DB Qual 2010

This exam consists of three parts. Please answer each part in a separate blue booklet. You have a total of 60 minutes for the entire exam. You can allocate time among the parts as you like, but be sure not to get stuck in one part without spending some time in the other parts.

The exam is open book and notes. No laptops are allowed. Simple calculators are allowed.

If you feel that a question is ambiguous or unclear, state any reasonable assumptions you need to make in order to answer the question.

DB Qual 2010; Part A

Problem A1 (10 points)

Consider the following three queries, the first a Datalog program, the second a sequence of assignments in relational algebra, and the third a SQL query. In each of these, Arc(x,y) and Path(x,y) are relations with attributes x and y.

I. Datalog program:

```
Path(a,b) :- Arc(a,b)
Path(a,b) :- Arc(a,c) & Path(c,d) & Arc(d,b)
```

II. Relational algebra (Note: $\bowtie$ is the natural join):

$$\text{Path} := \text{Arc}$$
$$\text{Path} := \pi_{x,w}(\text{Arc} \bowtie \rho_{A(y,z)}(\text{Arc}) \bowtie \rho_{B(z,w)}(\text{Path}))$$

(Note that the answer is Path at the end.)

III. SQL:

```
WITH
    Path(x,y) AS SELECT * FROM Arc
SELECT P1.x, P3.y
FROM Path P1, Path P2, Path P3
WHERE P1.y = P2.x AND P2.y = P3.x;
```

For each pair of queries, describe the differences, if any, between the results produced. The schemas (relation name and attributes) of the results are not significant and differences in the schemas need not be commented upon.

(a) Between I and II.

(b) Between I and III.

(c) Between II and III.

Problem A2 (10 points)

The decomposition of a relation R is said to be *lossless* if, when you project R onto each of the schemas of the decomposition and then take the natural join of the results, you are guaranteed to get back no tuple other than those originally in R. (Note that projection onto any decomposition whatsoever, followed by a natural join of the projections will always produce at least all the tuples originally in R.) Consider the relation $R(A, B, C, D, E)$ with functional dependencies $A \rightarrow E$, $E \rightarrow D$, $CD \rightarrow A$, and $BC \rightarrow E$. Let the decomposition be onto relations with schemas ABC, BCD, and CDE. Prove that this decomposition is lossless.

Problem B1 (8 points)

Consider two tables `R(P)` and `S(F)`. Your goal is to write SQL triggers to maintain a *referential integrity* constraint from `S.F` to `R.P`.

- Specify all and only those triggers needed to guarantee that referential integrity is enforced, regardless of database updates to either table.
- Your triggers should implement the "*on delete cascade*" and "*on update cascade*" referential integrity policies,
- You may assume the constraint that `P` is a primary key for table `R` is already enforced.
- You may assume a **raise-error** command is available for trigger actions.
- If you make any additional assumptions, please state them.

Your triggers should follow the SQL standard covered in the readings, although we are not overly concerned about low-level syntactic correctness.

Problem B2 (6 points)

Consider the following DTD for XML documents containing information about students taking systems quals.

```
<!DOCTYPE Quals [
  <!ELEMENT Quals (Qual*)>
  <!ELEMENT Qual (Area, Taker*)>
  <!ELEMENT Area (#PCDATA)>
  <!ELEMENT Taker EMPTY>
  <!ATTLIST Taker Name CDATA #REQUIRED ]>
```

Write an expression in XPath that operates on a document "**q.xml**" conforming to this DTD. The XPath expression should return the names of all students taking both the *Database* and the *Compilers* quals. Your expression will be graded on simplicity as well as correctness.

Problem B3 (6 points)

Consider query planning for the following two queries expressed in relational algebra, where $\bowtie$ is the *natural join* operator, and $⟗$ is the *natural full outerjoin* operator.

$$Q_1 = (R_1 \bowtie R_2) \bowtie R_3$$

$$Q_2 = (R_1 ⟗ R_2) ⟗ R_3$$

We wish to create candidate query plans represented as trees, with relations as leaves and physical operators as nodes. Assume you have just one type of asymmetric, binary **join** operator to use for query Q_1 (e.g., nested-loops join), and similarly just one type of asymmetric, binary **outerjoin** operator to use for query Q_2. Do not worry about access methods for the relations themselves. How many different candidate query plans are there to compute the result of Q_1 correctly? How many to compute Q_2 correctly? Please justify your answers.

DB Qual 2010; Part C

Problem C1 (10 points)

(a) Consider the following schedule:

$$S_1 = w_1(Z)w_1(X)r_2(X)r_2(Z)w_3(Y)c_3w_1(Y)c_1c_2$$

State if this schedule has the following properties:

1. Serial
2. Conflict serializable
3. Achievable with 2PL (two phase locking)
4. Recoverable
5. ACR (avoids cascading rollback)

(b) Consider the following schedule:

$$S_2 = w_1(Z)w_1(X)r_2(X)r_2(Z)w_3(Y)c_3w_1(W)c_2c_1$$

State if this schedule has the following properties:

1. Serial
2. Conflict serializable
3. Achievable with 2PL (two phase locking)
4. Recoverable
5. ACR (avoids cascading rollback)

Problem C2 (10 points)

You have decided to write a new database systems textbook, and you want to formally explain how checkpointing and recovery work. To start, you explain that after a crash you have a log, and in it each record has a log sequence number (LSN). To refer to the properties of the given log, you define the following functions:

- L: the set of all LSNs that appear in the log.
- COMMIT(i): true if log record with LSN i is a commit record.
- TRAN(i): the transaction id of the transaction what wrote log record i.
- CHS: the set of LSNs of all checkpoint start records in log.
- CHE: the set of LSNs of all checkpoint end records in log.
- ACTIVE(i): Given checkpoint start record with LSN i, function gives the set of active transactions at the time the checkpoint started.
- MAX(S): the maximum element in set S.
- MIN(S): the minimum element in set S.

Based on the functions above, write expressions for the following LSNs. You can use set expressions such as $\{j | j \in S \wedge j > 50\}$, and they can include existential and universal quantifiers. We give you the answer to part (a) as an example. Note that you can use your answers in expressions that follow. For instance, use can use REC(T) to answer the questions that follow part (a).

(a) Write an expression for REC(T), the set of LSNs of all log records for transaction T in log.

Answer: $\text{REC}(T) = \{i | i \in L \wedge \text{TRAN}(i) = T\}$.

Note you are *not* writing an algorithm to compute these expressions, you are only writing a formal description. Efficiency is not a concern here.

(b) Write an expression for CMT, the set of committed transactions.

(c) Write an expression for VALE, the LSN of the end checkpoint record you will use for recovery.

(d) Write an expression for VALS, the LSN of the valid start checkpoint record you will use for recovery.

(e) Write an expression for IGNORE, the largest LSN that is not needed for recovery. That is, LSNs 1 through IGNORE can be immediately discarded.