Seminar 3

Transactions

Concurrency Management in MS SQL Server

Transactions in SQL Server

- SQL Server uses transactions to compose multiple operations in a single unit of work
  - Each user's work is processed using a different transaction
  - To maximize throughput, transactions should be allowed to execute in parallel
  - The ACID properties of transactions require the net effect to be as if each transaction was serialized

Transactions in SQL Server

- Mechanisms to invoke transactions:
  - **each command** is a transaction if nothing else is specified
  - BEGIN TRAN, ROLLBACK TRAN, COMMIT TRAN is most often used
  - SET IMPLICIT_TRANSACTIONS ON ensures a transaction
  - SET XACT_ABORT ON cause **any** SQL errors to roll back a transaction

Transaction Types

- SQL Server can use local or distributed transactions
  - Nesting transactions (but not nested transaction) is supported
  - Named ***savepoints*** allow rollback of a portion of work
  - SET IMPLICIT_ TRANSACTIONS ON enables chained transactions

Concurrency Problems

- SQL Server transaction isolation solves four major concurrency problems
  - Lost updates occur when two writers modify the same piece of state
  - Dirty read occur when a reader reads uncommitted data
  - Unrepeatable read occurs when existing row change within a transaction
  - Phantoms occur when new rows are added and appear within a transaction

Concurrency Problems

- SQL Server insures isolation by means of locking
  - Write locks are exclusive locks, read locks allow other readers
  - A well-formed transaction acquires the correct lock type prior to using state
  - A two-phased transaction holds all locks until all locks have been acquired
  - Isolation levels determine how long locks are held

Locking in SQL Server

- Locks are typically handled dynamically by the Lock Manager, not through applications
- Granularity of locks:
  - Row/Key, Page, Table, Extent *(contiguous group of 8 pages)* , Database
  - Locks may be assigned at more than one level → hierarchy of related locks
  - Lock escalation > 5000 lock per object (pros&cons)
- Duration of locks (specified through isolation levels):
  - Until operation is done
  - Until the end of transaction

Locking in SQL Server

	S	U	X
S	Yes	Yes	No
U	Yes	No	No
X	No	No	No

- Types of locks:
  - ***Shared*** (S) – for data read
  - ***Update*** (U) – ***S*** lock anticipate to become ***X***
  - ***Exclusive*** (X) – for data modification. Incompatible with other locks (exception NOLOCK hint and READ UNCOMITTED isolation level)
  - ***Intent*** (IX, IS, SIX) – intention to lock (performance improvement)
  - ***Schema*** (Sch-M, Sch-S) – schema modification, schema stability (Sch-M and Sch-S are not compatible)
  - ***Bulk Update*** (BU) – locks the whole table during bulk insert
  - ***Key-range –*** locks many rows based on a condition

Special Locks

- Schema Locks
  - Does not block any DML, only DDL
- Bulk Update Locks
  - Explicit lock (TABLOCK)
  - Automatic during SELECT INTO
- Application Locks
  - Applications can use SQL Server internal lock manager to lock application resources

Isolation Levels in SQL Server

- **`READ UNCOMMITTED`**: no lock when reading;
- **`READ COMMITTED`**: holds locks during statement execution (default) (*Dirty reads*);
- **`REPEATABLE READ`**: holds locks for the duration of transaction (*Unrepeatable reads*);
- **`SERIALIZABLE`**: holds locks and key range locks during entire transaction (*Phantom reads*);
- **`SNAPSHOT`**: work on data snapshot
- SQL syntax:

```
SET TRANSACTION ISOLATION LEVEL …
```

Degrees of isolation

Common Name	Chaos	Read Uncommitted	Read Committed	Repeatable Read	Serializable
Lost Updates?	Yes	No	No	No	No
Dirty Reads?	Yes	Yes	No	No	No
Unrepeatable Reads?	Yes	Yes	Yes	No	No
Phantoms?	Yes	Yes	Yes	Yes	No

Key Range Locking

- Lock sets of rows controlled by a predicate

  …`WHERE salary between 35000 and 45000`

- Need to lock data that doesn't exist!
  - If using "`salary between 35000 and 45000`" doesn't return any rows the first time, it shouldn't return any on subsequent scans
- Earlier version locked larger units to prevent phantoms
  - Prior to SQL Server 7.0, only pages and tables were locked

Transaction Workspace Locks

- Indicate a Connection
  - Every connection to a db acquires ***Shared_Transaction_Workspace*** lock
  - Exceptions are connections in master and tempdb
- Used to Prevent:
  - DROP
  - RESTORE

Deadlocks

- MS SQL Server recognizes a deadlock.
- By default, the newest transaction is terminated
  - Error 1205 – should be captured and handled appropriately
- SET LOCK TIMEOUT: used to specify how long (in milliseconds) a transaction waits for a locked objects to be released (0 = immediate terminate)
- SET DEADLOCK PRIORITY
  - Values: Low, Medium, High, numeric -10…10

Reduce deadlock situations

- keep transaction short
- collect and verify input data from users before opening a transaction
- access resources in the same order within transaction
- keep transactions in a single batch
- use a lower isolation level or row versioning isolation level
- access the least amount of data possible in transaction