16

OVERVIEW OF TRANSACTION MANAGEMENT

Exercise 16.1 Give brief answers to the following questions:

1. What is a transaction? In what ways is it different from an ordinary program (in a language such as C)?
2. Define these terms: *atomicity, consistency, isolation, durability, schedule, blind write, dirty read, unrepeatable read, serializable schedule, recoverable schedule, avoids-cascading-aborts schedule.*
3. Describe Strict 2PL.
4. What is the phantom problem? Can it occur in a database where the set of database objects is fixed and only the values of objects can be changed?

Answer 16.1 The answer to each question is given below.

1. A *transaction* is an execution of a user program, and is seen by the DBMS as a series or list of actions. The actions that can be executed by a transaction include reads and writes of database objects, whereas actions in an ordinary program could involve user input, access to network devices, user interface drawing, etc.
2. Each term is described below.
   (a) *Atomicity* means a transaction executes when all actions of the transaction are completed fully, or none are. This means there are no partial transactions (such as when half the actions complete and the other half do not).
   (b) *Consistency* involves beginning a transaction with a 'consistent' database, and finishing with a 'consistent' database. For example, in a bank database, money should never be "created" or "deleted" without an appropriate deposit or withdrawal. Every transaction should see a consistent database.

(c) *Isolation* ensures that a transaction can run independently, without considering any side effects that other concurrently running transactions might have. When a database interleaves transaction actions for performance reasons, the database protects each transaction from the effects of other transactions.

(d) *Durability* defines the persistence of committed data: once a transaction commits, the data should persist in the database even if the system crashes before the data is written to non-volatile storage.

(e) A *schedule* is a series of (possibly overlapping) transactions.

(f) A *blind write* is when a transaction writes to an object without ever reading the object.

(g) A *dirty read* occurs when a transaction reads a database object that has been modified by another not-yet-committed transaction.

(h) An *unrepeatable read* occurs when a transaction is unable to read the same object value more than once, even though the transaction has not modified the value. Suppose a transaction T2 changes the value of an object A that has been read by a transaction T1 while T1 is still in progress. If T1 tries to read the value of A again, it will get a different result, even though it has not modified A.

(i) A *serializable schedule* over a set S of transactions is a schedule whose effect on any consistent database instance is identical to that of some complete serial schedule over the set of committed transactions in S.

(j) A *recoverable schedule* is one in which a transaction can commit only after all other transactions whose changes it has read have committed.

(k) A schedule that *avoids-cascading-aborts* is one in which transactions only read the changes of committed transactions. Such a schedule is not only recoverable, aborting a transaction can be accomplished without cascading the abort to other transactions.

3. *Strict 2PL* is the most widely used locking protocol where 1) A transaction requests a shared/exclusive lock on the object before it reads/modifies the object. 2) All locks held by a transaction are released when the transaction is completed.

4. The *phantom problem* is a situation where a transaction retrieves a collection of objects twice but sees different results, even though it does not modify any of these objects itself and follows the strict 2PL protocol. This problem usually arises in dynamic databases where a transaction cannot assume it has locked all objects of a given type (such as all sailors with rank 1; new sailors of rank 1 can be added by a second transaction after one transaction has locked all of the original ones).

   If the set of database objects is fixed and only the values of objects can be changed, the phantom problem cannot occur since one cannot insert new objects into the database.

Chapter 16

Exercise 16.2 Consider the following actions taken by transaction $T1$ on database objects X and Y:

R(X), W(X), R(Y), W(Y)

1. Give an example of another transaction $T2$ that, if run concurrently to transaction T without some form of concurrency control, could interfere with $T1$.
2. Explain how the use of Strict 2PL would prevent interference between the two transactions.
3. Strict 2PL is used in many database systems. Give two reasons for its popularity.

Answer 16.2 The answer to each question is given below.

1. If the transaction T2 performed $W(Y)$ before T1 performed $R(Y)$, and then T2 aborted, the value read by T1 would be invalid and the abort would be cascaded to T1 (i.e. T1 would also have to abort).
2. Strict 2PL would require T2 to obtain an exclusive lock on Y before writing to it. This lock would have to be held until T2 committed or aborted; this would block T1 from reading Y until T2 was finished, thus there would be no interference.
3. Strict 2PL is popular for many reasons. One reason is that it ensures only 'safe' interleaving of transactions so that transactions are recoverable, avoid cascading aborts, etc. Another reason is that strict 2PL is very simple and easy to implement. The lock manager only needs to provide a lookup for exclusive locks and an atomic locking mechanism (such as with a semaphore).

Exercise 16.4 We call a transaction that only reads database object a **read-only** transaction, otherwise the transaction is called a **read-write** transaction. Give brief answers to the following questions:

1. What is lock thrashing and when does it occur?
2. What happens to the database system throughput if the number of read-write transactions is increased?
3. What happens to the database system throughput if the number of read-only transactions is increased?
4. Describe three ways of tuning your system to increase transaction throughput.

Answer 16.4 The answer to each question is given below.

1. *Locking thrashing* occurs when the database system reaches to a point where adding another new active transaction actually reduces throughput due to competition for locking among all active transactions. Empirically, locking thrashing is seen to occur when 30% of active transactions are blocked.
2. If the number of read-write transaction is increased, the database system throughput will increase until it reaches the thrashing point; then it will decrease since read-write transactions require exclusive locks, thus resulting in less concurrent execution.

3. If the number of read-only transaction is increased, the database system throughput will also increase since read-only transactions require only shared locks. So we are able to have more concurrency and execute more transactions in a given time.

4. Throughput can be increased in three ways:

   (a) By locking the smallest sized objects possible.

   (b) By reducing the time that transaction hold locks.

   (c) By reducing hot spots, a database object that is frequently accessed and modified.