Recovery

Basic approach is to maintain a log. If a crash occurs we can scan log for operations to redo and operations that need to be undone.

Recap: A *commit point* of a transaction indicates that a transaction has completed and its effects are considered to reflected in the database. At commit point, we force-write the system log to disk and then append a *[commit, T]* to the system log.

General approach to recovery following a system crash.

Search log for:

transactions that have not yet reached commit point - the effects of these transactions can be undone.

transactions that have reached commit point- the effects of these transactions can be redone.

The system log is also kept on disk. It is common to keep block in memory until it is full. Hence, a part of the log may be lost is there is a system crash.

Hence, if a transaction T reaches its commit point, force write block of log to disk prior to appending *{commit, T}* to log.

A log may become quite large which can cause the recovery process to be quite slow.

Checkpoints are often used to improve performance.

A checkpoint involves:

suspend all transactions

force write all database pages in memory

append *checkpoint* to log

resume all suspended transactions

Usually issued a regular intervals. The recovery system need not look at transactions that have committed prior to the last checkpoint

Transactions usually operate under one of two protocols:

Deferred update:

updates not made to database until the transaction has committed.

Immediate update:

updates are immediately reflected in database

Deferred Update Protocol

Recovery Protocol

1. Examine system log back as far as the last *[checkpoint]* entry, making two lists: *uncommitted* transactions and *committed* transactions.

2. Ignore all the operations of the uncommitted transactions.

3. Redo all the operations of committed transactions.

Under the deferred update protocol, the system log needs only to contain the following entry types:

i) *[start_transaction, T]*
ii) *[write_item, T, X, new_value]*
iii) *[commit, T]*

Immediate Update Protocol

1. Make two lists: *uncommitted* transactions and *committed* transactions.

2. Undo all the operations of the uncommitted transactions

3. Undo all the operations of committed transactions that have read an item of previously written by a rolled back transaction.

4. Redo all the operations of committed transactions that have not read an item by a rolled back transaction

Can have cascading rollback.

We need to keep extra records in log to facilitate recovery:

[write_item, T, X, old_value, new_value]
[read_item, T, X]

SQL Support for Transactions

A single SQL statement is always considered to be atomic. Either the statement completes execution without error or it fails and leaves the database unchanged.

In SQL, there is no explicit Begin Transaction statement. Transaction initiation is done implicitly when particular SQL statements are encountered.

Every transaction must have an explicit end statement, which is either a COMMIT or ROLLBACK.

Can specify the characteristics of an SQL statement with the SET command.

Can specify the access mode and isolation level.

Access mode: READ ONLY or READ WRITE.

The default is READ WRITE unless the isolation level of READ UNCOMITTED is specified, in which case READ ONLY is assumed.

Isolation level <isolation>, where <isolation> can be:

READ UNCOMMITTED,
READ COMMITTED,
REPEATABLE READ
SERIALIZABLE.

The default is SERIALIZABLE.

With SERIALIZABLE: the interleaved execution of transactions will adhere to our notion of serializability. However, if any transaction executes at a lower level, then serializability may be violated.

Potential problem with lower isolation levels:

Temporary Update Problem:

Reading a value that was written by a transaction which failed.

Nonrepeatable Read:

Allowing another transaction to write a new value between multiple reads of one transaction.

Phantoms:

New rows being read using the same read with a condition.

Sample SQL transaction:

```
EXEC SQL whenever sqlerror go to UNDO;
EXEC SQL SET TRANSACTION
          READ WRITE
          DIAGNOSTICS SIZE 5
          ISOLATION LEVEL SERIALIZABLE;
EXEC SQL INSERT
         INTO EMPLOYEE (FNAME, LNAME, SSN, DNO, SALARY)
         VALUES ('Robert','Smith','991004321',2,35000);
EXEC SQL UPDATE EMPLOYEE
         SET SALARY = SALARY * 1.1
         WHERE DNO = 2;
EXEC SQL COMMIT;
         GOTO  THE_END;
UNDO: EXEC SQL ROLLBACK;
THE_END;
```

Summary:

Concurrency control – potential problems.

Transaction – states

Desirable properties of transactions

Schedules – serial, serializability

2 phase locking, timestamping

Recovery – system log, entries, algorithms for recovery