1995 Systems Qualifying Exam – Database Area
Three problems, 35 points total, 60 minutes

1. **(10 points)** The following is a description of aspects of the "real world" that we want to model. The description is "object-oriented" but also makes sense as a colloquial description.

   (a) There is a class of Employees. Information about Employees includes:

      i. Their Employee ID, which is unique.
      ii. Their name.
      iii. Their address.
      iv. The set of Projects on which they work.

   (b) There is a subclass Managers of Employees. Information about Managers includes all that about Employees, plus a

      v. Level, the "rank" of the manager.

   (c) There is a class of Projects. Information about projects includes:

      i. The location of the project.
      ii. The manager of the project.

   Note: it is possible that two or more projects are at the same location, have the same manager, and even have the same set of employees, yet are different projects. The "object ID" of each project differentiates it from any other project yet is not considered "information" about the project.

   (Part a) Design an Entity-Relationship diagram to represent the information described above. Indicate the entities, relationships, and attributes by rectangles, diamonds, and ovals, respectively.

   (Part b) Design a relational schema for the same information. Indicate primary keys for your relations. Your relation schemas should be in Boyce-Codd Normal Form; if they are not, explain why your choice is appropriate. You have the option to create new attributes to serve as keys if appropriate (again, explain why it is appropriate).

2. **(15 points)** Consider the following self-explanatory relational database schema:

```
student(ID, name, dept-no)      // ID is a key
department(dept-no, building)   // dept-no is a key
```

   (a) Write a relational algebra expression that returns each student's ID along with the building for that student's department.

   (b) Write an SQL query that returns the same result as part (a).

   (c) Suppose there are 5000 students and 100 departments, and suppose that in the underlying storage manager for these relations, 10 student tuples can fit on one page (block), and 25 department tuples can fit on one page. If a straightforward *nested-block* join algorithm is used to compute the query from part (b), approximately how many disk page reads can be expected? You may assume:

      i. Neither relation is sorted and there are no indices.
      ii. Relation `student` is used as the "outer" relation in the nested-block algorithm.

iii. A page is read into the buffer pool every time it's needed, and pages are pinned (fixed) while they are in active use. (In other words, you needn't consider the size of the buffer pool or the replacement strategy in computing your answer.)

Show your computation, and state any additional assumptions that you make.

(d) Use the same scenario as part (c), but drop assumption (i) and assume in its place:

ia. Relation `student` is sorted on attribute `student.dept-no`.

ib. Relation `department` has a dense index on attribute `department.dept-no` that guarantees 2 page accesses per lookup (such as an extendible hashing structure with no overflow buckets).

If the nested-block join algorithm takes advantage of these new assumptions in the obvious way, approximately how many page reads can be expected? Show your computation, and state any additional assumptions that you make.

(e) Use the same scenario as part (d), but drop assumption (ib) and assume in its place:

ib. Relation `department` is sorted on attribute `department.dept-no`.

Suppose now that a *sort-merge* join algorithm is used to compute the query. Approximately how many page reads can be expected? Show your computation, and state any additional assumptions that you make.

3. **(10 points)** You are designing a locking-based concurrency control mechanism for a new object oriented database system. Unlike a traditional relational database system that has tables and tuples, this system has "objects". Each object can have a set of `<label, value>` pairs, where values can either be atomic (integers, strings, reals) or sets of sub-object identifiers.

For example, a company may be represented by an object

```
c1: <name, "IBM">, <employees, {e1, e2, ...}>, <products, {p1, p2,...}>
```

In turn, `e1, e2, ...` identify employee sub-objects and `p1, p2,...` identify product sub-objects, each describing a company employee or a product made by the company. For example, here is one employee object:

```
e1: <name, "Joe">, <age, 21>, <tasks, {t1, t2, ...} >
```

Here, `t1`, `t2` are sub-objects representing the tasks that `e1` is involved in.

What type of concurrency control mechanisms would you build in this case? Transactions access objects in unpredictable ways, so you do not want to make assumptions about access patterns. However, when you do know that a transactions wants to access all sub-objects in a set, you want to be able to lock them quickly, without having to set a lock on each sub-object individually. (For example, a transaction may wish to give a raise to all "IBM" employees.)

Please describe your proposed mechanism briefly and justify your choice. Assume that whoever reads your description is familiar with standard concurrency control schemes, so you do not have to explain the basics.