CMPS 182, Final Exam, Winter 2015, Shel Finkelstein

Student Name: _____

Student ID: _____

UCSC Email: _____

Final Points

Part	Max Points	Points
I	30	
II	24	
III	18	
IV	36	
Total	108	

Part I: (30 points, 5 each) Assume that there are relations with the following schemas. Assume that attributes can't be NULL. Underlined attributes are keys. For each question in this part, write a SQL statement that does what's requested by the question.

Student(<u>studentID</u>, student_name, major)
For each studentID, gives the students name and major

Course(<u>courseID</u>, course_name, department, instructor)
For each course, gives the course's description and department

Enroll(<u>studentID, courseID</u>, grade)
For each studentID and courseID where the student taken that course, gives gives the sudent's grade, which is a float number.

If there is a row for a studentId and courseID in Enroll, then there always is a row in Student for that studentID, and in Course for that courseID.

Question 1: Find the studentId and student_name for all students whose major is 'CMPS', ordered by student_name.

Answer 1:

```
SELECT studentID, student_name
FROM Student
WHERE major='CMPS'
ORDER BY student_name;
```

Question 2: For each course that has students enrolled, find the average grade that students received in that course, listing the courseID for the course and the average grade.

Answer 2:

```
SELECT courseID, AVG(grade)
FROM Enroll
GROUP BY courseID;
```

Question 3: For each course, list student_name of any student who received the highest grade of any student enrolled in that course, together with the course_name.

Answer 3:

```
SELECT student_name, course_name
FROM Student s, Course c, Enroll e1
WHERE s.studentID = e1.studentID
AND c.courseID = e2.courseID
AND e1.grade >= ALL
    ( SELECT e2.grade
      FROM Enroll e2
       WHERE e2.courseID = e1.courseID );
```

This can also be written using MAX, as e1.grade >= MAX (subquery above), or using NOT EXISTS, saying that there doesn't exist a large grade in Enroll for the class. Any way that's correct is okay.

Question 4: List the courseID for each course in which some enrolled student has a grade of exactly 4.0. List each courseID at most once, even if multiple students received 4.0 grades.

Answer 4:

```
SELECT DISTINCT(courseID)
FROM Enroll
WHERE Enroll.grade = 4.0;
```

Question 5: Delete students from the Student relation if they aren't enrolled in any courses.

```
DELETE
FROM Student
WHERE NOT EXISTS
   ( SELECT *
     FROM Enroll
      WHERE Student.studentID = Enroll. studentID );
```

Question 6: For each courseID that has at least 10 students enrolled, list the courseID, course_name, department, and instructor.

Answer 6: Here are two different correct answers.

```
SELECT courseID, course_name, department, instructor
FROM Course
WHERE
  ( SELECT COUNT(*)
    FROM Enroll
    WHERE Enroll.courseID = Course.courseID )    >= 10;
```

```
SELECT courseID, course_name, department, instructor
FROM Course
WHERE Course.courseID IN
  ( SELECT courseID
    FROM Enroll
    WHERE Course.courseID = Enroll.courseID
    GROUP BY courseID
    HAVING COUNT(*)  >= 10 );
```

Part II: (24 Points, 4 each)

Question 7: If R(A,B) is a relation where A's domain is (a1, a2, a3, a4) and A can also be null, and B's domain is (b1, b2, b3, b4,b5,b6), and B can also be NULL, what the maximum number of different tuples that can be in R?

Answer 7: 35, which is 5*7, counting NULL values.

Question 8: What words do the four letters in ACID stand for in transaction processing?

Answer 8:

Atomicity
Consistency
Isolation
Durability

Question 9: What is an advantage of using Stored Procedures?

Answer 9: Some examples of good answers:

- Processing of a stored procedure can be done in the database, performing operations that may be hard or impossible to do in a single SQL statement, without having to do multiple calls to the database.

- By filtering data using a stored procedure, less data has to be moved from the database to the client application.

- A client may be authorized to execute a stored procedure and see the result, without being authorized to see the data that the stored procedure uses (or modifies).

Question 10: What is an advantage of XML over relational models?

Answer 10: Some examples of good answers:

- XML can represent semi-structured or unstructured data, where you don't know (or don't fully know) what the structure/schema of the data is before you see the data. It is sometimes described as "data first", whereas relational is described as "schema first".

- XML also allows data that isn't in First-Normal Form (that is, with non-primitive types), so a value may be a complex type (such as a set, bag, sequence, array, …) which relational doesn't permit.

- XML is similar to hierarchical data models, encouraging access and storage of data based on nesting of data within XML documents, which can be efficient for storage and access.

Question 11: In using SQL embedded in a programming language, you can execute a statement directly using:

EXEC SQL <text-of-query>;

You can also use dynamic SQL, with:

EXEC SQL PREPARE <query-name>
FROM <text of the query>;

And after that you can execute:

EXEC SQL EXECUTE <query-name>;

What are differences between the direct and dynamic use of SQL? When might dynamic SQL be needed?

Answer 11:

With direct use of SQL, you need to include a specific statement as part of the program, so you must know that specific statement when you write the program. With dynamic SQL, you can construct the statement at runtime, prepare it and execute.

Moreover, with dynamic SQL, you can prepare the statement and execute as often as you want from different parts of the program, using query-name to reference the statement.
You might use dynamic SQL when you don't know beforehand what the statement to be executed is, e.g., if a client is entering SQL statements for you to execute, whether they are entered as text of the SQL statement, or through a visual interface.

Question 12: Assume that relation R(A,B,C,D) includes the row (1,2,3,4), and that R has Functional Dependencies A,B → C and C → D,

Mark each row below with a check if it might also be in R, and mark each row below with an X if it can not also be in R.

Answer 12:

___✖___ (1,2,4,7)

___✔___ (1,3,8,9)

___✖___ (1,2,3,5)

___✔___ (2,2,4,6)

Part III: (18 points, 3 each)

Answer the following questions with **YES** or **NO**.

Question 13: Is the following a legal SQL query for a table
Staff(name, age, salary, department)?

```
SELECT MAX(age), department
FROM Staff
WHERE salary > 5000
GROUP BY department
HAVING MIN(age) > 21;
```

Answer 13: **YES**. Both MAX(age) and MIN(age) are defined for each department group, and of course, so is department.

Question 14: Is the following equality always true when R is a relation, *attribs* is some attributes of R, and *cond* is a condition on R that refers only to the attributes of R that are in *attribs*?

$\sigma_{cond}\ (\pi_{attribs}\ (R)\ ) = \pi_{attribs}\ (\sigma_{cond}\ (R)\ )$?

Answer 14: **YES**. If *cond* could refer to attributes that aren't in *attribs*, then they wouldn't be equivalent, since attributes necessary for the selection would be there after the projection.

Question 15: Is the following relation with the specified Functional Dependencies in **Third Normal Form**?

R(<u>city, street</u>, zip)
city,street → zip
zip → city

Answer 15: **YES**, because (zip, street) is a key, as is (city,street)

Question 16: Is the following relation with the specified Functional Dependencies in **Boyce-Codd Normal Form**? (Same relation and FDs as in question 15.)

R(city, street, zip)
city,street → zip
zip → city

Answer 16: **NO**. The FDs aren't trivial, and although (city,street) is a superkey, zip is not a superkey.

Question 17: For OLAP, if you have a Fact Table with associated Dimension Tables, must the dimension attributes of each row in the Fact Table correspond to values of keys in the Dimension Tables?

Answer 17: **YES**. The dimension attributes in the Fact Table are Foreign Keys referencing the keys of the dimension tables.

Question 18: If the following is part of an XML document:

```
<BAR><ADDR>101 Maple St.</ADDR>
  <PHONE>555-1212</PHONE>
  <PHONE>555-4567</PHONE>
</BAR>
```

Then could:

<!ELEMENT BAR (ADDR PHONE* MANAGER?)>

be the declaration for BAR in a DTD?

Answer 18: **YES**. There's an ADDR, 0 or more PHONE's, and 0 or 1 MANAGER.

Part IV: (30 points, 6 each):

Question 19: Write a SQL statement that creates a table Employee with four attributes , empNum, name, deptNum and salary, where empNum, deptNum and salary are integers, name is a character string of length 30, empNum is the primary key, and deptNum and salary can be NULL (but other attributes can't be).

Answer 19:

```
CREATE TABLE Employee (
        empNum          INT PRIMARY KEY,
        name            CHAR(30) NOT NULL,
        deptNum         INT,
        salary          INT
    );
```

There can be a PRIMARY KEY specification at the end of the CREATE statement, instead of putting it with empNum, and that's okay.

Question 20: Write a SQL INSERT statement that inserts a tuple into the Employee table (described in question 19) for an employee named John Smith who is in department (deptNum) 56, has salary 25000, and whose employee number (empNum) is 789.

Also, write a SQL DELETE statement that deletes just that tuple.

Answer 20:

```
INSERT INTO Employee VALUES (789,  'John Smith', 56, 25000);

DELETE  FROM Employee
WHERE empNum = 789;
```

For the DELETE, it's okay to specify additional values besides empNum, as long as they're right. But empNum is key, so only empNum is needed.

Question 21: Write a SQL UPDATE statement that increases salary by 100 for all employees in the Employee table (described in question 19) whose salary is less than 8000.

Also, for your UPDATE, what happens to employees whose salary is NULL?

```
UPDATE Employee
SET salary = salary + 100
WHERE salary < 8000;
```

For an employee whose salary is NULL, the predicate "salary < 8000" will be false, so nothing will happen for such an employee.

Question 22: How would you change the CREATE statement for question 19 if we wanted to enforce Referential Integrity by making deptNum in the Employee table be a foreign key referring to a deptNum attribute in a Department table?

Also, describe two (out of the three) different actions that might be taken if a department that had employees in it was deleted from the Department table.

Answer 22:

```
CREATE TABLE Employee (
        empNum          INT PRIMARY KEY,
        name            CHAR(30) NOT NULL,
        deptNum         INT REFERENCES Department(deptNum) ,
        salary          INT
    );
```

[It's also possible to do this by adding a FOREIGN KEY specification at the end of the CREATE statement, e.g., FOREIGN KEY (deptNum) REFERENCES Department(deptNum), but we didn't discuss that in class.]

The three actions that may be taken if a department that had employees in it were deleted from the Department table are:

- Default action: Reject the deletion
- Cascade: Delete employees in the department (matching deptNum) when the department is deleted.
- Set NULL: For all employees who were in the deleted department (matching deptNum), set deptNum to NULL.

Question 23: Assume that a JDBC connection myCon has been established to a database that has a table Sells(bar, beername, price), where bar and beername are character strings and price is float.

Print out all the beernames and prices that are sold at 'GoodBar'. Don't bother with including libraries or variables declarations, and you can have an informal print statement if you want.

Here 's an outline of what you need to write:

```
// Execute the query
// Loop through the results
// For each value in the result, get the values of beername and price, and print them
```

Answer 23:

```
// Execute the query
Statement stmt = myCon.createStatement();
resultset = stmt.executeQuery("SELECT beername, price FROM Sells WHERE bar
LIKE 'GoodBar'");

// Loop through the results
while (resultset.next()) {
   // For each value in result, get values of beername and price, and print them
   System.out.println(resultset.getString(1), resultset.getFloat(2));
}
```

Question 24: Normal forms help avoid anomalies. We discussed 3 anomalies (Update, Delete and Insert) in class. Explain 2 of these 3 anomalies, giving examples of the problems that could arise.

You may use this table Employee(eid, name, addr, rank, salary-scale), with the Functional Dependency rank → salary-scale, to discuss the anomalies, if you want.

eid	name	addr	rank	salary-scale
34-133	Jane	Elm St.	6	70-90
33-112	Hugh	Pine St.	3	30-40
26-002	Gary	Elm St.	4	35-50
51-994	Ann	South St.	4	35-50
45-990	Jim	Main St.	6	70-90
98-762	Paul	Walnut St.	4	35-50

Answer 24:

- Update anomaly: If Ann becomes rank 6, and her salary-scale doesn't also become 70-90, then the table is inconsistent, because the FD is violated.
- Delete anomaly: If Hugh is deleted, then the information that rank 3 is associated with salary scale 30-40 is lost.
- Insert anomaly: If Mary is entered with rank 6 but her salary-scale isn't set to 70-90, then the table is inconsistent, because the FD is violated.