Final Exam Review Sheet and Sample Questions

Information

- The final exam will be held on Wednesday June 10 from 8:30–10:30 AM. Note that the exam will be two hours long, not three hours as scheduled by the university. The exam will be held in the Gates Computer Science Building Room B01 (the Hewlett-Packard Auditorium) on the Stanford campus. All students, including SITN students, are expected to attend the exam on-campus. There will be no early or makeup exams.
- The exam will be closed book. However, each student may bring up to 6 pages of prepared notes. That's 12 total sides of writing on 8-1/2"x11" paper.
- **SITN students:** Please bring a routing slip and a stapler to the exam.
- A sample solution for Written Assignment #7 will be available via the course Web page by 7:00 PM on Friday June 5.
- All staff office hours will continue as usual through Tuesday June 9. Office hours after June 9 are by appointment only.

Review Session

A question-and-answer review session will be conducted by the TA's on Tuesday June 9 from 7:00–8:30 PM in the Gates Computer Science Building Room B08. The review session will not be televised.

Note that there will be *no* Monday afternoon review session on June 8.

Grade Reporting

- Graded Written Assignments #7 will be available at the final exam. Graded final exams with a sample solution will be available from Sharon Lambeth in Gates 419 by 1:00 PM on Friday June 12. Graded exams for SITN students will be sent by courier on Friday June 12. If you are an SITN student and would prefer to have your graded exam left with Sharon, please notify the TA's (by sending e-mail to `cs145ta@cs`) no later than Wednesday June 10.
- By Wednesday morning June 10 all students will be sent by email their scores for all 12 assignments and the midterm exam as recorded by us. If you find any errors in the recording of the scores, please return the relevant assignment or exam to one of the TA's to have the score corrected. All corrections must be submitted by noon on Thursday June 11.

Course Evaluations

We would appreciate having School of Engineering course evaluations returned by all on-campus and SITN students in the course. If you do not fill out an evaluation form in class, please spend a few minutes after the final exam filling out a form. Forms and pencils will be available from the TA's.

Material Covered

The final exam will cover all material covered by the midterm exam (see Handout #18, "Midterm Exam Review Sheet and Sample Questions"). In addition, the following material will be covered:

- All lectures through Wednesday June 3
- Textbook readings: Sections 5.8, 5.10, 6.1–6.6, 7.1, 7.2, 7.4, 8.1–8.7 (except 8.3.2–8.3.4, 8.4.4)
- Written Assignments #4–7
- Programming Assignments #2–5

A few things to note:

- Although all material from the entire course is eligible to appear on the final exam, the exam will be weighted heavily towards the material from the latter half of the course.
- The material on SQL (textbook Sections 5.1–5.7), although not included on this review sheet since it was covered just before the midterm, is considered part of the latter half of the course for exam purposes.
- Material that was not covered by an assignment (i.e., the material covered in the last lecture of the course) is eligible to appear on the final exam, but any questions on this material will be straightforward.
- As on the midterm exam, solutions on the final exam will be graded for simplicity and clarity as well as for correctness.

Here is an outline of the material we have covered since the midterm exam. All of this material is eligible to appear on the final exam.

1. Indexes
   - Properties and uses of indexes
   - Creating indexes in SQL
2. Views
   - Creating and using views
   - Modifying views
3. Constraints and triggers
   - Non-null constraints
   - Key constraints
   - Referential integrity
   - Attribute-based `check` constraints
   - Tuple-based `check` constraints
   - General assertions
   - SQL3 triggers

4. Programming with SQL
   - Embedded SQL, cursors
   - PL/SQL
5. Transactions
   - Motivation: multi-user, crash recovery
   - ACID properties
   - Serializability
   - Transaction rollback
   - Isolation levels: `read uncommitted`, `read committed`, `repeatable read`, `serializable`
6. Security and authorization
   - Privileges
   - Use of views for authorization
   - `grant` and `revoke` statements
   - Grant diagrams
7. Object-relational SQL3
   - Declaring and using row types
   - Queries over tables with row types
   - Declaring value types (ADT's)
   - Queries over tables with value types
8. OQL
   - Methods and class extents in ODL
   - Queries in OQL
9. Recursion in SQL3
   - `with` statement
   - Recursive union
   - Mutual recursion
10. Data warehousing concepts as covered in the last lecture
11. Data mining concepts as covered in the last lecture

Sample Questions

What follows are the questions from Prof. Widom's 1996 midterm or final exam that cover material from the latter half of the course this year.

1. **Queries in SQL**

   Consider the following relational schema:

```
course(course#, dept-name)     // course# is the key
enroll(studentID, course#)     // <studentID,course#> is the key
```

   (a) Write a SQL query to find the ID's of all students who are not enrolled in any courses in the EE department. Do not use the SQL `except` operator.

   (b) Write a SQL query to find the names of all departments that offer at least 20 courses.

2. **Indexes**

   Consider again the following relational schema:

```
course(course#, dept-name)     // course# is the key
enroll(studentID, course#)     // <studentID,course#> is the key
```

   Suppose there are three types of queries commonly asked on this schema:

   (1) Given a course#, find the name of the department offering that course.

   (2) Find all students together with all of the departments they are taking courses in; i.e., effectively take the natural join of `enroll` and `course`.

   (3) Given a studentID, find all courses the student is enrolled in.

   Here's the problem:

   (a) What is the minimal number of indexes needed to speed up all three types of queries?

   (b) On which attributes should these indexes be created?

3. **Triggers**

   Consider the simple relation `Employee(ID,salary)` storing employee ID's 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 new_table as New_Emp
update Employee
   set salary = 1.1 * (select max(salary) from Employee)
   where ID in (select ID from New_Emp)
```

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

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

4. **Constraints**

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) State in English the policy enforced by assertion `Policy`.

(b) Rewrite the above table declarations to use tuple-based `check` constraints instead of the general assertion. Your constraints should be defined so that under no circumstances can the policy be violated. Remember that you will be graded on simplicity as well as correctness.

5. **Authorization**

Consider the simple relation `Employee(ID,salary,dept#)` storing employee ID's, 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.

6. **Transactions**

Consider the simple relation `Employee(ID,salary)` storing employee ID's 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) 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) 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?

7. **OQL**

Consider the following ODL class declarations:

```
interface Employee (extent Emps, key ID) {
  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, key client#) {
  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 ID's 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**

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 BCNF.

Write a query in SQL3 to find the ID's 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**

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.