CSC 321 – Database Management Systems
Sample Test 1

This sample test has a few more "cascading" questions than I plan the actual test to have (problems where if you mess up one part, the following parts will likely be wrong too since they rely on earlier parts).

1. Consider a flight reservation system. The system should keep track of customers' data, flight information, and seats reserved by the customer. A reservation has a unique number and is for a single passenger. Customers can book tickets on flights on specific days. They may indicate preferred airlines and maximum amount they are willing to pay. Each customer is identified by his/her credit card number, and must give a contact phone number and address. A flight is operated by an airline, has a flight number (unique only within that airline), and operates on a given set of days of the week. The reservation must contain the row and the seat information of each passenger and the price paid for the reservation. A customer may have multiple reservations.

a. You need to build an E-R model to represent entity sets and their relationship sets of the flight reservation database, including mapping cardinalities and primary keys.

See accompanying slides

b. Transform your E-R model into relational tables, give the schemas, underline the primary keys, and mark the referential integrity (foreign key) constraints

See accompanying slides

c. Suppose you want to know the name of the passengers, the airline, the flight number, and the seat assignments of all the passengers who paid $1,000 or more for a single ticket. Give an SQL query and relational algebra statement that answer this question.

Turkett Typo: There is no "name" in the description of the system. Assume it is a property of "Traveler".

See accompanying slides

2. Describe the difference between natural join and theta-join when the condition of the theta-join is to equate the attributes appearing in the schemas of both relations.

The natural join will automatically perform a projection that removes from the joined table the extra copy of the attribute(s) that are shared by both relations.

3. Consider relation R(A, B, C) with n tuples, and relation S(C,D,E) with m tuples. Give the schema (list of attributes) and the minimum and maximum number of tuples to the following expressions:
   a. R x S
      This is the Cartesian product operation. The new schema (list of attributes) should be (A,B,R.C,S.C,D,E). It will be of size m x n (each tuple from relation R is joined with each tuple from relation S).

   b. R ∪ S

      This is the union operation. Since the relations have different attribute sets, this would be an illegal operation since they are not *union-compatible.*

   c. π_{ABD} (σ_C (R x S)) for some condition C

      This is a Cartesian product followed by selection on a condition C (so effectively a theta join), followed by a projection to three attributes. The new schema will be (A,B,D). The size (number of tuples) could be anywhere from *0* (if no tuples in the Cartesian product satisfied C) to *m x n* (if all tuples of the Cartesian product satisfied C).

4. Provide an example of a relation that will exhibit ‘deletion anomaly’ and discuss why a deletion anomaly is problematic.

   A relation that has a deletion anomaly follows. Assume that the tuples provided are all of the tuple. There are three employees working at two branches of a business. There is no separate Branch relation.

staffNo	staffName	branchNo	branchName	branch address
1	John Smith	1	Winston-Salem	101 E 1st Street
2	Jane Doe	1	Winston-Salem	101 E 1st Street
3	Jacob Dylan	2	Greensboro	100 Elm Street

   Removal of staff member #3 (Jacob Dylan) leads to loss of information about the Greensboro branch. Deletion anomalies in general are problematic as they represent a potential source of data loss. By not having a separate copy of the “branch information”, removal of a a staff member can lead to loss of all information for the branch, even if the branch itself should remain viable as an entity.

5. Excited by what you are learning in CSC 321/621, you decide to create a database to track the songs your favorite band plays in its live concerts. Since you decide

that E/R diagrams are for kids, you decide to create a relation schema directly for your database. After much consideration, you believe that a single schema will serve: Concerts(City, Venue, Year, Month, Date, Song, Album).

In this relation, City (e.g., “Winston Salem”) and Venue (e.g., “Ziggys”) record where the concert took place and Year, Month, and Date keep track of when the concert took place. The idea is that these five attributes uniquely specify a concert. The attribute Song records the name of a song performed at a concert. You add the attribute Album to record which album the song belongs to. Perfect!

However, after using the database for a few months, you realise that your band (and the real world) have some characteristics that you should model in your database. Convert each of the next four sentences about Concerts into a functional dependency. You can use the first letter of each attribute as an abbreviation for the attribute. Consider each of these four sentences independently. If you cannot write down a functional dependency, say so, and explain why you cannot, if possible. Do not assume any other constraints, even if they seem reasonable to you.

a. Each song appears in at most one album. In other words, the band does not repeat the same song in different albums.

   Song → Album

b. A city does not have two venues with the same name. In other words, City and Venue serve to identify the location of a concert uniquely.

   There is not a functional dependency here. Neither City → Venue (that only venue per city is every played) or Venue → City (that venues in different cities don’t have the same name) hold from this statement.

c. In an effort to please its fans, the band plays at most one song from any album in a given concert.

   City, Venue, Year, Month, Date, Album → Song

d. The manager books the band in any city at most once every year.

   City, Year → Venue
   City, Year → Month
   City, Year → Date

For the next two parts of this question, assume that all the functional dependencies you specified in the previous parts hold in Concerts, as do any dependencies that follow from them. However, no other dependencies hold in Concerts.

e. What are candidate keys for Concerts?

   Given (d), City, Year are enough to identify a particular concert. The tuple really holds songs from albums at the concerts. Since Song –> Album, we can just add Song to City, Year and get a unique identifier. So, {City, Year, Song}. Since they play just one song per album at a concert, {City, Year, Album} also works. No other candidate keys exist.

f. You realize that it is probably better, in a data sense, that you decompose Concerts into multiple relations. Here is a possible decomposition into two relations:

   Concerts1(City, Venue, Year, Month, Date)
   Concerts2(City, Year, Song, Album)

   For each relation Concerts1 and Concerts2, indicate whether they are in 3NF, 2NF, or (neither 3NF or 2NF). Use the generalized definitions of 2NF and 3NF.

   We should look at each relation independently. We should first consider candidate keys:

   > Concerts1: {City, Year} is the only candidate key.
   > Concerts 2: {City, Year, Song} and {City, Year, Album} are both candidate keys.

   Checking 2NF: For any non-candidate key attributes, are there partial dependencies on any candidate key?

   > Concerts1 : No, Venue requires knowing City and Year, as does Month and Date. So, Concerts1 is in 2NF.
   > Concerts2: There are non-candidate-key attributes (all attributes are involved with candidate keys), so it is automatically in 2NF.

   Checking 3NF: For any non-candidate key attributes, are there transitive dependencies on any candidate key?

   > Concerts1 : There do not exist any transitive dependencies with Venue, Month, Date. So, Concerts1 is in 3NF.

Concerts2: There are non-candidate-key attributes (all attributes are involved with candidate keys), so it is automatically in 3NF.