Answer Key for the Final Examination
Computer Science 420 (Dr. St. John)
Lehman College– CUNY
18 December 2001

1. True or False:

(a) T Every key is a superkey.
(b) F Bags are used in theory but not in practical query languages like SQL.
(c) F In the relational algebra, values appearing in one column of a relation can be summarized (aggregated) by using one of the keywords SUM, AVG, MIN, MAX, or COUNT.
(d) T Views can be queried as if they were tables.
(e) F In SQL, the expression, (NULL OR TRUE) OR FALSE evaluates to UNKNOWN.
(f) T The relational algebra expression $R(a, b) \cup S(a, b)$ is represented by the datalog rules:

$$\text{Answer}(x,y) \leftarrow R(x,y);$$
$$\text{Answer}(x,y) \leftarrow S(x,y);$$

(g) T In Datalog, negation is forbidden inside recursive rules.
(h) F Third Normal Form (3NF) is stronger than Boyce Codd Normal Form (BCNF).
(i) F Transactions are read-only statements executed in SQL.
(j) T A dirty read is a common term for data that has been written by a transaction that has not yet been committed.

2. (a) Write the relational database schema that corresponds to the following E/R diagram. Mark any keys in your schema.

Answer: \[
\text{Logins}(\text{userName, hostName, quota})
\]
\[
\text{Hosts}(\text{hostName, model})
\]
(b) What are the non-trivial functional dependencies?

Answer:

userName, hostName $\rightarrow$ quota
hostName $\rightarrow$ model

3. Consider the following three ODL schemas:

- **Schema1:**

  ```
  interface Book {
    attribute string title;
    relationship Set<Author> author
    inverse Author::book; }
  
  interface Author {
    attribute string name;
  }
  
  relationship Set<Book> book
  ```

- **Schema2:**

  ```
  interface Book {
    attribute string title;
    relationship Set<Author> author
    inverse Author::book; }
  
  interface Author {
    attribute string name;
  }
  
  relationship Book book
  ```

- **Schema3:**

  ```
  interface Book {
    attribute string title;
    relationship Author author
    inverse Author::book; }
  
  interface Author {
    attribute string name;
  }
  
  relationship Book book
  ```

(a) **Schema1** is most appropriate for data containing (circle one):

- One-one relationships between books and authors
- Many-one relationships between books to authors
- One-many relationships between books to authors
- **Many-many relationships between books and authors**

(b) **Schema2** is most appropriate for data containing (circle one):

- One-one relationships between books and authors
- Many-one relationships between books to authors
- **One-many relationships between books to authors**
- Many-many relationships between books and authors

(c) **Schema3** is most appropriate for data containing (circle one):

- **One-one relationships between books and authors**
- Many-one relationships between books to authors
- One-many relationships between books to authors
- Many-many relationships between books and authors

4. (a) Suppose we have a relation $R(A, B, C, D)$ with a functional dependencies: $AB \rightarrow C, C \rightarrow D, D \rightarrow A$. What is a set of keys for $R$?
(b) Suppose we have a relation \( S(A,B,C) \) with a multivalued dependency \( A \rightarrow \rightarrow B \). If we know that the tuples \((a,b_1,c_1)\) and \((a,b_2,c_2)\) belong to \( S \), what other tuples **must** also be in \( S \)?

**Answer:**

\[
(a, b_2, c_1) \\
(a, b_1, c_2)
\]

5. For each pair of queries below, write “YES” in the third column if they are equivalent and “NO” if they are not equivalent. Remember that two queries are equivalent if they always return exactly the same answer on all databases.

All queries refer to a schema containing two relations:

\[ R(A,B) \text{ where } A \text{ is a key and } B \text{ is a key} \]
\[ S(A,B) \text{ where } A \text{ is a key} \]

You may assume that the relations do not contain **NULL** values but do not make any other assumptions about the relations.

<table>
<thead>
<tr>
<th>Query 1</th>
<th>Query 2</th>
<th>Equivalent?</th>
</tr>
</thead>
<tbody>
<tr>
<td>select A from R</td>
<td>( \pi_A(R) )</td>
<td>YES</td>
</tr>
<tr>
<td>select A from R</td>
<td>select A from R group by A</td>
<td>YES</td>
</tr>
<tr>
<td>select B from S</td>
<td>select B from S group by B</td>
<td>NO</td>
</tr>
<tr>
<td>select B from S</td>
<td>select distinct B from S</td>
<td>NO</td>
</tr>
<tr>
<td>select R.B from R,S where R.A=S.A</td>
<td>select S.B from R,S where R.A=S.A</td>
<td>NO</td>
</tr>
<tr>
<td>select R.B from R,S where R.A=S.A</td>
<td>select B from R where A in (select A from S)</td>
<td>YES</td>
</tr>
<tr>
<td>(select B from S) except all (select B from R)</td>
<td>select B from S where B not in (select B from R)</td>
<td>YES</td>
</tr>
<tr>
<td>( \pi_A(R - S) )</td>
<td>( \pi_A(R) - \pi_A(S) )</td>
<td>YES</td>
</tr>
<tr>
<td>( R \cap S )</td>
<td>( R \bowtie S )</td>
<td>YES</td>
</tr>
<tr>
<td>( \pi_{R,A=5}(R) )</td>
<td>( \sigma_{R,A=5}(\pi_{R,A,B}(R \times S)) )</td>
<td>NO</td>
</tr>
</tbody>
</table>

6. Using the database schema:

Product(maker, model, type)
PC(model, speed, ram, hd, cd, type, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
(a) Write the statements that declares that the model in PC must occur in the Product table. Modifications that violate this constraint are rejected.

In the relational algebra:

\[ \pi_{\text{model}}(PC) \subseteq \pi_{\text{model}}(\text{Product}) \]

In SQL, there's several ways to do this, including adding a check statement:

```
CHECK (new.model IN (SELECT model FROM product))
```

(b) Write the code that guarantees that: A PC with a processor speed of less than 500 must not sell for more than $1500.

In the relational algebra:

\[ \pi_{\text{speed} < 500} \text{ AND } \pi_{\text{price} >= 1500}(PC) = \emptyset \]

In SQL, there's several ways to do this, including adding a check statement:

```
CHECK ((new.speed >= 500) or (new.price < 1500))
```

7. Suppose we have a relation in which we want to record for each person their name, Social Security number, and birthdate. Also, for each child of the person, the name, Social Security number, and the birthdate of the child, and for each automobile the person owns, its serial number and make. To be more precise, this relation has all tuples:

\[ (n, s, b, cn, cs, cb, as, am) \]

where \( n \) is the name of person with Social Security number \( s \), \( b \) is \( n \)'s birthday, \( cn \) is the name of one of \( n \)'s children, \( cs \) is the Social Security number of \( cn \), \( cb \) is the birthdate of \( cs \), \( as \) is the serial number of one of \( n \)'s automobiles, and \( am \) is the make of the automobile with serial number \( am \).

(a) What functional dependencies you would expect to hold:

\[
\begin{align*}
    s & \rightarrow n, b \\
    cs & \rightarrow cn, cb \\
    as & \rightarrow am \\
    as & \rightarrow s \text{ (if you assume one owner per car)} \\
    cs & \rightarrow s \text{ (if you assume one parent per child)}
\end{align*}
\]

(b) What multivalued dependencies you would expect to hold:

\[
\begin{align*}
    s & \rightarrow\rightarrow cn, cs, cb \\
    s & \rightarrow\rightarrow as, am
\end{align*}
\]
(c) Suggest a decomposition of the relation into Fourth Normal Form (4NF):
\[ R_1(s, cs, as), R_2(s, n, b), R_3(cs, cn, cb), R_4(as, m) \]

8. Assume you have two tables:

- **Movies** *(title, year, length, studioName)* where title and studioName are strings, and year and length are integers.
- **Stars** *(title, year, starName)* where title and starName are strings, and year is an integer.

Write the SQL statements that:

(a) Print out each star's name and the length of the longest movie they starred in:

```sql
SELECT starName, MAX(length)
FROM Movies, Stars
WHERE Movies.title = Stars.title
AND Stars.year = Movies.year
GROUP BY starName;
```

(b) Grant insert privileges to everyone for table Movies:

```sql
GRANT INSERT ON Movies TO ALL;
```

(c) Grant update privileges to the user movieFan for the attribute starName:

```sql
CREATE VIEW NamesOfStars
AS SELECT StarName
FROM Stars;
GRANT UPDATE ON NamesOfStars TO movieFan;
```

9. Assume your database has a single relation, \( \text{Par}(x,p) \) which holds if and only if \( p \) is a parent of \( x \).

(a) Write a datalog program \( \text{Sib}(x,y) \) which holds if and only if \( x \) and \( y \) are siblings.

\[
\text{Sib}(x,y) \leftarrow \text{Par}(x,p) \text{ AND } \text{Par}(y,p);
\]

(b) Write a datalog program \( \text{Cousin}(x,y) \) which holds if and only if \( x \) and \( y \) are cousins.
(\( \text{The example done in class assumes that siblings are cousins. If you don't assume that, then take the answer below, and subtract off the siblings.} \) )

\[
\text{Cousin}(x,y) \leftarrow \text{Sib}(x,y);
\text{Cousin}(x,y) \leftarrow \text{Par}(x, xp) \text{ AND } \text{Par}(y, yp) \text{ and Cousin}(xp, yp);
\]
(c) Are Sib(x,y) and Cousin(x,y) equivalent? If so, explain why. If not, give a database instance where the relations differ.

No, they are not equivalent. For example, consider the database with the table Par:

<table>
<thead>
<tr>
<th>x</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>George</td>
</tr>
<tr>
<td>Jeb</td>
<td>George</td>
</tr>
<tr>
<td>George P</td>
<td>Jeb</td>
</tr>
<tr>
<td>Jenna</td>
<td>W</td>
</tr>
<tr>
<td>Barbara</td>
<td>W</td>
</tr>
</tbody>
</table>

Jenna and George P are cousins, but not siblings.

10. Assume that a database called testdb has been created on your system. Write a complete Java program that creates a table called Courses, which has three attributes:

- the courseId, a unique number that identifies the course,
- the dept, the name of the department, and
- the number, the course number.

Your program should also insert into the table the following three tuples:

<table>
<thead>
<tr>
<th>courseId</th>
<th>dept</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0099</td>
<td>BIO</td>
<td>167</td>
</tr>
<tr>
<td>0457</td>
<td>GEO</td>
<td>101</td>
</tr>
<tr>
<td>1858</td>
<td>CMP</td>
<td>420</td>
</tr>
</tbody>
</table>

```java
import java.sql.*;
public class Lab6 {
    public static void main(String[] args) {
        try {
            Class.forName("postgresql.Driver");
        } catch(ClassNotFoundException e) {
            System.out.println(e);
            System.exit(1);
        }
        try {
            Connection db = DriverManager.getConnection(
                "jdbc:postgresql:testdb", "ks27", "pass");
            Statement st = db.createStatement();
            st.executeUpdate("create table Courses(courseId int4 primary key, dept varchar(20), number int4);");
            st.executeUpdate("insert into Courses values(99, 'BIO', 167)");
            st.executeUpdate("insert into Courses values(457, 'GEO', 101)");
            st.executeUpdate("insert into Courses values(1858, 'CMP', 420)");
            ResultSet rs = st.executeQuery("select * from t");
            System.out.println("Company Name");
            while(rs.next()) {
```
System.out.println(rs.getString(1)); }
rs.close();
st.close();
}
catch(SQLException e) {
    System.out.println(e);
    System.exit(1);
}
}