Seminar 3

Transactions
Concurrency Control in SQL Server

Transactions 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 - most often used
  - SET IMPLICIT_TRANSACTIONS ON
    - enables chained transactions
- SET XACT_ABORT ON
  - SQL errors => rollback transaction

Transactions in SQL Server

- SQL Server
  - local transactions
  - distributed transactions
- *nesting* transactions is supported (but transactions are not really nested, i.e., nesting is possible only syntactically)
- 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* - an existing row changes within a transaction (=> different reads of the row will return different values)
  - *phantoms* - new rows are added and appear within a transaction

Concurrency Problems

- in SQL Server, 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
- *well-formed transaction*
  - obtains the correct lock type before using the data

Concurrency Problems

- *two-phased transaction*
  - holds all the locks until all the locks have been obtained
- isolation levels determine:
  - whether read locks are acquired for read operations
  - the duration of the acquired 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

- duration of locks
  - controlled by the isolation level
    - until the end of the operation
    - until the end of the transaction

Locking in SQL Server

- lock types:
- *Shared* (S)
  - read operations
- *Update* (U)
  - S lock that is anticipated to become an X lock
- *Exclusive* (X)
  - write operations
  - incompatible with other locks
  - read operations by other transactions can be performed only when using the NOLOCK hint or the READ UNCOMMITTED isolation level

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

Locking in SQL Server

- lock types:
- *Intent* (IX, IS, SIX)
  - intention to lock (for performance improvement purposes)
- *Schema* (Sch-M, Sch-S)
  - schema modification, schema stability (Sch-S not compatible with Sch-M)
  - 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*
  - locks multiple rows based on a condition
- *Application Locks*
  - locks on application resources

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
- previous versions of SQL Server:
  - to prevent phantoms, larger units of data had to be locked
  - SQL Server 7.0: only pages and tables were locked

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`**
  - no locks when reading data
- **`READ COMMITTED`**
  - holds S locks during the execution of the statement (default) (prevents *dirty reads*)
- **`REPEATABLE READ`**
  - holds S locks for the duration of the transaction (prevents *unrepeatable reads*)
- **`SERIALIZABLE`**
  - holds locks (including key-range locks) during the entire transaction (prevents *phantom reads*)

Isolation Levels in SQL Server

- **`SNAPSHOT`**
  - working on a snapshot of the data
- SQL syntax
  - **`SET TRANSACTION ISOLATION LEVEL`**

    …

Degrees of Isolation

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 cheapest transaction is terminated
- error 1205
  - to be captured and appropriately handled
- SET LOCK_TIMEOUT
  - specify how long (in milliseconds) a transaction waits for a locked resource to be released
  - 0 = immediate termination
- SET DEADLOCK_PRIORITY
  - values: {*LOW*, *NORMAL*, *HIGH*, <numeric-priority>}
  - <numeric-priority> ::= {-10, -9, …, 10}

Reduce the Deadlock Likelihood

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