Final Examination

- Please read all instructions (including these) carefully.
- There are ten problems on the exam, with a varying number of points for each problem and subproblem for a total of 100 points. You have 100 minutes to complete the exam, along with a bonus of 20 extra minutes to check your work. You should look through the entire exam before getting started, in order to plan your strategy.
- The exam is closed book and closed notes, but you may refer to your six pages of prepared notes.
- Please write your solutions in the spaces provided on the exam. Make sure your solutions are neat and clearly marked. You may use the backs of the exam pages as scratch paper along with the scratch space provided. Please do not use any additional scratch paper.
- *Simplicity and clarity of solutions will count.* You may get as few as 0 points for a problem if your solution is far more complicated than necessary, or if we cannot understand your solution.

NAME: ______________________________

In accordance with both the letter and spirit of the Honor Code, I have neither given nor received assistance on this examination.

SIGNATURE: ______________________________

Problem	Max points	Points
1	15	
2	5	
3	15	
4	15	
5	5	
6	12	
7	20	
8	5	
9	5	
10	3	
TOTAL	100	

1. **SQL Queries** (15 points, 5 for each part)

   This problem uses the following self-explanatory relational schema.

```
Student(ID, name, dorm, GPA)  // ID is the only key
Plays(ID, sport)              // <ID,sport> is the key
```

   (a) Write a SQL query to find all dorms (no duplicates please) such that at least one student who lives in the dorm does not play any sports.

   (b) Can you write a SQL query that is equivalent to the following one but does not use the keyword DISTINCT? If so, give the query. If not, explain why not.

```
SELECT DISTINCT dorm
FROM Student
WHERE GPA > 3.9
```

(c) Can you write a SQL query that is equivalent to the following one but does not use a `HAVING` clause? If so, give the query. If not, explain why not.

```
SELECT sport
FROM Student, Plays
WHERE Student.ID = Plays.ID
GROUP BY sport
HAVING AVG(GPA) > 3.0
```

2. **Constraints and Assertions** (5 points)

Consider the following three schema declarations. You may assume that all columns have the same type, which has been omitted for brevity.

```
Schema 1: CREATE TABLE R(A PRIMARY KEY, B)
          CREATE TABLE S(C, D REFERENCES R(A))

Schema 2: CREATE TABLE R(A PRIMARY KEY, B)
          CREATE TABLE S(C, D CHECK(D IN (SELECT A FROM R)))

Schema 3: CREATE TABLE R(A PRIMARY KEY, B)
          CREATE TABLE S(C, D)
          CREATE ASSERTION Assert CHECK(
            NOT EXISTS (SELECT * FROM S
                        WHERE D NOT IN (SELECT A FROM R)))
```

(Problem continues on following page...)

Which of the following statements is true? (Please circle exactly one.)

- All three schemas are equivalent in terms of their behavior.
- Schemas 1 and 2 are equivalent but Schema 3 is different.
- Schemas 1 and 3 are equivalent but Schema 2 is different.
- Schemas 2 and 3 are equivalent but Schema 1 is different.
- None of the three schemas are equivalent.

3. **Assertions and Relational Algebra** (15 points, 3 for each part)

   Consider relations $R(A, B, C)$ and $S(A, B, C)$ with no assumptions about keys. For each of the statements in SQL or relational algebra below, write in the box provided the constraint that is specified by the given statement.

   - Please be specific about the actual attributes involved in the constraint.
   - In all cases the correct answer corresponds to a single concept from the course and can be specified in a few words or symbols. Much longer answers will be considered incorrect.

   (a) Statement:

   ```
   CREATE ASSERTION Mystery AS
      (NOT EXISTS (SELECT A FROM R
                   GROUP BY A
                   HAVING COUNT(*) > 1))
   ```

   Concept:

   (b) Statement: $\pi_A(R) \rhd\!\!< \pi_A(S) = \pi_A(R)$

   Concept:

   (c) Statement: $\sigma_{B1 \neq B2}(\rho_{R1(A,B1,C1)}(R) \bowtie \rho_{R2(A,B2,C2)}(R)) = \emptyset$

   Concept:

(d) Statement: $\pi_{A,B}(R) \bowtie \pi_{A,C}(R) = R$

Concept:

(e) Statement:

```
CREATE ASSERTION TrickyOne AS
   (NOT EXISTS ((SELECT * FROM R)
                EXCEPT
                (SELECT * FROM R WHERE A = A)))
```

Concept:

(Scratch space)

4. **Referential Integrity and Triggers** (15 points)

   Consider relations $R(A, F)$ and $S(P, B)$, where A is a key for R and P is a key for F. Suppose we wish to enforce a referential integrity constraint in which attribute $R.F$ (the foreign key) references attribute $S.P$ (the primary key). Furthermore, suppose we are using a DBMS that supports SQL3 triggers but does not support referential integrity. Write a set of triggers that enforces referential integrity between $R.F$ and $S.P$. Your triggers should implement the "`set null`" referential integrity enforcement policy, and your trigger actions may include the command "`<generate error>`" if appropriate. You should use the general SQL3 trigger syntax as covered in the textbook and in class, *not* the restricted trigger syntax supported by Oracle.

5. **Authorization** (5 points)

   Give the simplest sequence of `CREATE TABLE` and `GRANT` statements you can think of that results in a cycle in the corresponding grant diagram. For each statement indicate the name of the user issuing the statement. You need not show the grant diagram.

6. **Transactions** (12 points, 3 for each part)

   Consider a simple relation `Scores(s integer)` initially containing three tuples with values `<1>`, `<2>`, and `<3>`. Consider the following two concurrent transactions.

```
Transaction T1: update Scores set s = s + 2;
                insert into Scores (select * from Scores where s > 3);
                commit;

Transaction T2: select sum(s) from Scores;
                select sum(s) from Scores;
                commit;
```

   Assume that any interleaving between concurrent transactions is at the statement level. In other words, transaction `T2` cannot perform any read operations *within* the execution of transaction `T1`'s individual modification statements, although in some cases transaction `T2` may perform its read operations *between* the execution of transaction `T1`'s modification statements. Similarly, transaction `T1` cannot perform any modification operations *within* the execution of transaction `T2`'s individual queries, although in some cases transaction `T1` may perform its modifications *between* the execution of transaction `T2`'s queries. Assume for all parts of this problem that transaction `T1` executes with isolation level `serializable`.

   (Problem continues on following page...)

(a) If transaction `T2` executes with isolation level `serializable`, what are all the possible pairs of values selected by `T2`'s two queries? Clearly list each pair separately.

(b) If transaction `T2` executes with isolation level `repeatable read`, what are all the possible pairs of values selected by `T2`'s two queries? Clearly list each pair separately.

(c) If transaction `T2` executes with isolation level `read committed`, what are all the possible pairs of values selected by `T2`'s two queries? Clearly list each pair separately.

(d) If transaction `T2` executes with isolation level `read uncommitted`, what are all the possible pairs of values selected by `T2`'s two queries? Clearly list each pair separately.

7. **Row Types and Indexes** (20 points, 5 for each part)

   Consider the following SQL3 relational schema, where column types in `ROW TYPE` definitions are omitted for brevity.

```
CREATE ROW TYPE StudType AS (ID, name, dept)
CREATE ROW TYPE CourseType AS (num, dept, title)
CREATE ROW TYPE ProfType AS (name, dept, office)

CREATE TABLE Student OF TYPE StudType   // ID is the only key
CREATE TABLE Course OF TYPE CourseType  // num is the only key
CREATE TABLE Prof OF TYPE ProfType      // <name,dept> is the only key

CREATE TABLE Took(student REF(StudType), course REF(CourseType),
                  prof REF(ProfType), quarter STRING, grade FLOAT)
```

   (a) Write a SQL3 query to find the ID's and names (no duplicates please) of all students who took a course from a professor for which both the department of the course and the department of the professor were different from the student's department.

   (b) Write a SQL3 query to find the names and departments (no duplicates please) of all professors whose office includes the string "Gates" and who taught a course in the "CS" department in quarter "S98".

(c) Suppose you are using a non-SQL3 database system, so you don't have the capability to create row types or references as above. Give an alternate schema to the one above that uses generic table definitions. SQL syntax and column types are unnecessary—just specify the relation and column names. Your schema should match the one above as closely as possible.

(d) Now suppose you are allowed to create a total of four indexes on the tables in your schema from part (c). Assume that table `Took` is dramatically much bigger than the other tables, and that your goal is to speed up queries such as those in parts (a) and (b). Give the four index creation statements.

8. **OQL** (5 points)

   The following ODL schema is taken from Written Assignment #7.

```
interface Student
(extent Students, key ID) {
   attribute integer ID;
   attribute Struct{string first, string last} name;
   relationship Set<Internship> applied
      inverse Internship::applicants;
  }

interface Internship
(extent Positions, key (company, city)) {
   attribute string company;
   attribute string city;
   relationship Set<Student> applicants
      inverse Student::applied;
  }
```

   Write an OQL query to find the ID's of all students (no duplicates please) such that either all of the internships the student applied to are in Palo Alto, or all of the internships the student applied to are in San Jose.

9. **SQL3 Recursion** (5 points)

This problem uses relation `Student` from Problem 1:

```
Student(ID, name, dorm, GPA)  // ID is the only key
```

Can you write a SQL query that is equivalent to the following one but does not use recursion? If so, give the query. If not, explain why not.

```
WITH RECURSIVE Mystery AS
   ( (SELECT ID, name FROM Student WHERE GPA > 3.5)
     UNION
     (SELECT Student.ID, Mystery.name
      FROM Student, Mystery
      WHERE Student.name = Mystery.name) )
SELECT * FROM Mystery
```

(Scratch space)

10. **Data Warehousing and Mining** (3 points)

   (a) Give two differences between a *fact* table (e.g., table `Sales` in the lecture example) and a *dimension* table (e.g., tables `Stores`, `Items`, and `Custs` in the lecture example) in a ROLAP data warehousing scenario.

   Difference 1:

   Difference 2:

   (b) Give the most famous example of an *association rule* discovered by a data mining system, as reported in lecture.

(Scratch space)