Transactions and Logging

CS157B

Chris Pollett

Apr.18, 2005.

Outline

- Transactions
- Undo Logging

Transactions

- In order to understand the recovery process from system failure need to understand little about transactions.
- When doing a sequence of operations from a shell like sqlplus, each query or statement is roughly a transaction.
- In an embedded SQL/JDBC/SQLJ setting a transaction will be a sequence of database commands up until either a COMMIT or ROLLBACK (abort) command is issued.
- The *transaction manager* in the DBMS has the job of making sure a transaction is executed correctly.

Jobs of the Transaction Manager

- The Transaction Manager:
  - Issues signals to the log manager so that necessary log record is written.
  - Assures concurrently executing transactions do not interfere with each other.

Correct Execution of Transactions

- We will use the term *elements* to indicate the objects a transaction can manipulate.
- For instance, an element might be a relation, disk block or, individual tuple.
- A database *state* is a set of values for each of its elements.
- A database is in a *consistent* state if all the values of the elements satisfy the key and value constraints as well as implicit constraints intended by the DB designer.
- The **Correctness Principle** is: If a transaction executes in the absence of any other transaction or system errors, and it starts with a database in a consistent state, then the database will be in a consistent state when the transaction ends.

Primitive Operations of a Transaction

- To reason about transactions we will abstract our some common operations:
  - INPUT(X) - copy disk block of element X to memory. Abbreviate I(X).
  - READ(X,t) - copy X to the transactions local variable t. Abbreviate R(x,t).
  - WRITE(X,t) - copy the value of local variable t to database element X in memory. Abbreviate W(X,t).
  - OUTPUT(X) - copy the buffer containing X to disk. Abbreviate O(X).
- Example transaction: R(A,t), t:=t*2, W(A,t), R(B,t), t:= t*2, W(B,t), O(A), O(B).

Undo Logging

- A *log* is a sequence of *log records,* each telling something about the transaction being done.
- The actions of several transactions can interleave.
- Logging occurs while the transaction is running.
- When a crash occurs, the log records will be consulted to figure out what the database was doing at the time of the crash. This will be used to attempt to repair the database.
- One common style of logging is called *undo logging*.
- If we are using only undo logging, to recover from a crash, we only undo some of the operations that have not committed by the time of the crash and this should restore DB to a consistent state.

Log Records

- What is in a log record?
  - <START T> - indicates transaction T has begun.
  - <COMMIT T> - indicates T has completed successfully and will change no more DB elements. Any changes to the database made by T should appear on disk. Cannot be sure at this point all relevant memory buffers have been flushed.
  - <ABORT T> - Transaction T could not be completed successfully. If T aborts, the transaction manager must ensure no changes done by T should ever appear on disk.
  - Update records of the form <T, X, v> which says T changed the value of X from v to something new.
- Log records are recorded when W(X,v)'s occur not when O(X)'s occur.

The Undo-Logging Rules

- If undo-logging is to suffice for the TM to recover from a system failure two things are necessary:

  (U1) If T modifies X, then a log of the form <T,X,v> must be written before the new value of x is written to disk.

  (U2) If a transaction commits, then its COMMIT log record must be written to disk only after all database elements changed by the transaction have been written to disk, but as soon thereafter as possible.

- So the order a transaction must write stuff to disk is: log record, changed DB elements, and COMMIT log record.

Recovering using Undo Logging

- Suppose now a system failure occurs.
- It is the job of the *Recovery Manager* (RMAN) to get the DB back to some consistent state.
- We will first consider a simple way to do it by looking at the whole log file. We will later consider methods based on *checkpointing*.
- The first step is to divide transaction into committed and uncommitted transactions.
- If we see a <COMMIT T> we know by rule 2 that and changed of T were previously written to disk.
- If we see a <START T> without a <COMMIT T> then some operations done by T might not have been written to disk. T must be undone. We look at <T, X,v>'s and replace the new value of X with v.
- After we are done making these changes we write an <ABORT T> record. When finished processing the log file, we can resume operations.