Exam II
Computer Programming 420
Dr. St. John
Lehman College
City University of New York
20 November 2001

Exam Rules

- **Show all your work. Your grade will be based on the work shown.**
- **The exam is closed book and closed notes.**
- **When taking the exam, you may have with you pens or pencils, and an 8 1/2" x 11" piece of paper filled with notes, programs, etc.**
- **You may not use a computer or calculator.**
- **All books and bags must be left at the front of the classroom during this exam.**
- **Do not open the exam until instructed to do so.**

1. True or False:

   (a) **F** Once created, database tables and schemas cannot be modified.

   (b) **F** You cannot embed a query inside another query (ie a subquery) in SQL.

   (c) **T** A superkey for a relation is a set of attributes that functionally determine all the attributes of the relation.

   (d) **T** SQL regards relations as bags of tuples, not sets of tuples.

   (e) **T** Every set is a bag.

   (f) **T** Every functional dependency is a multi-valued dependency.

   (g) **T** A view is a definition of how one relation (the view) may be constructed from tables stored in the database.

   (h) **T** Views can be queried as if they were tables.

   (i) **F** In SQL, the declarations `UNIQUE` and `PRIMARY KEY` have the same effect.

   (j) **F** In SQL, the expression, `(NULL AND TRUE) OR FALSE` evaluates to `FALSE`.

2. Consider the following relational schema:

```
Name(ID, name)   // ID is a key
GPA(ID, gpa)     // ID is a key
```

   (a) Write a **relational algebra** expression to find all students and their GPA. (That is, your answer should be a relation, with two attributes, one for the student name and one for the GPA).

   $R := \pi_{name,gpa}(\texttt{Name} \bowtie \texttt{GPA})$;

(b) Write a **relational algebra** expression to find the names of all students with the highest GPA in the database:

From Prof. Widom's Spring 2000 exam:

The idea is to make tuples (Name, gpa, N, G) where gpa $< G$-- that is, the second person in every tuple has the higher GPA (see R_3 below). Then, find all the names that occur only in the second position-- this means that no other GPA is higher (see R_6):

$$
\begin{aligned}
R_1 &:= \rho_{R(N,G)}(R); \\
R_2 &:= R \times R_1; \\
R_3 &:= \sigma_{\texttt{gpa}<G}(R_2); \\
R_4 &:= \rho_{R(\texttt{name})}(\phi_N(R_3)); \\
R_5 &:= \phi_{\texttt{name}}(R_3); \\
R_6 &:= R_5 - R_4;
\end{aligned}
$$

This part is hard, and credit was given for solutions that used an aggregation operator (e.g. MAX).

3. Write a java program that prints "Hello, world" to the screen:

From Lab 4:

```
public class Hello
{  public static void main(String[] args)
   {  String greeting = "Hello, World!\n";
      System.out.print(greeting);
   }
}
```

The common mistakes were leaving off the class definition (c-style program) or forgetting the main method.

4. Suppose we have a relation $R(A, B, C, D, E)$ with the following functional dependencies: $AB \rightarrow C$, $CD \rightarrow E$, $C \rightarrow A$, and $C \rightarrow D$.

(a) What are all the keys for R?

From Ullman's Fall 1999 exam:

AB and BC, since the closure of each yields all attributes, and no smaller set does. Note that any key must include B since it never occurs on the right hand side of any dependency. Looking at the closures of all sets of 2 attributes that include B, we have $AB^+ = ABCDE$, $BC^+ = ABCDE$, $BD^+ = BD$, $BE^+ = BE$. You can check that the sets of 3 attributes that include B and are superkeys, also include the keys either AB or BC.

(b) Give an example of a functional dependency that is a BCNF violation for R:

(c) Into what two relations does this violation tell us to decompose R?

There were three possible answers for these parts. Any of them were fine:

CD->E: R1(C,D,E), R2(A,B,C,D)

C->A: R1(A,C), R2(B,C,D,E)

E->D: R1(D,E), R2(A,B,C,E)

Note that $AB \to C$ is not a BCNF violation since $AB^+ = ABCDE$.

To decompose the relations, we follow the algorithm from §3.7.4 Decomposition into BCNF. The idea is that the attributes on the left hand side of the BCNF violation will appear in both of the new relations. One relation will also have the attributes that follow from the dependency. The other will have the rest of the attributes.

5. Suppose R and S are relations.

   (a) Suppose relations R and S have 1 tuple and 2 tuples, respectively.

   What is the minimum number of $R \cup S$ could have, under the bag semantics?

   The bag will contain all elements of R and S (since duplicates are not eliminated). So, the answer is 3.

   What is the minimum number of $R \cup S$ could have, under set semantics?

   Since duplicates are eliminated in sets, we could have that all the elements in R are contained in S. So, the answer is 2.

   (b) Suppose relations R and S have 2 tuples and 3 tuples, respectively.

   What is the minimum number of $R \cup S$ could have, under the bag semantics?

   The bag will contain all elements of R and S (since duplicates are not eliminated). So, the answer is 5.

   What is the minimum number of $R \cup S$ could have, under set semantics?

   Since duplicates are eliminated in sets, we could have that all the elements in R are contained in S. So, the answer is 3.

   (c) Suppose relations R and S have n tuples and m tuples,

   What is the minimum number of $R \cup S$ could have, under the bag semantics?

   The bag will contain all elements of R and S (since duplicates are not eliminated). So, the answer is $|R| + |S| = m + n$.

   What is the minimum number of $R \cup S$ could have, under set semantics?

   Since duplicates are eliminated in sets, we could have that all the elements in R are contained in S, or vice versa. So, the answer is $\max(|S|, |R|) = \max(m, n)$.

   (In the above examples, R has less elements than S, so, if you use that assumption, the answer is n).

6. Answer the questions below based on the following schema:

```
companies(co_id, co_name, co_postcode, co_lastchg);
products(pr_code, pr_desc);
orders(ord_id, ord_company, ord_product, ord_qty, ord_placed,
       ord_delivered, ord_paid);
diary(dy_id, dy_company, dy_timestamp, dy_type, dy_notes);
```

All of these are exercises from Lab.

   (a) Write a query that returns the product codes contained in the database:

```
SELECT pr_code FROM products;
```

(b) Write a query that returns the product codes and the average number ordered of each per order:

```
SELECT pr_code, avg(ord_qty)
FROM products,orders
WHERE pr_code = ord_product
GROUP BY pr_code;
```

(c) Create a view that contains the name of each company and the total number of orders placed for that company:

```
CREATE VIEW CompanyOrderTotals AS
SELECT co_name, SUM(ord_qty)
FROM products,orders
WHERE co_id = ord_company
GROUP BY co_id;
```

(d) Create an index on `ord_company`:

```
CREATE INDEX order_co_index ON orders(ord_company);
```

(e) Write a query that select all orders that were placed in a different month from when the product was delivered. For example, the order is placed on 06-29-2001, and the product is delivered on 07-06-2001. Include in the output, the order ID, the product code, the date the order was placed and the date is was delivered:

```
SELECT *
FROM orders
WHERE date_part('month', ord_placed) <> date_part('month', ord_delivered)
```

7. Given two relations R and S:

(a) Give the definition of the natural join $R \bowtie S$:

Let $A_1, A_2, \ldots, A_m$ be the attributes that occur in R, but not S.
Let $B_1, B_2, \ldots, B_n$ be the attributes that occur in both R and S.
Let $C_1, C_2, \ldots, C_l$ be the attributes that occur in S, but not R.

Then, $R \bowtie S$ consists of all tuples t (on attributes

$$A_1, A_2, \ldots, A_m, B_1, B_2, \ldots, B_n, C_1, C_2, \ldots, C_l)$$

that are made by joining tuples from R and S that agree on $B_1, B_2, \ldots, B_n$. Note that the new joined tuples t have only one copy of the attributes $B_1, B_2, \ldots, B_n$.

(b) Give the definition of the theta-join $R \bowtie_C S$:

The theta-join $R \bowtie_C S$ is constructed by first taking the cross product of R and S, $R \times S$, and then selecting the tuples for which condition C is true.

Note that the theta-join $R \bowtie_C S$ has attributes:

$$A_1, A_2, \ldots, A_m, R.B_1, R.B_2, \ldots, R.B_n, S.B_1, S.B_2, \ldots, S.B_n, C_1, C_2, \ldots, C_l$$

(using the names of the attibutes for Part a))

(c) What is the difference between $R \bowtie S$ and $R \bowtie_C S$ where the condition C is that $R.A = S.A$ for each attribute A appearing in the schemas of both R and S?

From the practice problems of the homework.

Each will have the same number of tuples, but $R \bowtie_C S$ will have two copies of the attributes the relations have in common. The natural join $R \bowtie S$ will consist of all the joined tuples of R and S, with common attributes collapsed. $R \bowtie_C S$, with the condition given, will be the cross product of R and S with all common attributes identified, but two copies of each common attribute (e.g., $R.A$ and $S.A$).

8. (a) Rewrite the following SQL query **without** using the INTERSECT or DIFFERENCE operators:

```
(SELECT name, address FROM MovieStar WHERE gender = 'F')
        INTERSECT
(SELECT name, address FROM MovieExec WHERE netWorth > 10000000);
```

From the practice problems on the homework (5.3.5a, p 279).

There are many ways to solve this one. One possible answer is:

```
SELECT MovieStar.name, MovieStar.address
FROM MovieStar, MovieExec
WHERE MovieStar.name = MovieExec.name AND
    MovieStar.address = MovieExec.address AND
    gender = 'F' AND
    netWorth > 10000000;
```

(b) Show how to express the relational-algebra query

$$\pi_L(\sigma_C(R_1 \times R_2)))$$

in SQL, where L is a list of attributes and C is an arbitrary condition:

From the practice problems on the homework (5.2.4, p 263).

```
SELECT L
FROM R_1, R_2
WHERE C;
```