

CS 214 Database Management: Assignment 2

Sharad Mehrotra

January 31, 2008

Question 1. In class we noted that the basic advantage of MGL over basic granular locking is that it reduces the lock overhead of transactions that need to access a large number of data items at fine granularity. However, at the same time MGL increases the lock overhead of transactions that acquire locks at finer granularity (since they also need to acquire intention mode locks at coarser granules). Consider a system in which there are only two granules– file and record.

- Illustrate a scenario in which MGL requires fewer locks to be acquired than basic granular scheme.
- Illustrate a scenario in which MGL has a higher lock overhead than basic granular locking.
- After working out the above two examples can you identify the property satisfied by typical granules supported within databases that makes MGL quite attractive.

For the first two parts of the question assume that only 5% transaction request access at file granule. Rest (95%) request locks at the record granule). Furthermore, assume that a transaction requesting lock at a coarser granule accesses every record in the file, whereas transactions that request locks at the record granule request locks on 10 records.

Question 2. Consider a database containing the EMPLOYEE relation consisting of following attributes:

- EMP-NAME
- EMP-SALARY
- EMP-TITLE
- EMP-DEPARTMENT

Assume that the EMPLOYEE relation is maintained as an entry-sequenced file– that is, a new record in the EMPLOYEE relation is always inserted at the end of the file. To enable efficient retrieval of the records using the salary and the name attributes, an index is maintained on the salary and the name attributes. Both the indices are unique– that is, no two employees have the same name and that no two employees make the same salary. Also, each index maintains pointers to the record corresponding to the key value.

The granule graph for the database is shown in Figure 1. The system maintains the following granules: database D1, area A1, file f1, indices – I1 (on salary) and I2 (on name), Key ranges corresponding to I1 and I2, as well as the records.

Multi-granularity locking is used, key ranges and records are maintained as SEPARATE granules, and dynamic key range locking as discussed in class is used.

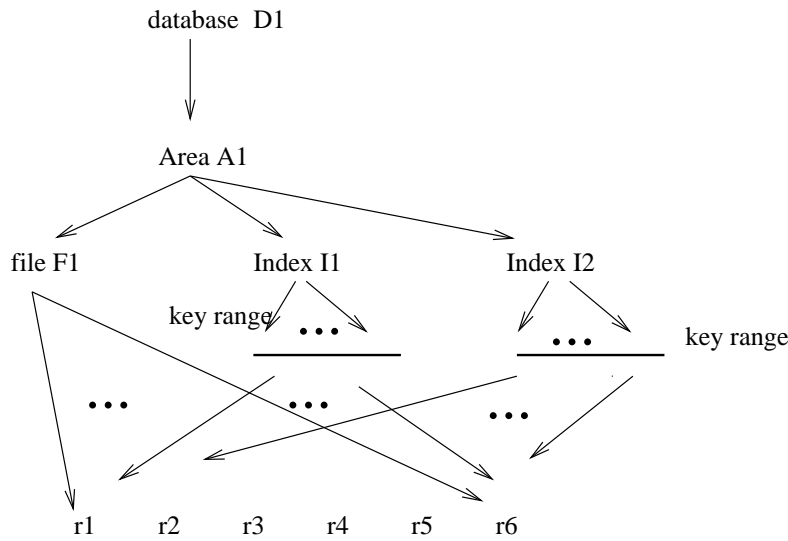


Figure 1: Granule Graph of the Database

Assume that the EMPLOYEE relation contains six records $r1, r2, r3, r4, r5,$ and $r6$ which are as follows:

- (John, 45,000, manager, databases)
- (stella, 21,000, secretary, databases)
- (Judith, 23,000, consultant, software engineering)
- (Sally, 25,000, designer, hardware)
- (Tom, 118,000, vice president, hardware)
- (Alice, 26,000, secretary, hardware)

Thus, the ranges associated with the Name and Salary attributes at the given instance are:

- **Name Ranges:** $(-, Alice], (Alice, John], (John, Judith], (Judith, Sally], (Sally, Stella], (Stella, Tom], (tom, -)$.
- **Salary Ranges:** $(-, 21000], (21000, 23000], (23000, 25000], (25000, 26000], (26000, 45000], (45000, 118000], (118000, -)$.

Specify for each of the following operations exactly what locks are obtained by the transactions. Assume that each transaction is trying to acquire least constraining locks so as to permit highest possible concurrency¹.

¹That is, a solution that states that each transaction acquires locks at the database granule level will get you zero points even though it is correct.

- A transaction that scans the relation looking for all people whose department = “databases” and changes their department name to “systems engineering”. Notice that since there is no index on department, and since the EMPLOYEE relation contains no primary index, the transaction decides to do a relation scan. That is, it sequentially reads in all the pages of the file containing the EMPLOYEE relation and looks for records of employees who work for the database department. For each such employee, it changes the department to systems engineering.
- A transaction that uses the index on salary to scan the EMPLOYEE relation for all employees whose salaries are in the range \$ [20,000, 30,000]. (Notice that the range is a closed interval).
- A transaction that uses the index on name to read the record corresponding to the employee whose name is “Susan”. Note that a record corresponding to “Susan” does not exist in the database.
- A transaction that inserts a new record (Harry, 50,000, Programmer, Database).
- A transaction that deletes Sally’s record from the database.
- A transaction that creates a new index on department name. Notice that the index on department name will not be a unique index.

Question 3. In class we studied two solutions to the phantom problem– basic granular locking (BGL) and its optimization multigranular locking (MGL). Consider transactions using BGL. If each transaction acquires appropriate locks on the granules in accordance with the BGL protocol (that is, lock on all granules in $lset$), and uses two-phase locking protocol, then the execution is serializable and free of all phantom problems. Similarly, MGL combined with 2PL ensures freedom from phantom problems.

Note that both BGL and MGL seem to require a lock based concurrency control protocol (e.g., some version of 2PL). So a valid question is how to guarantee phantom protection if the DBMS follows a different concurrency control protocol. Let us consider a DBMS that follows a timestamp ordering protocol. One way to address the phantom problem is to adopt an approach similar to basic granular locking. With each granule g_i two timestamps are associated– read time stamp ($g_i.rts$) and write timestamp ($g_i.wts$). Each transaction T_j acquires a timestamp $T_j.ts$. When a transaction wishes to execute an operation that accesses a predicate p , as before, the set $lset(p)$ of granules is identified. Let us consider a transaction executing a read operation. For each granule $g_i \in lset(p)$, the following is done. If $T_j.ts < g_i.wts$, then abort T_j . Else, assign to $g_i.rts$ the value $max(g_i.rts, T_j.ts)$ and execute the read operation. Else, if the operation is a write operation, then for each granule $g_i \in lset(p)$, the following is done. If $T_j.ts < g_i.wts$ or $T_j.ts < g_i.rts$, then abort T_j . Else, assign to $g_i.wts$ the value $T_j.ts$ and execute the write operation.

Let us refer to the above protocol as the basic granular timestamp ordering (BGTO) protocol. Will the BGTO protocol ensure conflict serializability of schedules. Will BGTO prevent the phantom problem. Give an intuitive argument or construct counterexamples.

In case the above described BGTO protocol is not correct, try to rectify it such that it prevents phantoms and ensures conflict serializability. Also, try to generalize the BGTO protocol to a multi-granular version in which similar to the MGL protocol different transactions may access data at different granularities.

Question 4. Let us assume that the database is distributed over sites s_1 , s_2 and s_3 and the the lock instance graph corresponding to a set of granules supported is a tree with ‘database’ being the root, and the three sites s_1 , s_2 and s_3 being the nodes at the next level. Recall that in MGL a transaction locking at a finer granule (say a record) needs to acquire intention locks at *all* the higher granules in the lock instance graph. This implies that the transaction wishing to lock at the record granule also needs to acquire a intention mode lock at a database granule. While the locks corresponding to the site (and finer granules) can be maintained by the lock manager at the same site, which lock manager should maintain the database granule lock?

Conceptually there is no problem (assuming there are no failures) with the lock being maintained at any site or it being replicated at certain set of sites, supporting ‘database’ as a granule is a big overhead in terms of performance. Acquiring such a database granule lock requires communication between sites— which is unacceptable since it increases the lock overhead of small transactions substantially. As a result, most systems do not support ‘database’ as a granule. That is, the lock instance graph is typically a forest containing one rooted tree per site with the root corresponding the the site granule.

- What can go wrong if ‘database’ is not supported as a granule.
- Suggest a mechanism for solving the problem in (1) that does not require transactions locking at record level to pay the overhead of supporting ‘database’ as a granule.

Question 5. In our discussion of KRL in class we noted that for a delete of key k_i following locks are required:

- Instant duration IX lock on $range(k_i)$ — that is, $(k_{i-1}, k_i]$.
- X lock on key k_i .
- SIX lock on $range(k_{i+1})$ — that is, $(k_i, k_{i+1}]$.

However, I mentioned that depending upon how certain pathological cases are handled, there may be a need for a stronger instant SIX lock on $range(k_i)$. The key range locking protocol in that case will be as follows:

- Instant duration SIX lock on $range(k_i)$ — that is, $(k_{i-1}, k_i]$.
- X lock on key k_i .
- SIX lock on $range(k_{i+1})$ — that is, $(k_i, k_{i+1}]$.

Is the instant SIX lock (as mentioned in the protocol) actually required or is the IX lock good enough for phantom protection? In case the instant SIX lock is required, illustrate its need by constructing an undesirable scenario that may result in case only an instant IX lock is acquired. Else, argue why an instant IX lock is good enough for correctness. (Hint: whether or not an instant SIX lock is needed will depend upon how you deal with the pathological cases like an inserter attempting to insert a key that is already present in the database, or a deleter attempting to delete a key that is not present in the database.)

Question 6. In class we studied key range locking under two assumptions: *unique keys*, and single access method. In case of a single access method and a unique key, the key and the data

records are in 1-1 correspondence. Thus, if transactions acquires locks on key they do not need to further acquire locks on records. However, in presence of multiple indices which directly point to records in a relation, a lock on a key value may not prevent access to the record using another index defined using a different key. Thus, it may be necessary to acquire locks on records as well.

1. Describe the basic key range locking protocol in case multiple indices may be present over a relation. In your protocol, consider the key ranges and records as separate lockable granules. Assume that each index is over a unique key.
2. Recall that even though using key ranges as lockable granules results in higher concurrency, it increases the lock overhead. One mechanism we studied to overcome this overhead is to merge the range and the key granules. Apply this idea of modify the protocol you described in (1) above such that the key and record granules are merged into a single resource.
3. How does the concurrency provided by (1) and (2) compare. Either argue that they offer the same degree of concurrency or illustrate using an example why one approach provides higher concurrency as compared to the other.
4. We studied a novel mechanism (creating new lock modes) using which the key and range resources can be merged without resulting in loss of concurrency under the assumption of a single index. Could we have used the same technique to merge the key range and record resources and designed a protocol for key range locking in presence of multiple indices that offers as high a degree of concurrency as the protocol that considers range and record as separate resources without the extra locking overhead. Justify your answer.

Question 7. We studied the key range locking protocol under the assumption of a unique key. Generalize the protocol to deal with non-unique keys. Describe your protocol under the assumption that there is a single index (or access method) over the relation. Comment on the degree of concurrency provided by your protocol.