Qualifying Exam – Spring 2000

PART A – Basic Section

Problem 1 (10 points)

You are building a lightweight database system dedicated to a particular application where transactions always access data in order. In particular, the database objects can be ordered as $x_1, x_2, ..., x_n$ so that if a transaction accesses x_i, it will no longer access any object before x_i. More formally, if the execution schedule S contains actions $p_k(x_i)$ and $p_k(x_j)$ (p can either be a read or a write), and $i < j$ then $p_k(x_i) <_S p_k(x_j)$.

Your colleague claims that because of this ordering property, schedules will always be serializable, and you do not need to implement a concurrency control mechanism. Your colleague tells you that the situation is analogous to deadlock detection when resources are ordered. In that case, transactions request resources in a given order, and hence there can be no cycles in the wait-for-graph. Not implementing concurrency control would mean that your database system for this specific application can be built much faster.

Is your colleague correct? If he is correct, prove that all schedules will be serializable when the given ordering property holds. If he is not correct, give a counter-example.

Qualifying Exam – Spring 2000

PART A – Basic Section

Problem 2 (10 points)

A given transaction T updates database objects A and B, in this order. At some point during or after the execution of T, the system crashes. For each of the two objects, there are four possible cases:

1. T's update was not written to the log, and it was not propagated to the database. (The update could have been reflected in the in-memory log-tail, or could have been made to a copy of the object in the buffer pool, but these changes did not make it to disk.)
2. T's update was written to the log (on disk), but it was not propagated to the database.
3. T's update was not written to the log, but it was propagated to the database (on disk).
4. T's update was written to the log (on disk), and it was propagated to the database (on disk).

Thus, we can represent the state of affairs (after the crash) by two numbers (one for each object) referring to the cases above. For example, $A:1, B:2$ means that A's update was neither logged nor flushed to the database, while B's update was logged but not written to the database.

- Assume that pure *undo* logging is used and that T did *not* commit before the crash. List all possible states using the $A:n, B:m$ notation.
- Assume that pure *undo* logging is used and that T *did commit* before the crash. List all possible states.
- Assume that pure *redo* logging is used and that T did *not* commit before the crash. List all possible states using the $A:n, B:m$ notation.
- Assume that pure *redo* logging is used and that T *did commit* before the crash. List all possible states.

Qualifying Exam – Spring 2000

PART A – Advanced Section

Problem 3 (20 points)

To index a file, you are considering either a partitioned hash organization, or a standard hash organization.

- For the *partitioned hash organization*, you generate an 8-bit hash from two fields, k_1 and k_2, where k_1 contributes the 4 high order bits, and k_2 the rest. The buckets in this organization are 8KB. Each record is 1KB and is stored in the appropriate bucket. Buckets are stored contiguously on disk, starting with the bucket with hash key 00000000 and ending with the one with key 11111111.
- For the *standard hash organization*, we have one hash table on field k_1. The k_1 field is hashed to 8-bits, and again, buckets are 8KB and stored in order on disk. The full record is stored in this hash table. There is also a second hash table on k_2. The hash key is 4 bits. Buckets are 1KB only, and hold entries of the form $\langle v, p \rangle$, where p is a pointer to the location of a record with field $k_2 = v$ in the first hash table. Each $\langle v, p \rangle$ record is 8 bytes.

For this problem we may assume that together k_1 and k_2 constitute a primary key for the file. We can also assume that on the average there are 3 records (and a maximum of 5) that share the same k_1 value, and 3 records on the average (maximum of 5) that share a k_2 value.

(a) For the partitioned strategy, how many IOs are required to find a record given its k_1, k_2 values?

(b) For the partitioned strategy, how many IOs are required to find all records with a given k_1 value? Give an expected and a maximum value if possible. Are some of these IOs sequential?

(c) For the partitioned strategy, how many IOs are required to find all records with a given k_2 value? Give an expected and a maximum value if possible. Are some of these IOs sequential?

(d) For the standard strategy, how many IOs are required to find a record given its k_1, k_2 values?

(e) For the standard strategy, how many IOs are required to find all records with a given k_1 value? Give an expected and a maximum value if possible. Are some of these IOs sequential?

(f) For the standard strategy, how many IOs are required to find all records with a given k_2 value? Give an expected and a maximum value if possible. Are some of these IOs sequential?

Qualifying Exam – Spring 2000

PART A – Advanced Section

Problem 4 (20 points)

Please answer the following questions briefly, in about 100 words or less. Do not just copy sentences from the papers you have read; rather, synthesize the main ideas related to the question asked. You will be graded not just on knowledge, but on how well you can synthesize ideas and explain them clearly.

(a) Database benchmarks often measure the cost in dollars of a particular operation (e.g., a sort), or the cost in dollars for achieving a certain transaction throughput. What cost factors are often overlooked in such benchmarks?

(b) What is chained declustering, and what are its advantages?

(c) What is two-tier replication, and what are its potential advantages?

(d) In the ARIES recovery method, how many recovery phases are there? What does each phase achieve?