Final Examination

- Please read all instructions (including these) carefully.
- There are nine 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. 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	10	
2	15	
3	10	
4	10	
5	10	
6	20	
7	10	
8	10	
9	5	
TOTAL	100	

1. **Triggers** (10 points)

   Consider the simple relation `Employee(ID,salary)` storing employee IDs and salaries, where `ID` is a key. Consider the following two triggers over this relation:

```
create trigger T1
after insert on Employee
referencing new as New_Emp
update Employee
   set salary = 1.1 * (select max(salary) from Employee)
   where ID = New_Emp.ID
for each row

create trigger T2
after insert on Employee
referencing newtable as New_Emp
update Employee
   set salary = 1.1 * (select max(salary) from Employee)
   where ID in (select ID from New_Emp)
for each statement
```

   Assume that relation `Employee` has no tuples in it initially. You are to show the simplest example you can think of where using trigger `T1` will produce a different final database state than using trigger `T2`.

   (a) (2 points) Show a sequence of inserted tuples. For purposes of the example, assume that all tuples are inserted as the result of a single SQL statement.

   (b) (4 points) Show the final database state after trigger execution if only trigger `T1` is defined.

   (c) (4 points) Show the final database state after trigger execution if only trigger `T2` is defined.

2. **Constraints** (15 points)

Consider the following SQL declarations:

```
create table Employee(ID integer unique, salary integer, dept# integer)
create table Department(number integer unique, salaryCap integer)

create assertion Policy check (
  not exists (select *
              from Employee, Department
              where Employee.dept# = Department.number
              and Employee.salary > Department.salaryCap) )
```

(a) (3 points) State in English the policy enforced by assertion `Policy`.

(Scratch space below; see next page for part (b).)

(b) (12 points) Rewrite the above table declarations to use relation-based **check** constraints instead of the general assertion. Your constraints should be defined so that under no circumstances can the policy be violated, even taking into account the "hole" in SQL constraint-checking. Remember that you will be graded on simplicity as well as correctness.

3. **Authorization** (10 points)

   Consider the simple relation `Employee(ID,salary,dept#)` storing employee IDs, salaries, and departments, where `ID` is a key. Suppose you are the owner of this relation, and you would like to give user `Mary` authorization to select records for those employees (and *only* those employees) who earn less than $50,000 and work in a department with fewer than 10 people. Show the necessary sequence of commands.

4. **Transactions** (10 points)

Consider the simple relation `Employee(ID,salary)` storing employee IDs and salaries, where `ID is a key`. Consider the following two transactions:

```
T1: begin transaction
    update Employee set salary = 2 * salary where ID = 25
    update Employee set salary = 3 * salary where ID = 25
    commit

T2: begin transaction
    update Employee set salary = 100 where salary > 100
    commit
```

Suppose the salary of the employee with ID = 25 is 100 before either transaction executes.

(a) (5 points) If both transactions T1 and T2 execute to completion with isolation level **serializable**, what are the possible final salaries for the employee with ID = 25?

(b) (5 points) Now suppose transaction T1 executes with isolation level **read committed**, transaction T2 executes with isolation level **read uncommitted**, and both transactions execute to completion. What are the possible final salaries for the employee with ID = 25?

5. **Functional Dependencies** (10 points)

Consider the following "rule" for functional dependencies:

If $A \to B$ and $B, C \to D$ then $A, C \to D$

(a) (4 points) Is this "rule" correct or incorrect? Answer with one word, please.

(b) (6 points) If you said in part (a) that the rule is correct, sketch a simple proof. If you said in part (a) that the rule is incorrect, give a simple counterexample that includes a relation schema and a set of tuples that satisfy the "if" part of the rule but not the "then" part.

6. **Relational Database Design** (20 points)

Consider the following relational schema:

```
Sale(clerk, store, city, date, item#, size, color)
  // records that a clerk sold an item on a particular day
Item(item#, size, color, price)
  // records prices and available sizes and colors for items
```

Make the following assumptions, *and only these assumptions*, about the real world being modeled:

- Each clerk works in one store.
- Each store is in one city.
- A given item# always has the same price, regardless of size or color.
- Each item is available in one or more sizes and one or more colors, and each item is available in all combinations of sizes and colors for that item.

Here are the problems:

(a) (4 points) Based on the assumptions above (and no others), specify all keys for relations `Sale` and `Item`.

(b) (4 points) Based on the assumptions above (and no others), specify an appropriate set of completely nontrivial functional dependencies for relations `Sale` and `Item`.

(c) (4 points) Are relations `Sale` and `Item` in Boyce-Codd Normal Form (BCNF)? If not, decompose the relations so they are in BCNF.

(d) (4 points) Now consider your decomposed relations from part (c), or the original relations if you did not need to decompose them for part (c). Based on the assumptions above (and no others), specify all nontrivial multivalued dependencies for the relations. Do not include multivalued dependencies that also are functional dependencies.

(e) (4 points) Are the relations you used in part (d) in Fourth Normal Form (4NF)? If not, decompose the relations so they are in 4NF.

7. **OQL** (10 points)

Consider the following ODL class declarations:

```
interface Employee {
  extent Emps;
  unique attribute integer ID;
  attribute Struct<string name, string street, string city> info;
  relationship Set<Client> my_clients
    inverse Client::my_sellers; }

interface Client {
  extent Clients;
  unique attribute integer client#;
  attribute Struct<string name, string street, string city> info;
  relationship Set<Employee> my_sellers
    inverse Employee::my_clients; }
```

Write a query in OQL to find the IDs and names of all employees who have a client located on the same street in the same city as the employee. Use `distinct` if and only if it's necessary to remove duplicates in the answer.

8. **Object-Relational SQL3** (10 points)

Consider the following SQL3 schema using *row types*, which emulates the ODL schema from Problem 7 but omits the inverse relationship in `Client`.

```
create row type InfoType (name string, street string, city string)
create row type ClientType (client# integer, info InfoType)
create row type EmpType (ID integer, info InfoType,
                         client ref(ClientType))

create table Employee of type EmpType   // ID is a key
create table Client of type ClientType  // client# is a key
```

Note that an employee may have many clients, so relation `Employee` is not in Boyce-Codd Normal Form.

Write a query in SQL3 to find the IDs and names of all employees who have a client located on the same street in the same city as the employee. Use `distinct` if and only if it's necessary to remove duplicates in the answer.

9. **Recursion** (5 points)

Consider a relation `Edge(N1 integer, N2 integer)`, where a tuple (n_1,n_2) in `Edge` indicates that there is an edge from node number n_1 to node number n_2 in a directed graph G. Consider the following `with` statement in SQL3.

```
with recursive Mystery as
  ( (select * from Edge)
    union
    (select Mystery.N1, Edge.N2
     from Mystery, Edge
     where Mystery.N2 = Edge.N1) )
select N1
from Mystery
where N1 = N2
```

In one sentence, state in English what information is returned by the above `with` statement.