CS145 Final Examination

Spring 2003, Prof. Widom

- Please read all instructions (including these) carefully.
- There are 11 problems on the exam, with a varying number of points for each problem and subproblem for a total of 120 points to be completed in 120 minutes. *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 three 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 blank areas and backs of the exam pages for scratch work. 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: ____

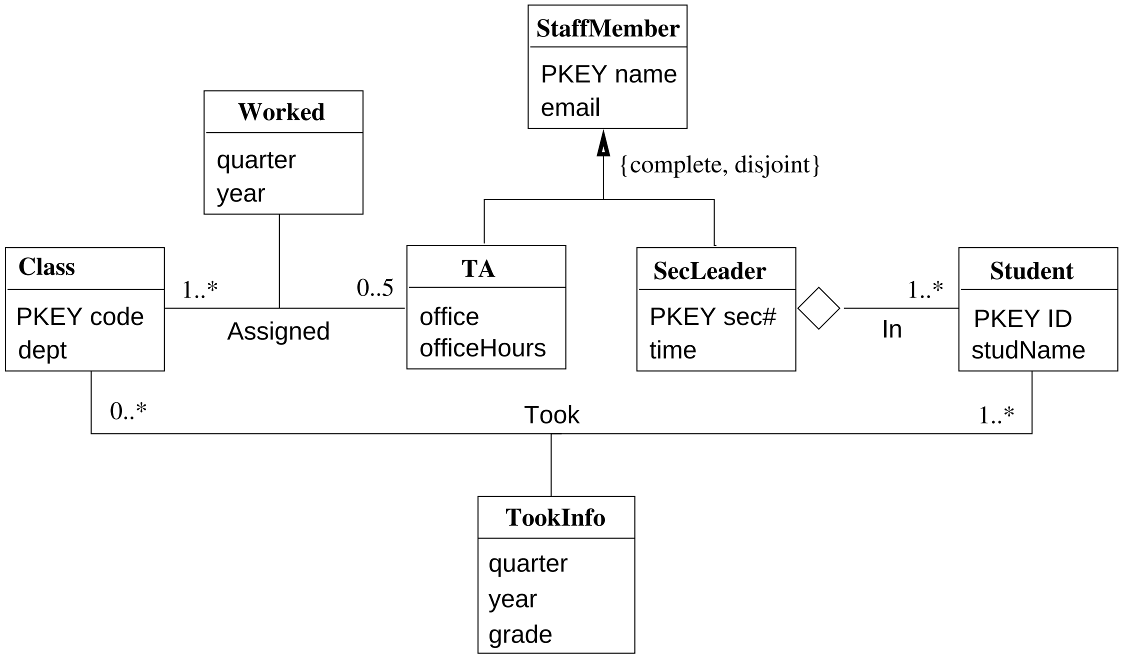
In accordance with both the letter and spirit of the Honor Code, I have neither given nor received assistance on this examination.

SIGNATURE: ____

Problem	1	2	3	4	5	6	7	8	9	10	11	TOTAL
Max. points	15	8	12	10	8	18	5	8	18	10	8	120
Points												

1. **UML** (15 points)

Consider the following UML design:



On the next page, fill in up to seven relations for a relational schema generated from this UML design:

- If you don't use all seven relations, cross out the unused ones.
- Specify the set of attributes for each generated relation.
- Underline a key for each generated relation.
- Assume no attributes in the UML classes are intended to take on NULL values. If any of the attributes in your relations need to allow NULLs, circle them.

You should not worry about the real-world interpretation of this database design, just follow the procedure for translating to relations. *Note:* There are multiple correct answers based on the different translation schemes for subclasses.

`StaffMember(                                                          )`
`TA(                                                                   )`
`SecLeader(                                                            )`
`Class(                                                                )`
`Worked(                                                               )`
`Student(                                                              )`
`TookInfo(                                                             )`

2. **Authorization** (8 points)

   Consider the following two tables:

```
Employee(ID, deptNum, salary)  // ID is key
Department(deptNum, type)      // deptNum is key
```

   Is it possible to authorize a user U to delete employees in departments of type "sales," but not delete any other employees? (Assume the SQL standard, not Oracle's implementation.)

   Circle one: YES NO

   If you circled YES, show a sequence of one or more SQL commands to grant this authorization to user U. If you circled NO, briefly explain why it is not possible.

3. **Constraints** (12 points, 6 per part)

Consider a table `Dancers(name,gender)` containing a list of dancers; `name` is a key.

(a) Write a SQL General Assertion stating that table `Dancers` contains the same number of males as females.

(b) There are two distinct reasons the assertion from part (a) cannot be implemented as an attribute- or tuple-based `CHECK` constraint on table `Dancers`. What are the reasons? (Assume the SQL standard for constraints, not Oracle's implementation.)

4. **Dependencies and Triggers** (10 points, 5 each part)

(a) Consider a table $R(A, B, C)$ and suppose we want to enforce the functional dependency $A \rightarrow B$ on R using triggers. For each of the following five operations, if we need to create a trigger on this operation in order to enforce our functional dependency, circle the operation. If we don't need a trigger on this operation, cross it out.

```
insert on R
delete on R
update of A on R
update of B on R
update of C on R
```

(b) Now suppose we want to enforce the multivalued dependency $A \twoheadrightarrow B$ on R using triggers. For each of the following five operations, if we need to create a trigger on this operation in order to enforce our multivalued dependency, circle the operation. If we don't need a trigger on this operation, cross it out.

```
insert on R
delete on R
update of A on R
update of B on R
update of C on R
```

(Ungraded scratch space)

5. **Triggers** (8 points) Consider a simple one-attribute table $R(A)$ where A has type integer, and the following two triggers:

```
create trigger T1
after insert on R
for each row
update R set A = A + 1

create trigger T2
after insert on R
update R set A = A + 1
```

Assume that table R begins empty.

(a) (4 points) We will create trigger $T1$ or trigger $T2$ but not both. Describe the *simplest* initial database modification you can come up with for which the final state of table R after trigger execution is different using trigger $T1$ than using trigger $T2$. You do not need to write the database modification statement in SQL, just specify the inserted, deleted, and/or updated tuples.

(b) (2 points) What is the final state of table R with your initial modifications from part (a) and trigger $T1$?

(c) (2 points) What is the final state of table R with your initial modifications from part (a) and trigger $T2$?

6. **Transactions** (18 points, 6 each part)

Consider a table `Bid(itemID,price)`. There are initially two tuples in `Bid` with values `(i1,10)` and `(i2,20)`. Consider the following two concurrently executing transactions:

```
T1: begin transaction;
    S1: update Bid set price = price + 5;
    S2: insert into Bid values (i3,30);
    commit;

T2: begin transaction;
    S3: select sum(price) as p1 from Bid;
    S4: select max(price) as p2 from Bid;
    commit;
```

In all parts of the problem transaction $T1$ executes with isolation level *serializable*. You may assume both transactions successfully commit, and the individual statements `S1`, `S2`, `S3`, and `S4` within the transactions each execute atomically.

(a) If transaction $T2$ also executes with isolation level *serializable*, what are the possible pairs of values $p1$ and $p2$ returned by $T2$? Please carefully indicate all possible pairs.

(b) If transaction $T2$ executes with isolation level *read committed*, what are the possible pairs of values $p1$ and $p2$ returned by $T2$? Please carefully indicate all possible pairs.

(c) If transaction $T2$ executes with isolation level *read uncommitted*, what are the possible pairs of values $p1$ and $p2$ returned by $T2$? Please carefully indicate all possible pairs.

7. **Object-Relational SQL** (5 points)

Consider the following type definitions in SQL-99:

```
create type PersonType (name NameType, address AddrType,
                        birthdate string)
create type NameType (lastName string, firstName string)
create type AddrType (street string, city string, zip char(5))
```

Let us say that two tuples $P1$ and $P2$ of type `PersonType` denote the same person if and only if all components are the same:

```
P1.name.lastName = P2.name.lastName and
P1.name.firstName = P2.name.firstName and
P1.address.street = P2.address.street and
P1.address.city = P2.address.city and
P1.address.zip = P2.address.zip and
P1.birthdate = P2.birthdate
```

Suppose we want to be able to always test equality of two tuples $P1$ and $P2$ of type `PersonType` by writing $P1 = P2$, instead of using the long conjunction above. What sequence of commands achieves this goal? Do not worry about precise syntax.

8. **Temporal Relational Algebra** (8 points)

Consider a temporal relation `Visits(student,TA,time)`, which records information about students visiting TAs during office hours. Attribute `time` is the special "timestamp" attribute of our temporal relational model. A student may visit different TAs, and may visit a single TA multiple times. Because the special `time` attribute contains sets of intervals as discussed in class, attribute pair `<student,TA>` forms a key for this relation. Write an expression in temporal relational algebra that returns all students who have spent a total of at least 30 hours visiting TAs.

9. **OLAP and Views** (18 points)

Consider the following table containing enrollment information for students in courses:

```
Took(studentID,year,dept,course,units)
```

Assume the following two queries are asked frequently over this data:

```
Q1: select year,dept,sum(units)
    from Took
    group by year, dept
```

```
Q2: select dept,course,sum(units)
    from Took
    group by dept, course
```

(a) (8 points) You are allowed to create one materialized view V over table `Took` to help answer both queries $Q1$ and $Q2$ efficiently in a data warehousing environment. Assume table `Took` is extremely large. Using your materialized view should substantially speed up the execution of both queries. Define the view using regular SQL view syntax:

```
create materialized view V as ...
```

(b) (4 points) Rewrite queries $Q1$ and $Q2$ into equivalent queries that use your materialized view V from part (a) instead of table `Took`.

(c) (6 points) Now suppose you have available the special table `CUBE(Took)`, where the aggregation operator incorporated into the `CUBE` is `sum`. Rewrite queries $Q1$ and $Q2$ into equivalent queries that use `CUBE(Took)` instead of `Took`. Write the queries so their execution is expected to be as efficient as possible over `CUBE(Took)`.

10. **Data Streams** (10 points, 5 per part)

Suppose your AuctionBase project were implemented using a Data Stream Management System (DSMS) instead of a Database Management System (DBMS). Specifically, consider the following two streams:

```
create stream Items(itemID,sellPrice)
create stream Bids(itemID,bidPrice)
```

In these very simplified streams, an element on the `Items` stream indicates a new auction item and the price for which it sells automatically; `itemID` is a key for this stream. An element on the `Bids` stream indicates a bid for a particular item. You may assume bids arrive in strictly increasing price order for each item, and if a bid meets or exceeds the sell price for an item then there are no later bids for that item.

(a) Consider the following query using the CQL language introduced in class for continuous queries over streams and relations:

```
select Istream(I.itemID)
from Items I [Range 1 Hour], Bids B [Rows 1]
where I.itemID = B.itemID and B.bidPrice >= I.sellPrice
```

In one sentence or less, state what this query produces.

(b) Now consider the following query, also in the CQL language:

```
select itemID, avg(bidPrice)
from Bids [Partition By itemID Rows 2]
group by itemID
```

The following elements have arrived on the `Bids` stream, in the following order:

`(i2,14) (i1,4) (i3,7) (i1,6) (i1,8) (i3,12) (i3,14) (i2,20)`

What is the current result of the query?

11. **Recursion** (8 points)

Consider a table $R(A)$ containing positive integers with no duplicates, and the following query in SQL-99 (allowing nonlinear recursion).

```
with recursive Mystery(X,Y) as
   (select A as X, A as Y from R)
    union
   (select m1.X, m2.Y
    from Mystery m1, Mystery m2
    where m2.X = m1.Y + 1)
select max(Y-X)+1 from Mystery
```

In one sentence or less, state what this query returns.