Transactions and concurrency problems

Managing concurrent use

Learning objectives

1. Learn to create transactions from database operations.
2. Learn about the problems of concurrent use in databases.

Concurrent use of a database

- One-user systems
  - The database is used only from a single workstation
  - No concurrency problems
- Multi-user systems
  - Each database connection forms an executable thread
  - The operating system is responsible for scheduling the threads
    - Each thread receives its own slice of execution time one after another
    - This gives an illusion of concurrency (simultaneous execution)
  - Problems occur, if different database connections handle the same data items
    - Operations may influence each other in an uncontrolled way

Transaction

- Transaction means a collection of operations that perform a single logical operation in database.
- The idea of transaction: an easy way to cancel the whole chain of operations, if one of the operations fails.
- A transaction is successful and its end result is meaningful only after all the operations belonging to the transaction have been successfully executed.
- Each transaction is
  - started (BEGIN TRANSACTION is written to log)
  - finished (END TRANSACTION is written to log)
  - committed (COMMIT is written to log)
- Allowing concurrent execution of different transactions is usually necessary for efficiency
  - Several transactions are unfinished at the same time and they are processed piecewise in turn

Example of transaction

- Money transfer between accounts: 100 euros is transferred from account 4 to account 2
- Two operations:
  1. decreasing the balance of account 4
  2. increasing the balance of account 2

```
1.   UPDATE ACCOUNT
     SET Balance = Balance-100
     WHERE AccountNumber=4;

2.   UPDATE ACCOUNT
     SET Balance = Balance+100
     WHERE AccountNumber=2;
```

Total	**4500,03**
Account #	Balance
1	1000.00
2	0.03
3	500.00
4	3000.00

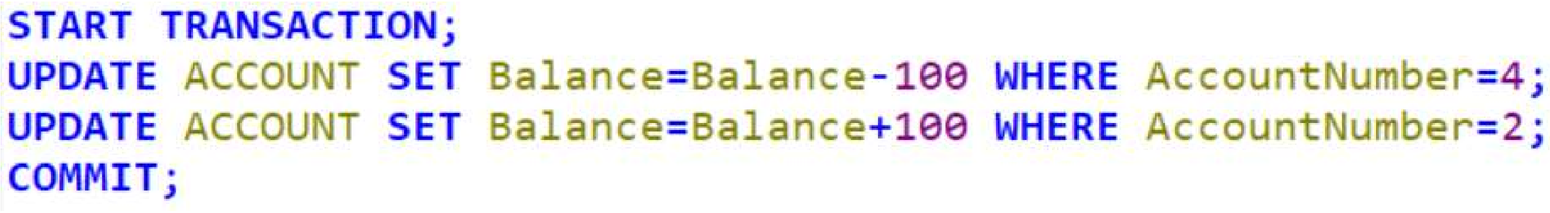
→

Total	**4400.03**
Account #	Balance
1	1000.00
2	0.03
3	500.00
4	2900.00

→

Total	**4500.03**
Account #	Balance
1	1000.00
2	100.03
3	500.00
4	2900.00

Example of transaction



```
START TRANSACTION;
UPDATE ACCOUNT SET Balance=Balance-100 WHERE AccountNumber=4;
UPDATE ACCOUNT SET Balance=Balance+100 WHERE AccountNumber=2;
COMMIT;
```

- In MariaDB, the operations can be entered as a single transaction.

Read and write operations

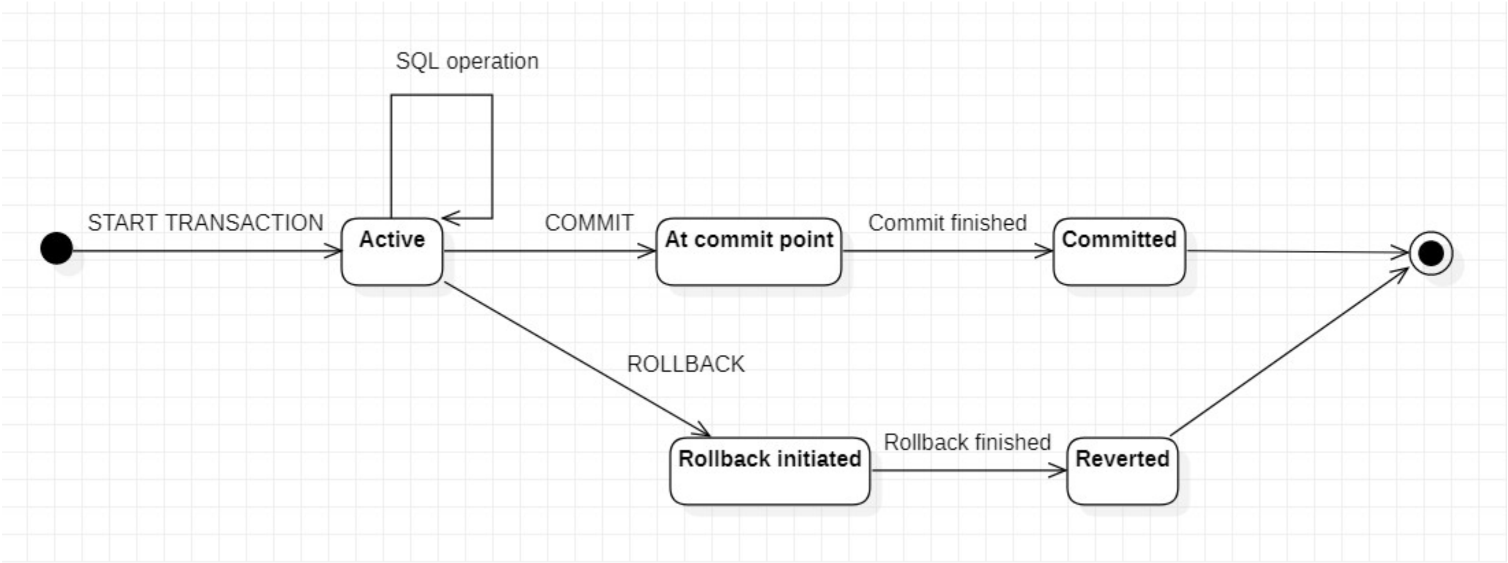
- In the transaction of the previous example consequent read and write operations are applied to the data items
- In practice the data items are disk blocks, but here individual records are considered as data items

TR1
Read 4 (3000)
Write 4 (2900)
Read 2 (0.03)
Write 2 (100.03)

Reasons for transaction failure

1. Database connection drops down.
2. One of the operations cannot be executed.
   - Violation of an integrity constraint (foreign key constraint, CHECK constraint, etc.)
3. The application using the database wants to cancel the transaction.
4. Concurrency management may require cancelling a transaction.
   - Such transactions are automatically restarted.

Transaction states



Committing the transaction

- A transaction is about to commit, when
  - All the operations have been executed successfully
  - All the operations have been written to the log
- When a transaction is about to commit, the DBMS checks that an up-to-date log is on the disk (for efficiency reasons the latest version could be in the central memory only)
- After commit point the impacts of operations are permanent and a COMMIT log entry is written
- In case of a failure
  - All unfinished transactions are found (START_TRANSACTION has been written to the log, but not COMMIT)
  - The operations related to these transactions are cancelled (rolled back)
  - If needed, the finished transactions can be repeated by repeating the write operations related to them (can be found in the log)

When is a transaction cancelled?

- In MariaDB environment a transaction is automatically cancelled in the following situations:
  1. The database connection becomes broken.
  2. The transaction waits for a lock longer than the maximum waiting time .
- The transaction is also cancelled if the user/application layer sends a ROLLBACK command
- If one of the operations in a transaction just fails, the transaction is not automatically cancelled:
  - It is the responsibility of the application layer to observe, if any of the operations fails.
  - If an operations fails the application layer must send the ROLLBACK command.

Concurrency problems

- Here are some of the problems that might occur in concurrent use of a database, if the transactions are not handled correctly:
  - Lost Update
  - Dirty Read
  - Incorrect Summary
  - Phantom Read

Lost Update

- Lost update occurs when two transactions are merged so that one of the updates disappears

- In the example
  - TR1: 100 euros from account 4 to account 2
  - TR2: 100 euros from account 3 to account 2.

TR1	TR2
Read 4 (3000.00)	
	Read 3 (500.00)
Write 4 (2900.00)	
	Write 3 (400.00)
Read 2 (0.03)	
	Read 2 (0.03)
Write 2 (100.03)	
	Write 2 (100.03)

Total	4500,03
Account #	Balance
1	1000.00
2	0.03
3	500.00
4	3000.00

Total	4400.03
Account #	Balance
1	1000.00
2	100.03
3	400.00
4	2900.00

Dirty Read

- Dirty read occurs when an uncommitted changed value is read and this change will later be cancelled (in another transaction)
- In example :
  - TR1: 100 euros from account 4 to account 2
  - TR2: 100 euros from account 1 to account 4
  - Increasing the balance for account 2 fails (TR1)

TR1	TR2
Read 4 (3000.00)	
	Read 1 (1000.00)
	Write 1 (900.00)
Write 4 (2900.00)	
	Read 4 (2900.00)
	Write 4 (3000.00)
Read 2 (0.03)	
failure	
Restore 4 (3000.00)	

Total	4500,03
Account #	Balance
1	1000.00
2	0.03
3	500.00
4	3000.00

→

Total	4400.03
Account #	Balance
1	900.00
2	0.03
3	500.00
4	3000.00

Incorrect Summary

- The problem occurs if an aggregate is based on data items and some of them have been updated while others have not
- In the example:
  - TR1: 100 euros from account 4 to account 1
  - TR2 calculate the sum of balances
  - TR2 receives as a result 4400,03 euros

Total	4500,03
Account #	Balance
1	1000.00
2	0.03
3	500.00
4	3000.00

TR1	TR2
Read 4 (3000.00)	
	Sum 1 (1000.00)
	Sum 2 (0.03)
Write 4 (2900.00)	
	Sum 3 (500.00)
	Sum 4 (2900.00)
Read 1 (1000.00)	
Write 1 (1100.00)	

Phantom Read

- Phantom read occurs when the same data is read twice within the transaction
- Different read occurences return different results because of an insert or delete operation made in another transaction
- When phantom reads occur the transaction is not fully isolated from other transactions
  - Intermediate results that have been calculated within a transaction are not reliable
- Example:
  - TR1: calculate the count of accounts ; calculate the count once again
  - TR2: add account number 5

Total	4500,03
Account #	Balance
1	1000.00
2	0.03
3	500.00
4	3000.00

➡

Total	8625.03
Account #	Balance
1	900.00
2	0.03
3	500.00
4	3000.00
5	4125,60

TR1	TR2
Calculate count (4)	
	Add record (5;4125.60)
Calculate count (5)	

ACID properties

- In the situations presented before, the transactions were not handled correctly
- DBMS offers tools for transaction management
- ACID compatible databases support the correct processing of transactions

A

Atomicity

- Each transaction must be indivisible: it is executed fully or not at all
- A transaction can be successful or fail
- If any of the operations belonging to a transaction fails, the whole transaction fails.

C

Consistency

- The database must maintain consistency: no dangling references, violated constraints etc.
- The database must be in a consistent state
  - Before each transaction
  - After each transaction
- A failed transaction must not bring the database to an inconsistent state.

I

Isolation

- Transactions shall not disturb the execution of other transactions.
- There are mechanisms to avoid missing updates, dirty reads, incorrect summaries and phantom reads.

D

Durability

- When a transaction is committed, the changes shall be permanent.
- No failure in disk systems, operating systems or other errors may endanger the changes.

Isolation Levels

- E.g. in MariaDB, the administrator can control the level at which the data items are isolated from each other
  - 4 isolation levels:
    - READ UNCOMMITTED
      - Allows uncommitted reads and effectively turns off transaction management
    - READ COMMITTED
      - Only the results of committed transactions are visible to other transactions.
      - This is the default isolation level
    - REPEATABLE READ
      - Guarantees that within a transaction all reads are identical, i.e. there are no phantom reads.
    - SERIALIZABLE
      - The data can be updated only when no other transaction is reading it (currently or later during the transaction)
  - The isolation level can be defined separately for every database connection (if the privileges allow); this will override the general setting for the server
- Isolation levels will be implemented with the help of locks and versioning.
  - These techniques will be discussed later on the course.