Database Systems Qual
Spring 1996

This exam is open book and notes. You have 3 hours to complete it. In case the exam is too long for the given three hour period, just try to do the best you can and complete as much of the exam as you can. **Please do each part in a separate blue booklet.** Be sure to write down your "magic identifying number" on each booklet.

Part A (**60 points**)

1. **(10 points)** We consider Datalog programs over a single binary EDB relation E and a single IDB relation R. Program P is *contained* in Q if for each value I for EDB relation E, $P(I) \subseteq Q(I)$; P is *uniformly contained* in Q if for each value I consisting of an EDB relation E and zero or more tuples that are given to be in IDB relation R, $P(I) \subseteq Q(I)$. Show that containment does not imply uniform containment by giving a specific example of Datalog programs P and Q such that P is contained in Q but not uniformly contained in Q.

2. **(10 points)** Consider a binary relation R over AB and the nest operation. For each I, $size(I)$ is defined as the number of entries in the relation. For instance, for I, J, K as follows:

A	B
a	b
a	c
a	d
a'	b
a'	c
a'	d
b	b

instance I

A	B_1
a	$\{b, c, d\}$
a'	$\{b, c, d\}$
b	$\{b\}$

$J = nest_{B_1=(B)}(I)$

A_1	B_1
$\{a, a'\}$	$\{b, c, d\}$
$\{b\}$	$\{b\}$

$K = nest_{A_1=(A)}(J)$

$size(I) = 14$; $size(J) = 10$; $size(K) = 7$. Assuming that the size of I is n,

(a) give (without proof) some bounds on the size of $nest_{B_1=(B)}(I)$;

(b) give (without proof) some bound on $nest_{A_1=(A)}(nest_{B_1=(B)}(I))$.

3. **(15 points)** Consider the following datalog program consisting of two rules:

```
1) p(X,Y) :- a(X,Y) & p(Y,Y)
2) p(X,X) :- a(X,X)
```

We can expand these rules into an infinite sequence of conjunctive queries $Q_0, Q_1, \ldots, Q_i, \ldots$ such that Q_i is constructed by using rule (1) to substitute for the p subgoal i times, followed by one use of rule (2) to eliminate the p subgoal.

(a) Describe Q_i for all $i = 0, 1, \ldots$.

(b) For which i and j is $Q_i \subseteq Q_j$?

(c) Give a nonrecursive datalog program equivalent to the original program above.

4. **(10 points)** Consider the join dependency $J = \bowtie (AB, BC, CD)$ and the multivalued dependency $M = C \twoheadrightarrow A$.

(a) Show that M does not follow from J. (Give a counterexample relation over $ABCD$.)

(b) Prove, using the "chase," that if N is the multivalued dependency $D \twoheadrightarrow B$, then M follows from J and N.

(c) Does M follow from J and the functional dependency $D \rightarrow B$? Explain briefly.

5. **(15 points)** Consider an object database consisting of a single type

$$Person(Name : string, Spouse : Person).$$

The set $Persons$ contains the set of all $Person$ objects. The *deep-equal* relation between persons is defined as the coarsest equivalence relation over $Persons$ such that

$$x \equiv y \Rightarrow x.Name = y.Name \wedge x.Spouse \equiv y.Spouse$$

(a) Give an algorithm computing the deep-equal relation.

(b) Assume this data is represented in a relation R, the semantics of $R(o, x, o')$ is object o has name x and spouse o'. Give a Datalog$^\neg$ program (with edb R) computing the deep-equal relation with the well-founded semantics. You may find it useful to use two binary idb's $Equiv$ and $Not\text{-}Equiv$. Follow the computation of the alternating fixpoint for your program on the instance:

R		
o_1	"$john$"	o_2
o_2	"$mary$"	o_1
o_3	"$john$"	o_4
o_4	"sue"	o_3
o_5	"$john$"	o_6
o_6	"$mary$"	o_5

(c) Can deep-equality be expressed in stratified Datalog$^\neg$? If yes, give the program; if not, give a one line intuitive argument (no proof).

Part B (60 points)

1. **(15 points)** Imagine you're a database administrator (God forbid) for the Internal Revenue Service. For each year and each citizen, you need to maintain information on the citizen's name, dependents (who are also citizens), and dollar amount of tax paid. Each citizen is identified by a social security number, which is the citizen's only immutable key.

   (a) Specify a relational schema for your database and write an SQL `select` statement over that schema for the following query. *Find the social security numbers of all citizens who have used more than one name, had at least three dependents in some year, and paid in some year more tax than the average paid by all citizens over all years.*

   (b) Using your favorite object-oriented data model and declarative object query language (ODMG and OQL are good choices but are not required), specify an object-oriented schema for your database and express the same query as in part (a). For full credit, you should exploit object-oriented constructs rather than simply translating your relational schema and query from part (a) to their object-oriented counterparts.

2. **(10 points)** Consider the following query:

$$(\sigma_{C_1} R_1) \bowtie (\sigma_{C_2} R_2) \bowtie (\sigma_{C_3} R_3) \bowtie (\sigma_{C_4} R_4)$$

   where R_1, R_2, R_3, R_4 are relations and C_1, C_2, C_3, C_4 are selection conditions. Suppose that for executing this query there are two possible access methods for each $\sigma_{C_i} R_i$ and three possible join methods for each join. Assume that all combinations of access and join methods are feasible, and that all joins are asymmetric. How many different physical plans are there for obtaining the result of this query? (Please show your computation if you're interested in the possibility of partial credit. You don't need to do the arithmetic for a final single number.)

3. **(15 points)** Consider execution of the query $R_1 \bowtie R_2 \bowtie R_3$. Let the cardinalities of the three relations be: $|R_1| = 3$, $|R_2| = 6$, $|R_3| = 4$. Assume that whenever a join is performed during query execution, the join has selectivity $= \frac{1}{2}$ (i.e., half of the tuples in the cross-product satisfy the join predicate). Suppose the *iterator* technique is used and nested-loop joins are performed.

   (a) If the join is executed as the left-deep tree $(R_1 \bowtie R_2) \bowtie R_3$, how many times is the *get-next* iterator function called?

   (b) If the join is executed as the right-deep tree $R_1 \bowtie (R_2 \bowtie R_3)$, how many times is the *get-next* iterator function called?

   In both cases, you should account for a "get-result" operator that calls *get-next* on the root of the query plan in order to return the tuples in the result. (Please show your computation if you're interested in the possibility of partial credit. You don't need to do the arithmetic for a final single number.)

4. **(20 points)** Consider a materialized view V defined over a set of base relations $R_1, R_2, \ldots, R_n$. When users make changes to the base relations, the changes must be propagated appropriately to V so that the view stays consistent with the base relations. Assume that V is never modified directly by users, and that before each base relation change V is consistent.

   V is said to be *compile-time self-maintainable* with respect to a modification type M—insert, delete, or update on one of the base relations—if, for all possible states of the view and all possible modifications of type M, V can be modified to correctly reflect the base relation change using only the modification and the contents of V (i.e., not consulting any of the base relations themselves).

   A specific instantiation of V is said to be *run-time self-maintainable* with respect to a specific modification m—an inserted, deleted, or updated tuple in one of the base relations—if V can be modified to correctly reflect the base relation change using only the modification and the contents of V.

   (a) Give the simplest view definition V you can think of and a modification type M such that V is compile-time self-maintainable with respect to M.

   (b) Give the simplest view definition V you can think of and a modification type M such that V is not compile-time self-maintainable with respect to M.

   (c) Give the simplest view definition V you can think of, the (partial) contents of V, (partial) contents of the base relations if necessary, and a modification m such that V is run-time self-maintainable with respect to m but V is not compile-time self-maintainable with respect to m's operation type.

   (d) Give the simplest view definition V you can think of, the (partial) contents of V, (partial) contents of the base relations if necessary, and a modification type m such that V is neither run-time self-maintainable with respect to m nor compile-time self-maintainable with respect to m's operation type.

   View definitions for all four parts should be specified in relational algebra.

Part C (**60 points**)

1. **(30 points)** You are writing the lock manager for a memory-resident database system. You expect that since transactions will run very fast, they will usually hold on to locks for very short periods of time, and thus you expect lock conflicts to be very rare. Because of this, you want to optimize your system so that locking an unlocked item (and releasing a lock when no transactions are waiting for it) are extremely fast operations.

   In your system, you decide to allocate two bits to every in-memory database record. (You do not want to allocate more bits per record since the storage overhead would be too high, according to your boss.) The two bits are:

   - `L`: Set to true if this record is locked exclusively, false if not.
   - `W`: Set to true if some transaction is waiting for this record; false otherwise. If a transaction is waiting, then a conventional locking table is used to record the list of waiting transactions, as explained below.

   Note that we should never have `L = false` and `W = true` in the same record. Also note that to keep things simple, in this problem we are only dealing with exclusive locks.

   To write your lock manager, you have the following:

   - An atomic "test and swap" operation:

     ```
     IF L,W = <V1, V2> THEN L,W <- <V3, V4>
     ```

     This operator checks if the `L` bit equals `V1` and the `W` bit equals `V2`. If so, the `L` bits gets the `V3` value and the `W` bit gets the `V4` value. All this is done as a single atomic operation.
   - For each transaction `T`, a list `LIST(T)` of the record ids of the records this transaction has currently locked. This `LIST(T)` is private to `T` and can only be accessed by it (or the TP system running on its behalf). This list is used when a transaction commits, to know what records to unlock. It is also used for something else, but you'll have to explain that below.
   - A procedure `WAIT(R, T, LIST)`. This procedure (which you do *not* have to write) records the fact that transaction `T` is waiting for record `R`. The data structures used by this procedure are essentially the same as those used by a conventional locking table, and we will refer to them here as the lock table. Parameter `LIST` is the list of records that `T` has locked up to this point.

     Notice that when a transaction `T1` locks a database record `R` that is unlocked, it only sets the `L` bit in `R` itself and does not record anything in the lock table. Thus, when a second transaction `T2` tries to lock `R`, it will call `WAIT(R, T2, LIST)`. Procedure `WAIT` will check the lock table and find no entry for `R`. It will at that point create a record for the lock holder and then will create another record for `T2` waiting for `R`. The lock holder record will have a blank transaction id field because at that point it is not known who locked `R`.

   (a) **(20 points for this sub-part)** Write `LOCK(R, T)`, the procedure that transaction `T` calls to lock database record `R`. Use high level pseudo-code. Be sure that your code is efficient in the expected common cases, even if it is less efficient in the uncommon cases.

Also, be sure to structure your code well and to comment it. Don't forget to update LIST(R).

(b) **(10 points for this sub-part)**: When a waiting transaction calls WAIT(R, T, LIST), it passes in LIST the list of database records locked up to now by T. What should procedure WAIT do with that list? Please explain why it is necessary to pass LIST and how this information will be used.

2. **(30 points)** You are designing an extensible hashing structure that can span multiple computers. You start with a single computer with disks that hold M buckets (disk blocks). As your hash table fills and buckets split, you include more and more of these buckets. When you run out of buckets at the first machine, you add a second computer with M more buckets, then a third, and so on. You are considering at least the following three options for managing the directory for this extensible hash table:

   (a) *Centralized.* The directory is always located in the memory of the first machine. It points to blocks on any of the computers.

   (b) *Replicated.* Same as the centralized solution, except that the full directory is replicated at all computers that are currently participating in the extensible hash table.

   (c) *Hierarchical.* Here we implement a two level extensible hashing scheme. At the first level, each computer is treated as a large bucket. At the second level, each computer is subdivided into smaller buckets (the disk blocks at the computer). To search for a record, we hash the key and separate the hashed value into two parts, one with the high order k_1 bits that the second part with the remaining k_2 bits. We then use the high order i_1 bits of the first part to index the first level directory; this tells us what computer the record is in. We then go to that computer and use the i_2 high bits of the second part to index the second level directory at that machine to identify the disk bucket. The first level directory can either be centralized or replicated at all active computers. Each second level directory is only stored at the computer that holds the disk buckets it points to.

   Please discuss the potential advantages and disadvantages of these strategies. This is an open ended question. Try to cover the important issues: performance, space utilization, flexibility, ease of implementation. Are there improvements to these strategies? Are there other strategies that might be better?

   Be sure to write clearly and to organize your thoughts well. Clarity and conciseness will count substantially.