### S3 - 921 - 03.04.2025

### Transactions.

### Concurrency Control in SQL Server

Transaction = multiple SQL statements executed atomically

- for SQL server command, a transaction is created in the background

- Goal: run transactions in parallel

- ACID properties: Atomicity “execute all or nothing”

Consistency - when T starts and ends data should be consistent

Isolation - Transactions do not influence each other

Durability - the changes should be persisted after T commits successfully

- BEGIN TRAN, COMMIT TRAN, ROLLBACK TRAN

- SET XACT\_ABORT ON (default) - if SQL errors appear, T is aborted

- local/ distributed transactions

- autocommit/ implicit/ explicit transaction

- nested transactions (similar to save points)

- e.g BEGIN TRAN T1

.…

...

BEGIN TRAN T2

…

..

COMMIT TRAN T2 -- does not physicallyy commit the data on the disk

COMMIT TRAN T1

BEGIN TRAN

….

….

SAVE TRANSACTION SavePoint123

...

...

ROLLBACK TRANSACTION SavePoint123 -- revert a portion of a T

…

…

COMMIT TRAN

### Concurrency Issues

1. Lost Update
2. Dirty Reads
3. Non-repeatable Reads
4. Phantom Reads

### Locks

- Write/ Read locks

- Write locks - Exclusive locks; do not allow multiple readers / writers

- Read locks - allow multiple readers / do not allow writers

- managed by Lock Manager

- Graularity: row/ key, page, table, extent, database

- extent = 8 contiguous data pages

- Types:

- Shared (S) - read operations

- Update (U) - deadlock avoidance

- Exclusive (X) - write operations, acquired when T whats to modify the data and released when T ends

- Intent - efficiency

- Schema - Sch-M (prevents concurrent access to the same table) , Sch-S (prevents modifying the schema of the table)

- Bulk Update - BULK INSERT operation

- Key-Range locks

### Transaction Isolation Levels:

1. READ UNCOMMITTED
   1. Dirty Reads
   2. No S locks are acquired
   3. X are acquired and released when T ends
2. READ COMMITTED
   1. X: Dirty Reads
   2. Allows: Non-repeatable Reads
   3. X are acquired and released when T ends
   4. S acquired + released when the reading operation ends
3. REPEATABLE READ
   1. X: DR, NR
   2. Allows: Phantom Reads
   3. X are acquired and released when T ends
   4. S acquired + released when the T ends
4. SERIALIZABLE
   1. X are acquired and released when T ends
   2. S acquired + released when the T ends
   3. Key- range locks
5. SNAPSHOT
   1. Readonly copy of the data
   2. Historical data

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Concurrency issue / Isolation Level | chaos | READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE |
| Lost Updates | Yes | no | no | No |  |
| Dirty Reads | yes | yes | no | no |  |
| Non-repeatable Reads | yes | yes | yes | no |  |
| Phantom Reads | yes | yes | yes | Yes-- |  |

### Deadlocks

- SQL SERVER does deadlock detection

- SET LOCK\_TIMEOUT …

- SET DEADLOCK\_PRIORITY LOW, NORMAL, HIGH, {-10, -9, …, 10}

- Preventing deadlocks:

Accessing resources in the same order

Keep transactions short

Lower transaction isolation level or RLV (row level versioning)

Check input from the user

QUIZ: Give two examples of concurrency issues.

OBS.

Transaction vs. Stored Procedures (SPs)

Transactions

- execute all statements or nothing at all

- if at least one statement has an error, all statements are undone

- may be slower compared with SPs

SPs

- groups SQL statements

- not atomically executed

- if one statement has an error, the previous statements which executed successfully are committed, their modifications are persisted

- if one statement has an error, the next statements are not executed

- faster compared to Transactions because they are compiled at once

A good practice could be creating a SP with Transactions.