Telerik Academy

Database Transactions

Transaction Management and Concurrency Control

Databases

Telerik Software Academy

http://academy.telerik.com

Telerik Academy

Table of Contents

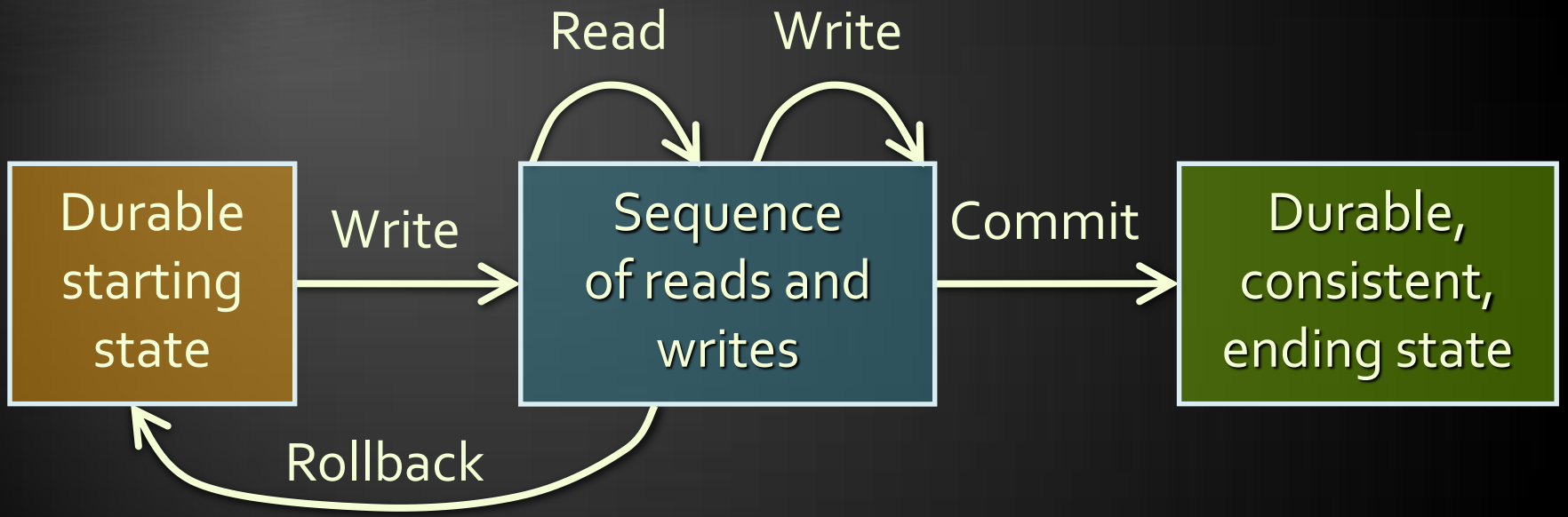
Telerik Academy

What is a Transaction?

Transactions

- Transactions is a sequence of actions (database operations) executed as a whole:
  - Either all of them complete successfully
  - Or none of the them
- Example of transaction:
  - A bank transfer from one account into another (withdrawal + deposit)
  - If either the withdrawal or the deposit fails the whole operation is cancelled

A Transaction



Transactions Behavior

- Transactions guarantee the consistency and the integrity of the database
  - All changes in a transaction are temporary
  - Changes are persisted when COMMIT is executed
  - At any time all changes can be canceled by ROLLBACK
- All of the operations are executed as a whole
  - Either all of them or none of them

Transactions: Example

Withdraw $100

1. Read current balance
2. New balance = current - $100
3. Write new balance
4. Dispense cash

Transfer $100

1. **Read savings**
2. **New savings = current - $100**
3. **Read checking**
4. **New checking = current + $100**
5. **Write savings**
6. **Write checking**

What Can Go Wrong?

- Some actions fail to complete
  - For example, the application software or database server crashes
- Interference from another transaction
  - What will happen if several transfers run for the same account in the same time?
- Some data lost after actions complete
  - Database crashes after withdraw is complete and all other actions are lost

ACID Transactions

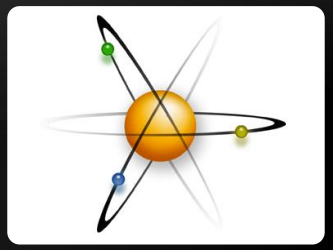
ACID stands for:

A	Atomicity
C	Consistency
I	Isolation
D	Durability

Transactions Properties

- Modern DBMS servers have built-in transaction support
  - Implement "ACID" transactions
  - E.g. MS SQL Server, Oracle, MySQL, …
- ACID means:
  - Atomicity
  - Consistency
  - Isolation
  - Durability

Atomicity



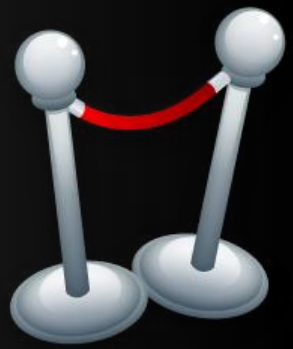
- Atomicity means that
  - Transactions execute as a whole
  - DBMS to guarantee that either all of the operations are performed or none of them
- Atomicity example:
  - Transfer funds between bank accounts
    - Either withdraw + deposit both execute successfully or none of them
    - In case of failure the DB stays unchanged

Consistency

- Consistency means that
  - The database is in a legal state when the transaction begins and when it ends
  - Only valid data will be written in the DB
  - Transaction cannot break the rules of the database, e.g. integrity constraints
    - Primary keys, foreign keys, alternate keys
- Consistency example:
  - Transaction cannot end with a duplicate primary key in a table

Isolation

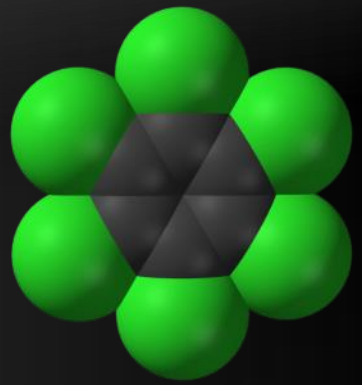
- Isolation means that
  - Multiple transactions running at the same time do not impact each other's execution
  - Transactions don't see other transaction's uncommitted changes
  - Isolation level defines how deep transactions isolate from one another



- Isolation example:
  - Manager can see the transferred funds on one account or the other, but never on both

Durability

- Durability means that
  - If a transaction is committed it becomes persistent
    - Cannot be lost or undone
  - Ensured by use of database transaction logs



- Durability example:
  - After funds are transferred and committed the power supply at the DB server is lost
  - Transaction stays persistent (no data is lost)

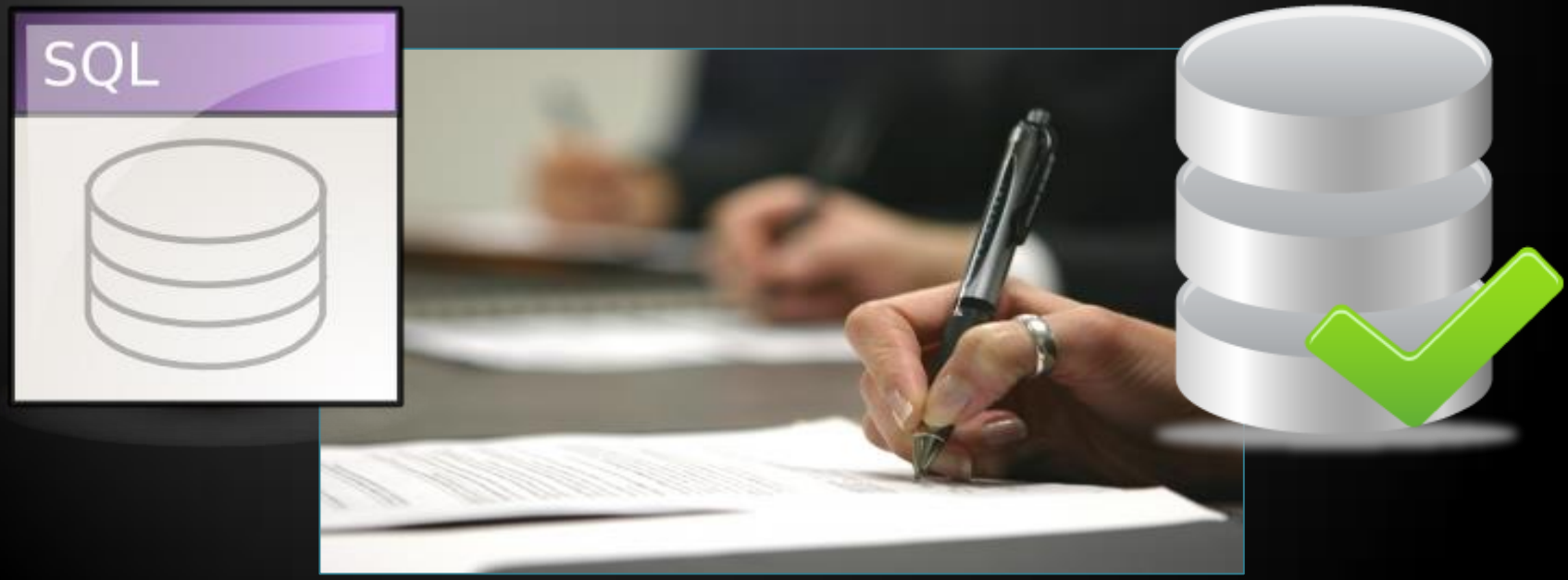
Telerik Academy

ACID Transactions and RDBMS

- Modern RDBMS servers are transactional:
  - Microsoft SQL Server, Oracle Database, PostgreSQL, FirebirdSQL, …
- All of the above servers support ACID transactions
  - MySQL can also run in ACID mode (InnoDB)
- Most cloud databases are transactional as well
  - Amazon SimpleDB, AppEngine Datastore, Azure Tables, MongoDB, …

Telerik Academy

Managing Transactions in SQL Language



Transactions and SQL

- Start a transaction
  - `BEGIN TRANSACTION`
  - Some RDBMS use implicit start, e.g. Oracle
- Ending a transaction
  - `COMMIT`
    - Complete a successful transaction and persist all changes made
  - `ROLLBACK`
    - "Undo" changes from an aborted transaction
    - May be done automatically when failure occurs

Transactions in SQL Server: Example

- We have a table with bank accounts:

```
CREATE TABLE Accounts(
  Id int NOT NULL PRIMARY KEY,
  Balance decimal NOT NULL)
```

- We use a transaction to transfer money from one account into another

```
CREATE PROCEDURE sp_Transfer_Funds(
  @from_account INT,
  @to_account INT,
  @amount MONEY) AS
BEGIN
  BEGIN TRAN;
```

(example continues)

Transactions in SQL Server: Example (2)

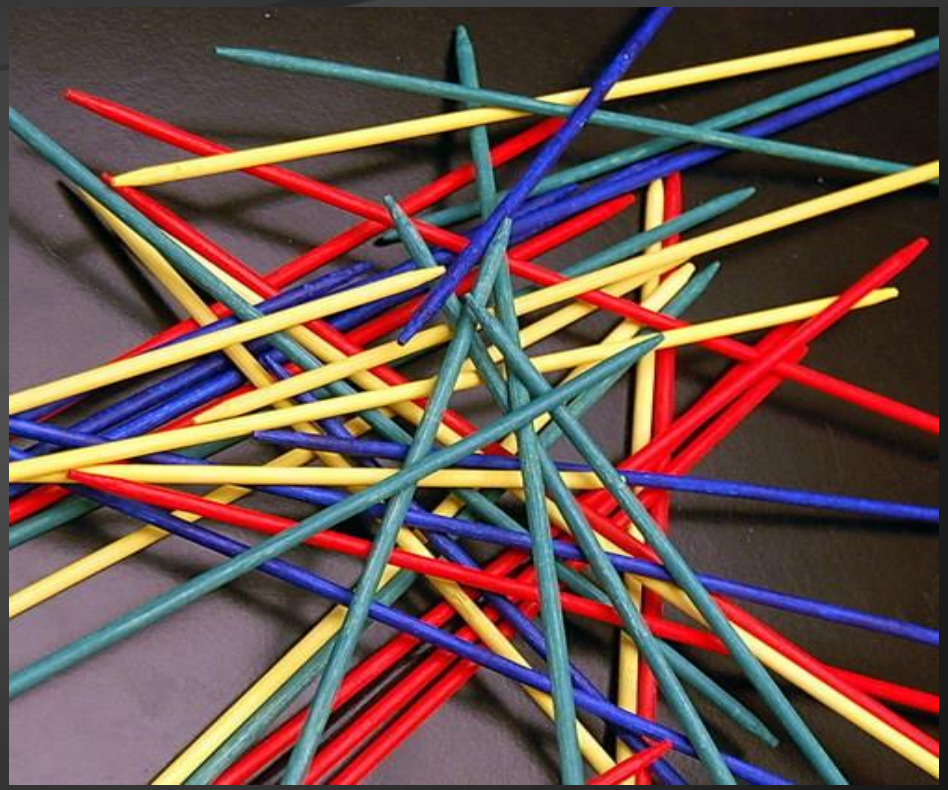
```
  UPDATE Accounts SET Balance = Balance - @amount
  WHERE Id = @from_account;
  IF @@ROWCOUNT <> 1
  BEGIN
    ROLLBACK;
    RAISERROR('Invalid src account!', 16, 1);
    RETURN;
  END;

  UPDATE Accounts SET Balance = Balance + @amount
  WHERE Id = @to_account;
  IF @@ROWCOUNT <> 1
  BEGIN
    ROLLBACK;
    RAISERROR('Invalid dest account!', 16, 1);
    RETURN;
  END;

  COMMIT;
END;
```

Telerik Academy

Transfer Funds

Live Demo

Telerik Academy



Concurrency Problems in Database Systems

Scheduling Transactions

- Serial schedule – the ideal case
  - Transactions execute one after another
    - No overlapping: users wait one another
  - Not scalable: doesn't allow much concurrency
- Conflicting operations
  - Two operations conflict if they:
    1. are performed in different transactions
    2. access the same piece of data
    3. at least one of the transactions does a write operation to that piece of data

Serial Schedule – Example

- T1: Adds $50 to the balance
- T2: Subtracts $25 from the balance
- T1 completes before T2 begins
  - No concurrency problems

Time	Trans.	Step	Value
1	T1	Read balance	100
2	T1	Balance = 100 + 50	
3	T1	Write balance	150
4	T2	Read balance	150
5	T2	Balance = 150 - 25	
6	T2	Write balance	125

Serializable Transactions

- Serializability
  - Want to get the effect of serial schedules, but allow for more concurrency
  - Serializable schedules
    - Equivalent to serial schedules
    - Produce same final result as serial schedule
- Locking mechanisms can ensure serializability
- Serializability is too expensive
  - Optimistic locking allows better concurrency

Concurrency Problems

- Problems from conflicting operations:
  - Dirty Read
    - A transaction updates an item, then fails
    - The item is accessed by another transaction before the rollback
    - The second transaction reads invalid data
  - Non-Repeatable Read
    - A transaction reads the same item twice
      - And gets different values
    - Due to concurrent change in another transaction

Concurrency Problems (2)

- Problems from conflicting operations:
  - Phantom Read
    - A transaction executes a query twice
      - And gets a different number of rows
    - Due to another transaction inserted new rows in the meantime
  - Lost Update
    - Two transactions update the same item
    - The second update overwrites the first
      - Last update wins

Concurrency Problems (3)

- Problems from conflicting operations:
  - Incorrect Summary
    - One transaction is calculating an aggregate function on some records
      - While another transaction is updating them
    - The result is incorrect
      - Some records are aggregated before the updates
      - Some after the updates

Dirty Read – Example

Time	Trans.	Step	Value
1	T1	Read balance	100
2	T1	Balance = 100 + 50	
3	T1	Write balance	150
4	T2	Read balance	150
5	T2	Balance = 150 - 25	
6	T1	Rollback	
7	T2	Write balance	125

Uncommitted

Undoes T1

T2 writes incorrect balance

- Update from T1 was rolled back, but T2 doesn't know about it, so finally the balance is incorrect

Lost Update – Example

Time	Trans.	Step	Value
1	T1	Read balance	100
2	T2	Read balance	100
3	T1	Balance = Balance + 50	
4	T2	Balance = Balance - 25	
5	T2	Write balance	150
6	T1	Write balance	75

- Update from T1 is lost because T2 reads the balance before T1 was completed

Lost update

Telerik Academy

Concurrency Control Techniques

Concurrency Control

- The problem
  - Conflicting operations in simultaneous transactions may produce an incorrect results
- What is concurrency control?
  - Managing the execution of simultaneous operations in the database
  - Preventing conflicts when two or more users access database simultaneously
  - Ensuring the results are correct like when all operations are executed sequentially

Locking Strategies

- Optimistic concurrency control (no locking)
  - No locks – all operations run in parallel
  - Conflicts are possible
    - Can be resolved before commit
  - High concurrency – scale very well
- Pessimistic concurrency control (locking)
  - Use exclusive and shared locks
  - Transactions wait for each other
  - Low concurrency – does not scale well

Optimistic Concurrency

- Optimistic concurrency control (optimistic locking) means no locking
  - Based on assumption that conflicts are rare
  - Transactions proceed without delays to ensure serializability
  - At commit, checks are made to determine whether a conflict has occurred
    - Conflicts can be resolved by last wins / first wins strategy
    - Or conflicted transaction can be restarted
- Allows greater concurrency than pessimistic locking

Telerik Academy

Optimistic Concurrency: Phases

- Three phases of optimistic concurrency:
  - Read
    - Reads DB, perform computations, store the results in memory
  - Validate
    - Check for conflicts in the database
    - In case of conflict → resolve it / discard changes
  - Write
    - Changes are made persistent to DB

Telerik Academy

Optimistic Concurrency Example

1. Read the data from DB:

```
SELECT @fname = FirstName FROM Persons WHERE PersonId = 7
```

2. Remember the state and perform some changes:

```
@old_fname = @fname
@fname = "Some new name"
```

This could take some time (e.g. wait for user action)

3. Update the original database record:

```
UPDATE Persons SET FirstName = @fname
WHERE PersonId = 7 AND FirstName = @old_fname
```

4. Check for conflicts happened during the update:

```
IF @@ROWCOUNT = 0
  RAISERROR ('Conflicting update: row changed. ', 16, 1);
```

Pessimistic Concurrency

- Pessimistic concurrency control (pessimistic locking)
- Assume conflicts are likely
  - Lock shared data to avoid conflicts
  - Transactions wait each other → does not scale well
- Use shared and exclusive locks
  - Transactions must claim a read (shared) or write (exclusive) lock on a data item before read or write
  - Locks prevents another transaction from modifying item or even reading it, in the case of a write lock

Locking – Basic Rules

- If transaction has read lock on an item
  - The item can be read but not modified
- If transaction has write lock on an item
  - The item can be both read and modified
- Reads are not conflicting
  - Multiple transactions can hold read locks simultaneously on the same item
- Write lock gives exclusive access to the locked item
- Transaction can upgrade a read lock to a write lock
  - Or downgrade a write lock to a read lock
- Commits and rollbacks release the locks

Deadlock

- What is deadlock?
  - When two (or more) transactions are each waiting for locks held by the others
- Deadlock example:
  - A locks the "Authors" table
    - And tries to modify the "Books" table
  - B locks the "Books" table
    - And tries to modify the "Authors" table
- Breaking a deadlock
  - Only one way: abort some of the transactions

Dealing with Deadlock

- Deadlock prevention
  - Transaction can't obtain a new lock if the possibility of a deadlock exists
- Deadlock avoidance
  - Transaction must obtain all the locks it needs upfront (before it starts)
- Deadlock detection and recovery
  - DB checks for possible deadlocks
  - If deadlock is detected, one of the transactions is killed and an exception is thrown

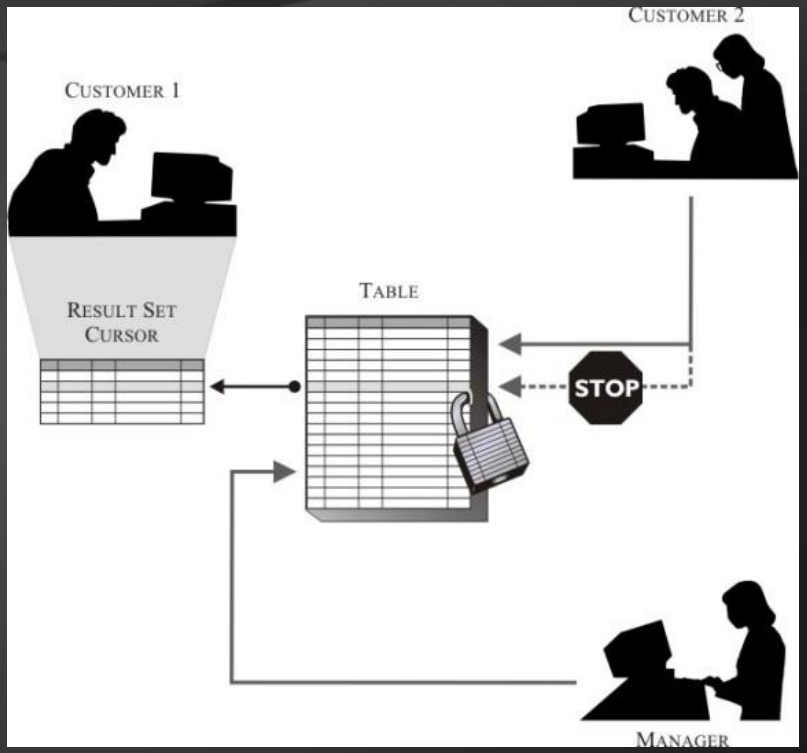
Locking Granularity

- What is locking granularity?
  - The size of data items chosen as unit of protection by concurrency control
- Ranging from coarse to fine:
  - Entire database
  - Single data file
  - Data page (block)
  - Table record
  - Field value of a record

Coarse vs. Fine Granularity

- Coarse granularity
  - Small number of locks protecting large segments of data, e.g. DB, file, page locks
  - Small overhead, small concurrency
- Fine granularity
  - Large number of locks over small areas of data, e.g. table row of field in a row
  - More overhead, better concurrency
- DBMS servers are "smart" and use both

Telerik Academy



Transaction Isolation Levels

Transactions and Isolation

- Transactions can define different isolation levels for themselves

Level of Isolation	Dirty Reads	Repeatable Reads	Phantom Reads
Read uncommitted	yes	yes	yes
Read committed	no	yes	yes
Repeatable read	no	no	yes
Serializable	no	no	no

  - Stronger isolation
    - Ensures better consistency
    - Has less concurrency
    - The data is locked longer

Isolation levels

- Uncommitted Read
  - Reads everything, even data not committed by some other transaction
  - No data is locked
  - Not commonly used
- Read Committed
  - Current transaction sees only committed data
  - Records retrieved by a query are not prevented from modification by some other transaction
  - Default behavior in most databases

Isolation levels

- Repeatable Read
  - Records retrieved cannot be changed from outside
  - The transaction acquires read locks on all retrieved data, but does not acquire range locks (phantom reads may occur)
  - Deadlocks can occur
- Serializable
  - Acquires a range lock on the data
  - Simultaneous transactions are actually executed one after another

Telerik Academy

Snapshot Isolation in SQL Server

- By default MS SQL Server applies pessimistic concurrency control
  - When some transaction updates some data, the other transactions wait it to complete
- A special SNAPSHOT isolation level in MS SQL

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT
```

  - It enables optimistic concurrency control
  - When some transaction updates some data, all other transactions see the old data (snapshot)
  - No locking is applied → no waiting transactions

Telerik Academy

Transactions and Isolation in MS SQL Server

Live Demo

Telerik Academy

Transaction Log and Recovery after Crash

Transaction Log

- What is transaction log (REDO log)?
- Keep a log of all database writes ON DISK (so that it is still available after crash)
  - <transaction ID>; <data item>; <new value>
    - (Tj; x=125) (Ti; y=56)
    - Actions must be idempotent (undoable / redoable)
  - But don't write to the database yet
- At the end of transaction execution
  - Add "commit <transaction ID>" to the log
  - Do all the writes to the database
  - Add "complete <transaction ID>" to the log

Sample Transaction Log

Tid	*Time*	*Operation*	*Object*	*Before image*	*After image*	*PPtr*	*NPtr*
T1	10:12	START				0	2
T1	10:13	UPDATE	STAFF SL21	(old value)	(new value)	1	8
T2	10:14	START				0	4
T2	10:16	INSERT	STAFF SG37		(new value)	3	5
T2	10:17	DELETE	STAFF SA9	(old value)		4	6
T2	10:17	UPDATE	PROPERTY PG16	(old value)	(new value)	5	9
T3	10:18	START				0	11
T1	10:18	COMMIT				2	0
	10:19	CHECKPOINT	T2, T3				
T2	10:19	COMMIT				6	0
T3	10:20	INSERT	PROPERTY PG4		(new value)	7	12
T3	10:21	COMMIT				11	0

Recovering From a Crash

- 3 phases in the recovery algorithm:
  - Analysis
    - Scan for dirty pages in the transaction log
  - Redo
    - Redoes all updates to dirty pages to ensure committed transactions are written to the disk
  - Undo
    - All transactions that were active at the crash are undone, working backwards in the log
- Also handle the cases during the recovery process

Telerik Academy

When and How to Use Transactions?

Transactions Usage

- When to use database transactions?
  - Always when a business operation modifies more than one table (atomicity)
  - When you don't want conflicting updates (isolation)
- How to choose the isolation level?
  - As a rule use read committed, unless you need more strong isolation
- Keep transactions small in time!
  - Never keep transactions opened for long

Transactions Usage – Examples

- Transfer money from one account to another
  - Either both withdraw and deposit succeed or neither of them
- At the pay desk of a store: we buy a cart of products as a whole
  - We either buy all of them and pay or we buy nothing and give no money
  - If any of the operations fails we cancel the transaction (the entire purchase)

Telerik Academy

Database Transactions

http://academy.telerik.com