**Test 1 Review**

**Test is on Tues, Sep 27**

* Review all Chapter 1 Questions (the google doc)

**Additional terms (see Chapter 3)**

* **Tuple** – A table row, which corresponds to an entity.
* **primary key** – the unique and not null identifier of all entities in a relation. Also, has an index.
* **Determination**  - The state in which knowing the value of one attribute makes it possible to determine the value of another. This forms the basis of the idea of a key.
* **functional dependence** **(determinant, dependent) –** The value of one or more attributes determines the value of one or more other attributes. The **determinant** is the attribute that determines the value of the **dependent** attribute.
* **composite key** – a key made up of multiple attributes
* **Null** – is not a value. This could be because the value is unknown, doesn’t exist, or doesn’t make sense. **Flags** can be used to handle values that would otherwise be null. A NOT NULL constraint can also be placed on an attribute. Unique can also enforce multiple values in a column not being null.
* **candidate key** – an irreducible super key
* **super key** – a key that uniquely identifies all of the entities in a relation
* **foreign key** – a key, which can be null, that corresponds to the primary key of another relation
* **secondary** **key** – a key used solely for data access purposes
* **entity** **integrity** – the property that each entity should be uniquely identifiable and not null (pk)
* **referential** **integrity** – the property that a foreign key should not point to something that does not exist. It is also important that the foreign key point to the primary key of the sister relation.
* **system** **catalog –** The system data dictionary that describes all objects within the database. This is broader than the data dictionary, which focuses on tables.
* **data** **dictionary** – the part of the DBMS that holds onto the metadata and relationships in the database. It is the description of all tables in the database created by the user and designer.
* **Homonym –** This term refers to the same name being used to label different attributes. These should be avoided to lessen confusion.
* **Synonym –** This term refers to different names being used to label the same attribute. Should be avoided for the same reasons as the above.

1. Why do we try to avoid designs that will require frequent use of nulls?

If we have too many nulls, that means we have a lot of **ambiguous** data. It’s important to avoid ambiguity as much as possible. We don’t want a bunch of “not applicable” data to appear in our tables. If you have rows that don’t fit the table structure time and time again, then you should rethink it.

1. Let’s say we are creating a Student table. Every student must have a major. What can we do (when creating the table) to ensure that we never have a student with null for the major?

We can use a constraint on the major attribute that requires an entity to have a non-null value for major. If we do this, the DBMS will enforce the constraint. **Use the not null constraint**.

1. Considering relational algebra: (Homework #1)

Relational algebra refers to the theoretical way of manipulating table contents using relational operators. **Relvar** is the term given to a variable that holds an entire relation. The heading contains attributes, and the body is the relation itself. The relational operators exhibit the property called **closure**, which means that using the relational operators on relations produce new relations.

* 1. What does it mean to say that two tables are union compatible?

Two relations are union compatible when they have the same number of columns and the corresponding columns share compatible domains. This means that they attributes are of the same or compatible data types.

* 1. Given two tables, be able to show the results of: natural join, left outer join, right outer join.

The natural join multiplies the tables together, selects the matching values in the attributes that have the same names, and then drops one of the redundant attributes. A left outer join returns all of the values in the first table, and the matching ones in the second table, with nulls for those that don’t match. The right outer join is the same thing but the second table retains all of its values. Remember that an outer join is an inner join +.

* 1. Given two tables, show the result of a divide operation

This needs to be done on paper, but here are the steps:

1. Use one 2 column table as the dividend

2. Use one 1 column table as the divisor.

3. The output is a single column that contains all values from the second column of the dividend that correspond to every row (value) in the divisor.

* 1. What is the difference between select and project?

Select grabs a tuple and project grabs an attribute. It’s row vs. column.

* 1. What is the sequence of operations used to perform a natural join?
     + 1. Multiply the two tables
       2. Select only to get rows where the primary key and foreign key are equal
       3. Project to remove duplicate columns.
          1. Equijoin is the same except the join columns don’t have the same name. This means the join columns must be explicitly defined and one of the duplicates is not automatically dropped.
          2. Theta join is the same as “a” except the operator does not have to be an equals.
  2. As implemented in SQL, what is the practical difference between the following: natural join, equijoin, theta join

In the “practical sense”, there is no difference between these kinds of joins. SQL handles all of this stuff for you. A natural join is simply a join where common columns are connected and one of the identical ones is dropped. An equijoin is a join where values are equal. A theta join is a join where arbitrary logical operators are used instead of an equals.

* 1. What notation is used in SQL to perform a select operation?

In SQL, the keyword SELECT with the WHERE helper does a relational algebra select operation.

* 1. What notation is used in SQL to perform a project operation?

In SQL, the keyword SELECT also does projects. When you select something you are selecting attributes.

1. Given descriptions of problem-domain entities (similar to what we have practiced in class), identify the following relationships: 1:M, 1:1, M:N and draw an ERD that models the entities.

See the sample drawing in my class notes. I will discuss the relationships here really briefly though. The 1:1 relationship means that the on either side of the relationship, there can be only one entity. This is like a husband in a wife in my tradition. There can be one and only one.

The 1:M relationship is like that of a painter to paintings. A painter has many paintings, but a painting has only one painter.

The M:N relationship refers to many to many. Many students enroll in many sections of courses. This kind of relationship requires a **bridge table**. There is no other real way to get around it.

Remember that foreign keys go on the side of the many, and the crow’s foot diagram looks like the foot of the crow is grabbing the thing that holds the foreign keys

1. Be able to draw (by hand) ERD (crow’s foot) diagrams for each of the above relationships. See the sample exercise at the end of this review sheet.

See the drawings in my class notes.

1. Be able to define tables that correctly implement each of the above relationships.

Again, this is really more of the same. If you look at the drawings for the stuff on the board, the attributes are held in the boxes. It is only a matter of displaying them in table form and adding data.

1. Given descriptions of problem-domain entities and existing table definitions, identify whether the data in the table is redundant.

This can be a little tricky. It is going to rely on whether or not there is a historical reason—or perhaps for some failure critical information—to keep duplicated data around. It is fairly easy to see data that doesn’t belong, or should belong in a bridge table. Whenever you see some data that could easily become out of sync, it begins to be suspect. Use **foreign keys** to help control this.

1. Describe the scenario where the DBMS will automatically create an index.

When a primary key is identified, the DBMS automatically creates an index for it.

1. True/False: The index is a separate file from the data that it is indexing.

True.

1. How would you know whether you should create a secondary index on a column?

You should create a secondary index whenever you are going to be searching based on that attribute. Doing this will allow for searching without having to deal with all of the extra attributes in the tuple.

1. In general, what does an index make *faster*?

Indexes make searches faster, which also means faster inserts since things can simply be appended into the database without having to sort the massive amount of data on disk.

1. What is the cost of an index (i.e., what extra data/processing do we have to do because we have the index)?

Indexes save in search speed, but they cost in space. The more indices used in a database, the more files exist. When the files are small, that isn’t much of a problem, but for a large database, an index file can also become very large.

1. If binary search is so great, why not just binary search directly on the data file, instead of using it on the index?

The data on file is a whole lot larger than the index file. The index file essentially just has a key and a record offset in it. That is a whole lot less data to move around than ALL of the attributes in all entities, etc.

1. Consider a flat index:
   1. Describe, in general, what data an index actually contains.

An index contains a key value and an offset to the beginning of the associated data on file in the DB.

* 1. Explain/Justify how use of the index will make searches faster than *not* having an index.

Performing a binary search on a lighter weight file is faster by definition. Additionally, the index is physically sorted in order of the key values. This allows for binary search because binary search requires things to be sorted. So, it is smaller and we can keep it sorted for binary search.

* 1. Explain how use of the index would make insertions of new records in the data file *faster* than not having an index.

Using an index means you don’t have to rewrite the data that is actually on file. You can stick the data on file just at the end of the file. That way you don’t have to maintain the actual database in sorted order to do binary search on it.

* 1. Describe the scenarios in which the data in the index itself will have to be re-written.

This should happen when there is an insertion or deletion into the data on file. If you update the indexes key, that could cause a rewrite as well. So, these can cause rewrites, but they are a lot cheaper than sorting the whole database. Remember, these are stored in sorted order, so they index will have to be changed. **If it causes the key value to change or the record to move, then there needs to be a rewrite.**

1. Consider an index implemented as a B-tree:
   1. In general, how would the size the B-tree (on disk) compare to the size of a flat index (assume both indexes are indexing identical data files).

The B-Tree is larger. It has all of the same information that a flat key has, but it also has a hierarchical structure that contains more data. It has extra indices, so it has to be bigger.

* 1. What process(es) would a B-tree make faster than using a flat index?

Search is faster because the indices are shallower. The insert is faster as well. Worst case, you will have to update as many blocks as the tree is tall. In a flat file, you have to update all of the blocks. That is the true win of the B-tree. The shallower the tree, the better.

* 1. What would be a typical number of key-ptr pairs in a node in a real B-tree?

Something on the order of at least 1024 is what we would expect. The number could be substantially higher. This allows the B-tree to become very wide before it becomes tall. The shorter the B-tree the better, because the worst case is we have to check as many blocks as there is height essentially.

* 1. If a B-tree has 4 levels, what is the worst-case number of disk blocks we would have to read/search?

4. Since it searches with a binary search, or something similar, it is able to follow the keys to the next level in the correct direction. This means it will never check two blocks on the same level. That is the “branching” aspect of a B tree.

1. Given a hypothetical sequence of key values, show the resulting B-tree that would be constructed to hold the key values.

Be able to build the B-tree. See notes.

1. Given a small sample table, show the contents of an index on a specified column of that table.

An index is a key and an offset to the beginning of the record on file in the database. So, this question is asking for one to identify the keys and their order in the index based on the column. I think the trick here will be to remember that the data in the database is stored in entry order. The data in the index is sorted by the key.

1. Misc Homework Notes:
   1. **When the book leaves a field blank, they mean null.**
   2. Referential integrity does not demand that every foreign key have a value, it just says that if it does have a value, it must reference something that exists.
   3. Know/use Crow’s foot (not Chen…see p. 122 for the difference)
   4. The term “superkey” refers to any key that can uniquely identify a row. However, “super” means that it *can* have *more* attributes than necessary to do this. A *candidate* key doesn’t have any superfluous attributes.
   5. **A bridge table** ( like BENEFIT on p. 111) **would use a composite key for the primary key**. Just because it doesn’t have a column labelled Benefit\_ID doesn’t mean it doesn’t have a primary key.
   6. Problem 4 (p. 308). The correct answer is COMMIT, but our admin tool automatically commits, so it’s pointless for us while using the admin tool.
   7. Problem 7 (p. 309). Similar to 4, the correct answer is ROLLBACK, but that won’t have any effect for us because the admin tool automatically committed (you can only rollback prior to a commit).
   8. Problems 24-25: These do not require subqueries. They were trying to illustrate the fact that you can query the results of another query (i.e., you could do a subquery to produce P7.22, and use that as the input to a Select). Something like:

SELECT Sum(SumOfASSIGN\_HOURS) AS SumOfSumOfASSIGN\_HOURS,

Sum(SumOfASSIGN\_CHARGE) as SumOfSumOfASSIGN\_CHARGE

FROM (SELECT ASSIGNMENT.PROJ\_NUM,

Sum(ASSIGNMENT.ASSIGN\_HOURS) AS SumOfASSIGN\_HOURS,

Sum(ASSIGNMENT.ASSIGN\_CHARGE) AS SumOfASSIGN\_CHARGE

FROM ASSIGNMENT

GROUP BY ASSIGNMENT.PROJ\_NUM

);

That is a nice thing to know how to do, but in this case it is a pathological way to do it, since you can get the data from the original Assignment table.

1. SQL Topics:
   1. Closed Book
      1. Be able to *read/explain* any SQL statements of the type covered in the SQL lab and Homework #3.

Reviewed the homeworks…

* + 1. How do you know when to use GROUP BY?

You use a group by whenever you need a “many “ result in conjunction with an aggregate function that returns a single result. The big picture is you want to get one result out of groups rather than out of a whole table.

* + 1. What is wrong with the following?
       1. select max(age), name from people

You are saying both that you want one and many results. This is illegal. You have to use a group by on name in order to get this to work. Or you need to just choose which one.

* + 1. Why does the following not work to give you the oldest person’s name:
       1. select name from people where age = max(age)

This doesn’t work because aggregate functions only work in the context of projecting. You have to use them in a SELECT, on the left side of WHERE.

* + - 1. How do you handle this sort of problem?

To handle this, you can give a nested query in the WHERE clause in order to handle that. So in the WHERE, you would write: AGE = (SELECT MAX(AGE) FROM TABLE);

* + 1. How do you know when you need to use HAVING?

HAVING is a WHERE on the results of a GROUP BY. This eliminates some of the groupings.

* + 1. Be able to answer questions 1-4, 7-9 on p. 306-307
  1. Open Book
     1. Be prepared for any question similar to what you did in SQL lab or Homework 3

Reviewed the homeworks…

* + 1. (You will turn in your paper test prior to working on the SQL portion of the test).

**Sample ERD Exercise**

Considering OC’s course schedule and related entities, create an ERD (using Crow’s foot) to describe the entities, their attributes, and relationships. You do not need to go beyond the information listed below, and you do not need to show data types.

* Course
* Time
* Room Number
* Course Description
* Credit Hours
* Instructor
* Section
* Section number
* Capacity (i.e., can’t enroll more than 20 people in this section)
* Status (i.e., whether closed)
* Course Number (i.e., CMSC 1113)
* Instructor’s Office Hours
* Student
* Student ID
* Student Name
* What classes a student is currently enrolled in

SEE NOTEBOOK