Course Name:
Database Management Systems

Lecture 23
Topics to be covered

- Concurrency Control
  - Introduction
  - Concurrency Control vs. Serializability Tests
  - Weak Levels of Consistency
  - Transaction Definition in SQL
  - Locking Schemes
  - Applications
  - Scope of Research

Introduction

- A database must provide a mechanism that will ensure that all possible schedules are
  - either conflict or view serializable, and
  - are recoverable and preferably cascadeless
- A policy in which only one transaction can execute at a time generates serial schedules, but provides a poor degree of concurrency
  - Are serial schedules recoverable/cascadeless?
- Testing a schedule for serializability *after* it has executed is a little too late!
- **Goal** – to develop concurrency control protocols that will assure serializability.

Concurrency Control vs. Serializability Tests

- Concurrency-control protocols allow concurrent schedules, but ensure that the schedules are conflict/view serializable, and are recoverable and cascadeless .
- Concurrency control protocols generally do not examine the precedence graph as it is being created
  - Instead a protocol imposes a discipline that avoids nonseralizable schedules.
  - We study such protocols in Chapter 16.
- Different concurrency control protocols provide different tradeoffs between the amount of concurrency they allow and the amount of overhead that they incur.
- Tests for serializability help us understand why a concurrency control protocol is correct.

Weak Levels of Consistency

- Some applications are willing to live with weak levels of consistency, allowing schedules that are not serializable
  - E.g. a read-only transaction that wants to get an approximate total balance of all accounts
  - E.g. database statistics computed for query optimization can be approximate (why?)
  - Such transactions need not be serializable with respect to other transactions
- Tradeoff accuracy for performance

Transaction Definition in SQL

- Data manipulation language must include a construct for specifying the set of actions that comprise a transaction.
- In SQL, a transaction begins implicitly.
- A transaction in SQL ends by:
  - **Commit work** commits current transaction and begins a new one.
  - **Rollback work** causes current transaction to abort.
- In almost all database systems, by default, every SQL statement also commits implicitly if it executes successfully
  - Implicit commit can be turned off by a database directive
    - E.g. in JDBC, connection.setAutoCommit(false);

Implementation of Isolation

- Schedules must be conflict or view serializable, and recoverable, for the sake of database consistency, and preferably cascadeless.
- A policy in which only one transaction can execute at a time generates serial schedules, but provides a poor degree of concurrency.
- Concurrency-control schemes tradeoff between the amount of concurrency they allow and the amount of overhead that they incur.
- Some schemes allow only conflict-serializable schedules to be generated, while others allow view-serializable schedules that are not conflict-serializable.

Locking Scheme

A lock is available associated with each data item in the database. Manipulating the values of a lock is called locking. Locking the items being used by a transaction can prevent other concurrently running transaction from using these locked items. The locking is done by a sub system of the database management system called lock manager.

Exclusive Lock

The exclusive lock is also called an update or a write lock. The intention of this lock is to provide exclusive use of the data item to one transaction. If a transaction T locks a data item Q in an exclusive mode no other transaction can access Q until the lock is released by transaction T.

Shared Lock

The shared lock is also called a read lock. The intention of this mode of locking is to ensure that the data item does not undergo any modifications while it is locked in this mode. Any number of transactions can concurrently lack and access a data item in the shared mode, but none of these transactions can modify the data item. A data item locked in a shared mode can not be locked in the exclusive mode until the shared lock is released by all transactions holding the lock.

Applications

- Concurrency control in data bases is used to avoid the problems of dead locks as this is one of the serious problems in data base management systems

Scope of Research

- Isolation requirements in concurrency control
- Thread safety for concurrent clients
- Concurrency control in distributed database systems
- Realization of Transaction Concurrency Control in Grid Database
- Realization of concurrency control in grid data bases