Data Base Systems

VU 184.686, WS 2020

Transaction Management

Anela Lolić

Institute of Logic and Computation, TU Wien

TU WIEN FAKULTÄT FÜR !NFORMATIK Faculty of Informatics

Acknowledgements

The slides are based on the slides (in German) of Sebastian Skritek.

The content is based on Chapter 9 of
(Kemper, Eickler: Datenbanksysteme – Eine Einführung).
Many examples and illustrations are taken from there.

For related literature in English see Chapter 16 of
(Ramakrishnan, Gehrke: Database Management Systems).

Transaction Management

1. Architecture of a DBMS

2. Transactions

2.1 Definition

2.2 Requirements and Features

3. Transaction Management in SQL

Overview

1. Architecture of a DBMS

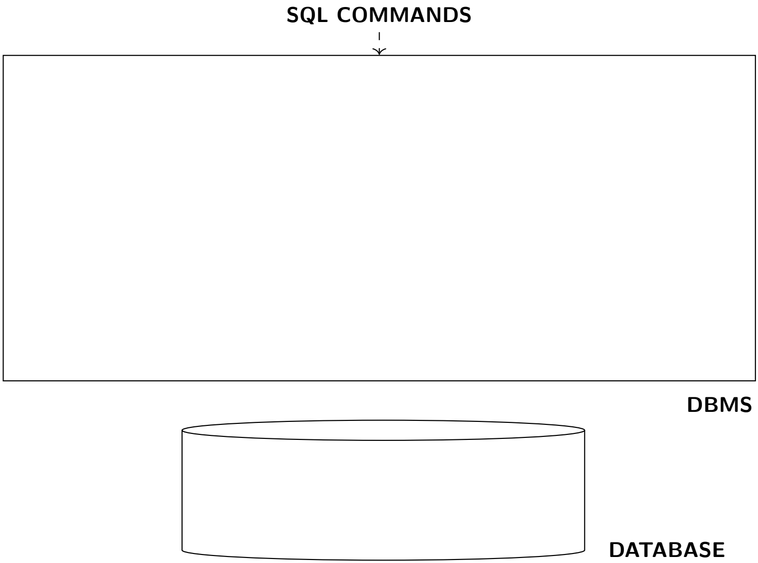
2. Transactions

2.1 Definition

2.2 Requirements and Features

3. Transaction Management in SQL

Architecture of a DBMS

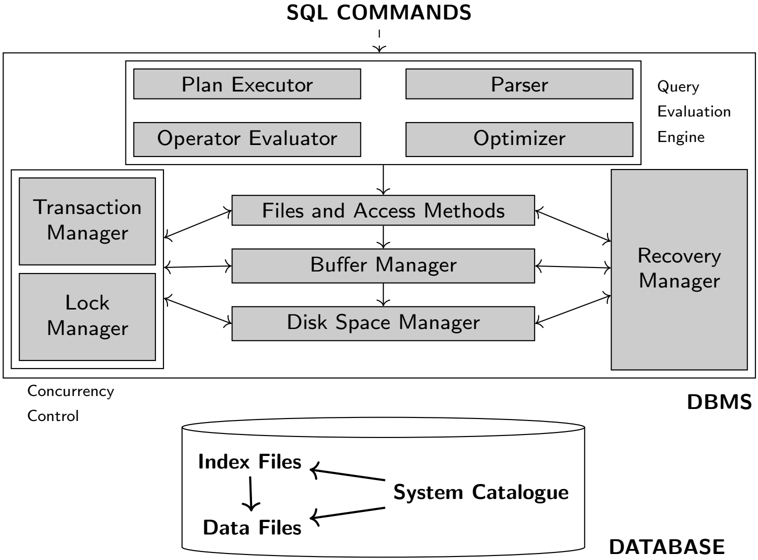


SQL COMMANDS

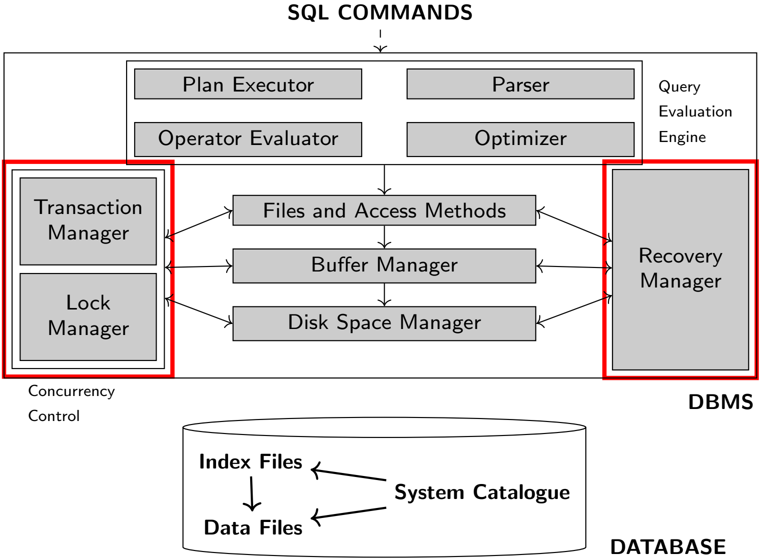
DBMS

DATABASE

Architecture of a DBMS



Architecture of a DBMS



Concurrency Control

transaction manager:

- controls the processing of transactions

locking manager:

- maintains lock requests on data objects (tuple, page, . . .)
- fulfils lock requests for data objects as soon as they are available

Recovery Manager

during operation:

- management of log-file

during recovery after system failure or crash:

- recovery of a consistent state, i.e.
  - redo of all lost operations of successfully completed transactions
  - undo of all operations of not successfully completed transactions

Overview

1. Architecture of a DBMS

2. Transactions

2.1 Definition

2.2 Requirements and Features

3. Transaction Management in SQL

Overview

1. Architecture of a DBMS

2. Transactions

2.1 Definition

2.2 Requirements and Features

3. Transaction Management in SQL

What is a Transaction?

Example

a typical transaction in a bank:

account A, account B

1. read the balance from A into the variable a: **read**(A,a);
2. reduce the balance by 50 €: $a := a - 50$;
3. write the new balance into the database: **write**(A,a);
4. read the balance of B into the variable b: **read**(B,b);
5. increase the balance by 50 €: $b := b + 50$;
6. write the new balance into the database: **write**(B,b);

Transaction

Definition (transaction)

A transaction is a sequence of database operations that transform the database from a consistent state into a consistent state. The execution of a transaction is atomic (= (logical) not interruptible) i.e.

- as unit sound and
- without interference by other transactions.

Operations of a Transaction

for **data handling:**

read(A,a) reads the value of a field A into a *local variable* a

write(A,a) writes the value a into the field A

Operations of a Transaction

for **data handling:**

read(A,a) reads the value of a field A into a *local variable a*

write(A,a) writes the value a into the field A

for **transaction control:**

BOT (begin of transaction)
marks the beginning of a transaction

commit initiates the successful termination of a transaction

abort initiates the abortion of a transaction,
causes the DBMS to reset the database to the state
it was in before the transaction was executed

Operations of a Transaction

additional operations for transaction control:

define savepoint defines a savepoint to which the (still active) transaction can be reset
complete abortion via **abort** is still possible

roll-back to save point initiates the reset of the active transaction to a savepoint
based on the DBMS reset to the last savepoint or to older savepoints possible

Schedule

schedule describes the order of elementary operations during an interleaved execution of several transactions

Example

	T_1	T_2	T_3
1	BOT		
2		BOT	
3			BOT
4		read(B,b_1)	
5	write(A, a_1)		
6			read(C, c_1)
7	commit		
8			abort

Termination of a Transaction

1. successful termination through a commit
2. unsuccessful termination (requires subsequent reset)
   - initiated by user via abort (or roll-back)
   - initiated by DBMS based on an error

Termination of a Transaction

1. successful termination through a commit
2. unsuccessful termination (requires subsequent reset)
   - initiated by user via abort (or roll-back)
   - initiated by DBMS based on an error

BOT	**BOT**	**BOT**
op_1	op_1	op_1
op_2	op_2	op_2
$\vdots$	$\vdots$	$\vdots$
op_n	op_m	op_k
commit	**abort**	∿∿∿∿∿ error

Overview

1. Architecture of a DBMS

2. Transactions

2.1 Definition

2.2 Requirements and Features

3. Transaction Management in SQL

Requirements of Transactions– ACID

Atomicity a transaction is the smallest, not further composable, unit (“everything or nothing at all”).

Consistency a transaction after completion leaves the database in a consistent sate

Isolation concurrently executed transactions shall not influence each other

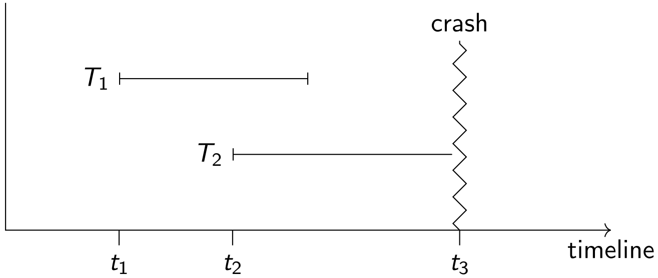
Durability the effects of a successfully completed transactions are not lost
(even when system errors occur)

Components of Transaction Management

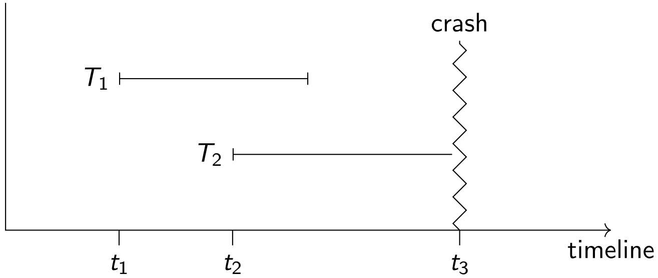
concurrency control: ensure isolation; controls concurrency
concurrency is necessary for performance

recovery: ensures atomicity and durability
guarantees „everything or nothing at all" and makes sure that modifications of a successfully terminated transactions are not lost even when system errors occur

Atomicity and Durability

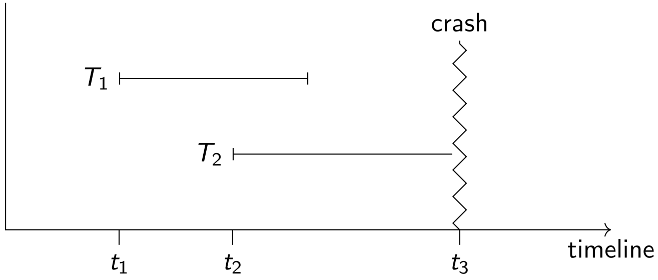


Atomicity and Durability



- transaction T_1 has to exist after recovery

Atomicity and Durability



- transaction T_1 has to exist after recovery
- transaction T_2: all modifications to the database through T_2 have to be removed after recovery

Learning Objectives

- What are transactions?
- Are they important?
- Which operations within a transaction are there?
- What are the possible ends of transactions?
- What do the ACID-features tell us and what is their effect to the transaction management?

Overview

1. Architecture of a DBMS

2. Transactions

2.1 Definition

2.2 Requirements and Features

3. Transaction Management in SQL

Transaction Management in SQL

remark: differences in details between the different DBMS

start of a transaction:

- implicitly through an instruction
- explicitly through SQL commands

```
START TRANSACTION
BEGIN [ WORK | TRANSACTION ]
```

Transaction Management in SQL

termination of a transaction:

- implicitly (“auto commit”)
- explicitly

```
COMMIT [ WORK | TRANSACTION ]
```

  - as long as no problems occur (e.g. consistency violations) the modifications are performed

```
ROLLBACK [ WORK | TRANSACTION ]
ABORT [ WORK | TRANSACTION ]
```

  - modifications are reset
  - DBMS has to guarantee the successful execution

Implicit Transaction End

implicit `commit`

- at implicit beginning and `AUTOCOMMIT = ON`: after each DML/DDL command
- based on DMBS always after DDL (`CREATE TABLE, ...`) and DCL (`GRANT, ...`) commands

implicit `roll-back`

- at problems like system crash, disconnections, consistency violations at `commit`, . . .

good stile: whenever possible, end transactions explicitly

Transaction Management in SQL – Savepoints

- `SAVEPOINT` *sp_name*
  - defines reset point within running transaction
  - enables reset of transaction to this point
  - "economical" use is recommended:
    Can a big transaction be split in smaller ones?

-
  ```
  ROLLBACK [WORK | TRANSACTION]
        TO [SAVEPOINT] sp_name
  ```
  - resets modifications of transactions since *sp_name*

- `RELEASE [SAVEPOINT]` *sp_name*
  - deletes save point (no other effect)