL?)

Normal Forms

- Note: because of their importance in understanding normalization and normal forms, the definitions for functional dependency, superkey, minimal key/candidate key, and primary key may also be the focus of some Exam 2 questions.
- What is a partial dependency?
- What is a transitive dependency?
- 1NF, 2NF, 3NF (first normal form, second normal form, third normal form)
  - EXPECT to have to normalize sets of relations to these normal forms;
  - Could also be questions about them in general, also;
- what does it mean if a set of relations is in 1NF? ... 2NF? ... 3NF?
  - What kinds of anomalies are reduced/eliminated when a set of relations is in each form?
  - If a set of relations is indeed in 1NF, what can no longer exist? If it is in 2NF...? If it is in 3NF...?
- BCNF, 4NF, 5NF, 6NF (Boyce-Codd Normal Form, fourth normal form, fifth normal form, sixth normal form)
  - Only need to know that they exist, and how they "relate" to one another (a set of relations in 2NF is also in 1NF; a set of relations in 3NF is also in 1NF and 2NF; ... a set of relations in 6NF is also in 1NF, 2NF, 3NF, BCNF, 4NF, and 5NF);
  - I **won't** ask you to normalize sets of relations into these forms.
- What is **denormalization**? Why would we do that? Why do we not always normalize "to the max"?

Converting an ER Model into a Database Design/Schema

- recall: a **database schema/design** defines a database's structure:
  - its **tables**, (which includes each table's attributes and primary key),
  - **relationships**,
  - **domains**, and
  - **business rules**
- what is (should be) the database development process? (come up with a data model, THEN convert that data model into a database schema/design!)
  - and then normalize further if necessary --- this is a good "double-check" on the model/design process, making sure that your relations are in 1st, then 2nd, and then 3rd normal form;
  - BUT note that, if you have really modeled the entity classes in your scenario, you may not have to do much in this normalization. The "themes" that are separated in normalization should/could correspond to entity classes...
- remember: an entity class is NOT equivalent to a table or relation!! (eventually, each entity class will *result* in *one or more* corresponding tables/relations in the database schema/design that we develop from a model;)
- how do maximum cardinalities affect the eventual design?
- how do minimum cardinalities affect the eventual design?
- EXPECT to have to convert models into appropriate sets of relations;
  - what are the considerations when deciding upon primary keys for each "base" table?
  - how are multi-valued attributes handled?
  - if you know that a single-valued attribute should always have a value for some entity class (that is, that it should not be permitted to be `NULL`), what SQL feature would you want to use in defining its (physical) domain within a `create table` statement?
  - be sure you know how to handle 1:N, M:N, 1:1 relationships;
- Handling 1:N, M:N, 1:1 relationships:
  - how do you change the relations involved?
  - when are additional relations necessary?
  - what are the primary keys of the tables involved?
  - which attributes need to be foreign keys reflecting referential integrity constraints?
  - why should you sometimes view mandatory 1:1 relationships with suspicion?
  - when does it matter which "base" table gets the foreign key in handling a 1:1 relationship? When does it not matter?
  - how do you tell which is the "parent", and which is the "child", in a 1:N relationship? Also need

to know which one will get the primary key of the other placed in it as a foreign key;

- what is an intersection table? When is one needed in a design/schema? What does an intersection table include, and how is it implemented?

- how do you implement a foreign key in SQL? ...a primary key in SQL? ...a multi-attribute primary key in SQL? (this had better be review! BUT it could very well be on the test.)

LAB-RELATED TOPICS:

- EXPECT IT --- you will HAVE to write at least one join using SQL.
  - that join might involve more than two tables, of course;
  - IF a `FROM` clause contains N tables --- how many join conditions should there be, to make that Cartesian product of N tables into an equi- or natural join?
- EXPECT IT --- you will be required to read AND write proper syntax SQL and SQL*Plus statements.
  - (by "read", I mean that I may give you a statement and ask you questions about it; I could also give you various table contents, and ask you what the results of running a given statement would be;)
  - and, of course, I could ask you to write a SQL statement that would perform a specified action or query;
- Note -- even through IN, AND, OR, and NOT, and the comparator operators <, >, =, etc., were fair game for Exam 1, they are also very likely to play a part in Exam 2 as well. There are important aspects to how they are used with nested selects, for example.

SQL Support for further restriction of domains

- be comfortable with further ways you can restrict the domains of attributes (`not null`, `default`, `check`)
- Which version of `insert` do you need to use to make sure you get any `default` values for attributes that have them?

Sub-selects/nested selects

- What is meant by a sub-select/nested select? What are some of the possibilities for where these may be placed within a `select` statement?
- EXPECT to have to read AND write nested queries on this exam;
- EXPECT to have to write queries in SQL that answer a given "question" in different ways --- one part might ask you to answer it using a join, another might ask you to answer it using a nested query, another might ask you it answer it using `EXISTS` or `NOT EXISTS`, etc.
  - IF, however, I do **not** specify that a certain style or feature be used, then you may answer it however you like, as long as it correctly answers the question and follows class style guidelines,

of course.

`IN` operator

- covered in Exam 1, but it is so important/useful used with sub-selects that it will be covered on this exam, also.
- why is `IN` sometimes the better/more correct choice for use with a subquery rather than =?

Projecting a constant literal and concatenation

* what does it mean to project a literal constant string, such as `'a'`?

```
select 'a'
from   empl;
```

* how can this be useful, especially when combined with concatenation (`||`)?

`EXISTS` and `NOT EXISTS` predicates

- expect at least one question reading these, one question writing these;
- what's the "big" thing to remember? (that the subquery that is the right-hand-side of the `exists/not exists` predicate had better be **correlated** to the outer query!)
- what is a correlated query? what is a correlation condition?
  - how does a correlated query differ from a "plain" nested query?
  - how does a correlation condition differ from a join condition? How can you tell the difference between them? (this is important when dealing with `EXISTS`, `NOT EXISTS`)

`ORDER BY` clause

- EXPECT to have to read and write `SELECT` statements including this clause;
- what does this clause do?
- how do you order rows in increasing order? in descending order?
- what does it mean if there is more than one attribute in an `ORDER BY` clause?
  - remember to specify `desc` *separately* for EACH attribute that is to be in descending order;
- remember: this clause SHOULD NOT be used in a sub-select, only in an outermost select

`GROUP BY` clause

- EXPECT to have to read and write `SELECT` statements including this clause;
- remember: `GROUP BY` lets you group the rows in a table based on equal values within specified columns;

- if there is **no** `GROUP BY` clause, how many **rows** will **ALWAYS** be in the result of a `select` statement projecting an aggregate function result? How many **can** there be if there **is** a `GROUP BY` clause?
- what is allowed in the `SELECT` clause when you have a `GROUP BY` clause within a `SELECT` statement?
- what does it mean if there is more than one attribute in a `GROUP BY` clause?
- is grouping done **before** or **after** any row selection specified by a `WHERE` clause?
- be careful not to confuse `ORDER BY` and `GROUP BY`; remember that `GROUP BY` does not infer any ORDER that the resulting groups will be displayed --- if you want that, you SHOULD include an `ORDER BY` clause, also.

`HAVING` clause

- EXPECT to have to read and write `SELECT` statements including this clause;
- remember: a `HAVING` clause must be used in conjunction with a `GROUP BY` clause;
- it lets you limit which GROUPS you'll see in the result;
- so, `HAVING` is to groups what `WHERE` is to rows, kind of;
  - be careful to understand the difference between `WHERE` and `HAVING`.