Seminar 3

Transactions

Concurrency Control in SQL Server

Transactions in SQL Server

- combine multiple operations into a single unit of work
- the actions of each user are processed using a different transaction
- objective:
  - maximize throughput => transactions must be allowed to execute in parallel
- ACID properties
- serializability

Transactions in SQL Server

- transaction invocation - mechanisms:
  - unless specified otherwise, each command is a transaction
  - BEGIN TRAN, ROLLBACK TRAN, COMMIT TRAN
  - SET IMPLICIT_TRANSACTIONS ON
    - enables chained transactions
- SET XACT_ABORT ON
  - SQL errors => rollback transaction

Transactions in SQL Server

- local transactions / distributed transactions
- one can *nest* transactions (but transactions are not really nested)
- named *savepoints*
  - allow a portion of work in a transaction to be rolled back

Concurrency Problems

- transaction isolation tackles four major concurrency problems:
  - *lost updates* - two transactions (writers) modify the same piece of data
  - *dirty reads* - a transaction (reader) reads uncommitted data, i.e., data changed by another ongoing transaction
  - *unrepeatable reads* - a row read by a transaction (reader) is changed by another transaction while the reader is in progress (if the 1st transaction reads the row again it will get different row values)

Concurrency Problems

- transaction isolation tackles four major concurrency problems:
  - *phantoms* - transaction T1 reads a set of rows based on a search predicate; transaction T2 generates a new row (I/U) that matches the search predicate while T1 is ongoing; if T1 issues the same read operation, it will get an extra row

Concurrency Problems

- transaction isolation is achieved through the locking mechanism
- *write locks*
  - exclusive locks, i.e., they don’t allow other readers / writers
- *read locks*
  - allow other readers
  - don’t allow other writers

Concurrency Problems

- isolation levels determine:
  - whether read locks are acquired for read operations
  - the duration of read locks
  - whether key-range locks are acquired to prevent phantoms

Locking in SQL Server

- locks
  - usually managed by the Lock Manager (not via apps)
- lock granularity:
  - *Row / Key, Page, Table, Extent**, Database*
- hierarchy of related locks
  - locks can be acquired at several levels
- lock escalation
  - > 5000 locks per object (pros & cons)

* contiguous group of 8 pages

Locking in SQL Server

- lock types:
- *Shared* (S)
  - read operations
- *Update* (U)
  - deadlock avoidance mechanism
- *Exclusive* (X)
  - write operations
  - incompatible with other locks

	S	X
S	Yes	No
X	No	No

Locking in SQL Server

- lock types:
- *Exclusive* (X)
  - read operations by other transactions can be performed only when using the NOLOCK hint or the READ UNCOMMITTED isolation level
  - a transaction always acquires exclusive locks to modify data (regardless of the isolation level)
  - exclusive locks are released when the transaction completes execution

Locking in SQL Server

- <u>lock types:</u>
- *Intent* (IX, IS, SIX)
  - intention to lock (for performance improvement purposes)
- *Schema* (Sch-M, Sch-S)
  - schema modification, schema stability
  - Sch-M
    - prevents concurrent access to the table
  - Sch-S
    - doesn’t allow DDL operations to be performed on the table

Locking in SQL Server

- lock types:
- *Bulk Update* (BU)
  - bulk load data concurrently into the same table
  - BULK INSERT statement
  - TABLOCK hint
- *Key-Range*
  - protect a range of rows implicitly included in a set of records read by a transaction (under the SERIALIZABLE isolation level)

Key-Range Locking

- lock sets of rows defined by a predicate

  …`WHERE grade between 8 and 10`
- lock existing data, as well as data that doesn't exist
- use predicate "`grade between 8 and 10`" 2 times => obtain the same rows

Transaction Workspace Locks

- every connection to a database acquires a *Shared_Transaction_Workspace* lock
- exceptions - connections to master, tempdb
- used to prevent:
  - DROP
  - RESTORE

Isolation Levels in SQL Server

- **`READ UNCOMMITTED`**
  - allows dirty reads (a transaction can see uncommitted changes made by another ongoing transaction)
  - no S locks when reading data
- **`READ COMMITTED`** (default isolation level)
  - a transaction cannot read data that has been modified by another ongoing transaction
  - allows unrepeatable reads
  - S locks - released as soon as the SELECT operation is performed

Isolation Levels in SQL Server

- **`READ COMMITTED`**
  - X locks - released at the end of the transaction
- **`REPEATABLE READ`**
  - holds S locks and X locks until the end of the transaction
  - doesn’t allow dirty reads, unrepeatable reads
  - phantom reads can occur

Isolation Levels in SQL Server

- **`SERIALIZABLE`**
  - highest isolation level
  - holds locks (including key-range locks) during the entire transaction
  - doesn't allow dirty reads, unrepeatable reads, phantom reads
- **`SNAPSHOT`**
  - working on a snapshot of the data

- SQL syntax
  - **`SET TRANSACTION ISOLATION LEVEL`** …

Isolation Levels in SQL Server

concurrency probl. / isolation level	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

Deadlocks

- SQL Server uses deadlock detection
- the transaction that's least expensive to roll back is terminated
- capture and handle error 1205
- SET LOCK_TIMEOUT
  - specify how long (in milliseconds) a transaction waits for a locked resource to be released
  - value 0 - immediate termination
- SET DEADLOCK_PRIORITY
  - values: {*LOW*, *NORMAL*, *HIGH*, <numeric-priority>}
  - <numeric-priority> ::= {-10, -9, …, 10}

Reduce the Likelihood of Deadlocks

- transactions - short & in a single batch
- obtain / verify input data from the user before opening a transaction
- access resources in the same order
- use a lower / a row versioning isolation level