

Lecture

# Database Implementation Techniques / Databases II

*OvGU Magdeburg, WinSem 2009/2010*

Sandro Schulze, Gunter Saake

{sanschul,saake}@iti.cs.uni-magdeburg.de.

# Overview

---

1. Functionality and principles of database systems
2. Architecture of database systems
3. Management of storage device(s)
4. File organisation and data structures
5. Data structures for specific applications
6. Basic algorithms for database operations
7. Query optimization
8. Further aspects and outlook

# Required Knowledge

---

Databases I:

- Basic principles of database systems
- Tables, attributes, keys
- Relational algebra and SQL

*Short repetition at the beginning of the lecture!*

# Literature

---

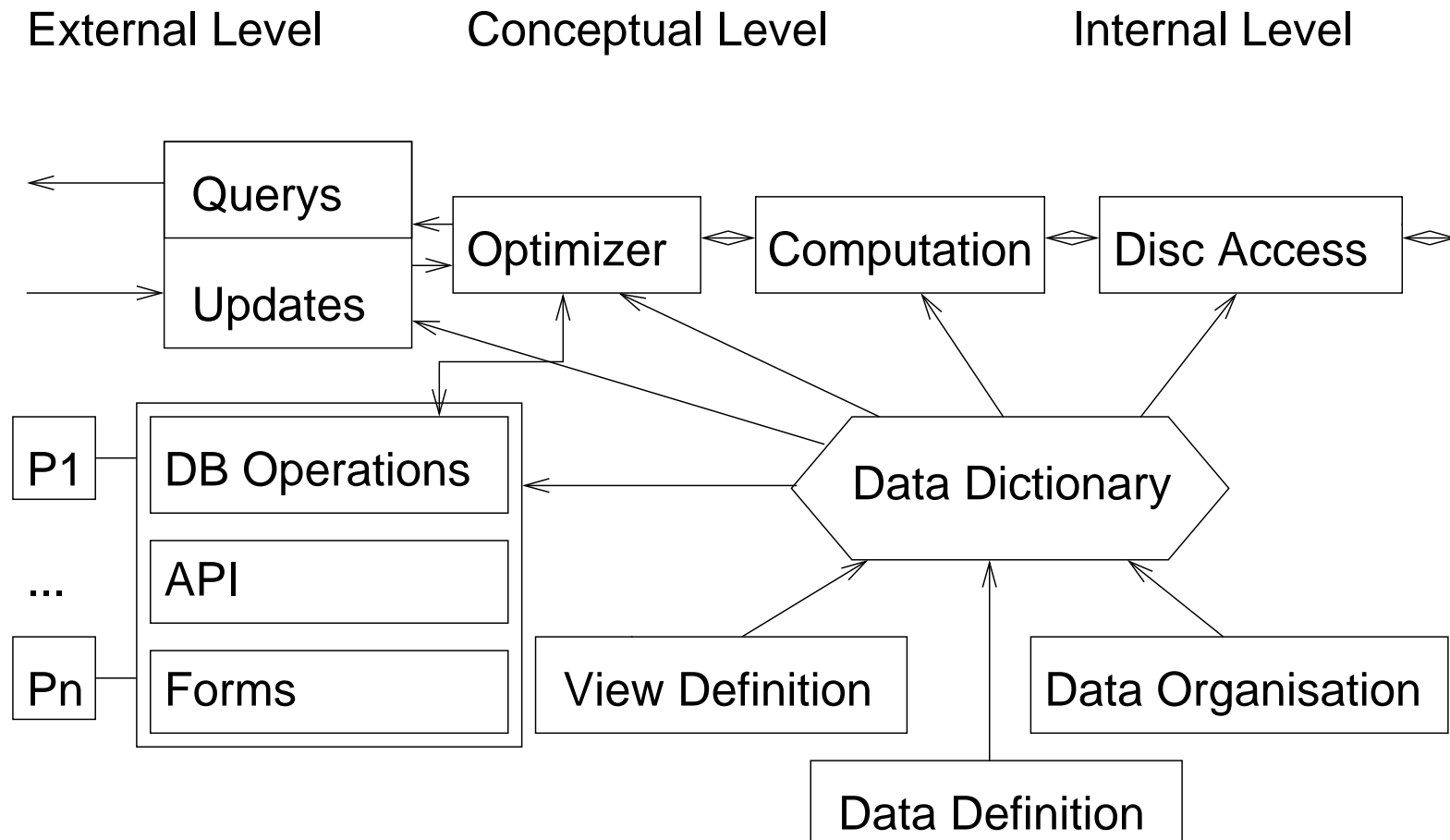
- Saake, G.; Heuer, A., Sattler, K.-U.: *Datenbanken — Implementierungskonzepte*. 2. Auflage, mitp-Verlag, Mai 2005 (*in German*)
- Härder, T.; Rahm, E.: *Datenbanksysteme — Konzepte und Techniken der Implementierung*. Springer-Verlag, 2001 (*in German*)
- Garcia-Molina, H.; Ullman, J.; Widom, J.: *Database System Implementation*. Addison-Wesley, 1999.
- Silberschatz, A.; Korth, H. F.; Sudarshan, S.: *Database System Concepts*. Wiley & Sons, 2001.
- Elmasri, R.; Navathe, S.B.: *Fundamentals of Database Systems.*, 5th Edition, Addison-Wesley, 2007.

# 1. Requirements and Principles of DBS

---

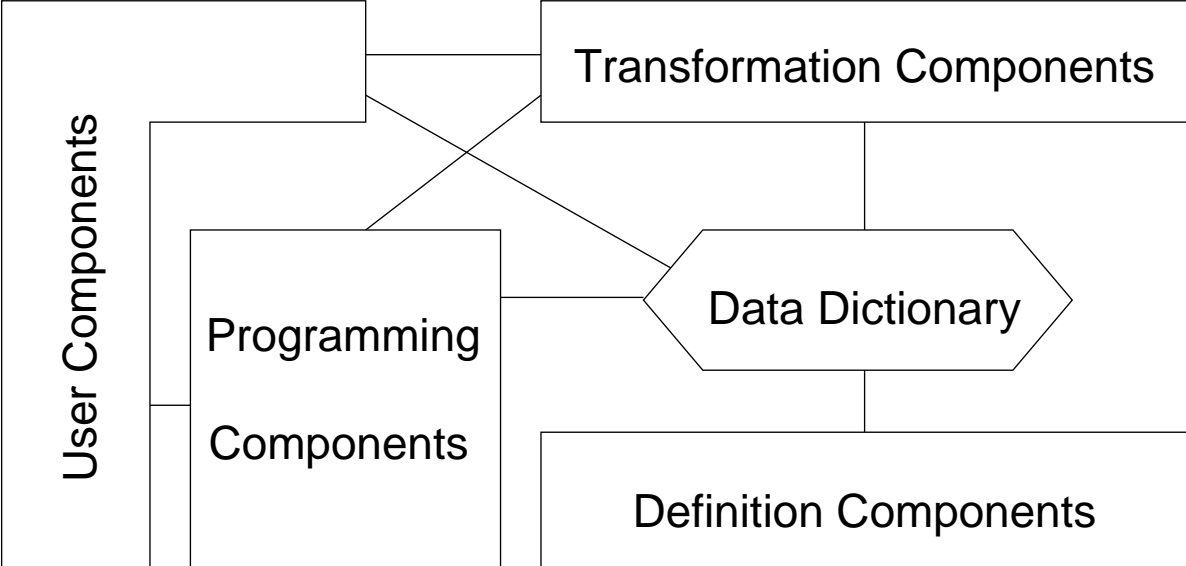
- Repetition of basic terms (of databases)
- Overview over discussed components

# Basic Terms: Components



# Classification of Components

---



# Nine Rules of Codd

---

1. Integration
2. Operations
3. *Catalog (Data Dictionary)*
4. *(Multi)User Views*
5. Consistency
6. (Data) Privacy
7. *Transactions*
8. *Synchronization*
9. *Data Backup*

# Database Models and Data Definition

---

Important models in commercial systems

- *Hierarchical model*: Data exist in a tree-like shape with hierarchical structured data sets,
- *Network model*: Support of networks of data sets, referenced by pointers,
- *Relational data model*: Data exist in tabular form,
- *Object(-oriented) model*: object-oriented modelling of data by objects, organized in classes and referenced by pointers,
- *Semi-structured model*: Management of self-describing data without schemata in graph structures (XML).

# Relational Databases

---

Borrowing	InventoryNo	Name
	4711	Brown
	1201	Smith
	0007	Miller
	4712	Brown

Book	InventoryNo	Title	ISBN	Author
	0007	Dr. No	3-125	James Bond
	1201	Object Databases	3-111	Heuer
	4711	Databases	3-765	Vossen
	4712	Databases	3-891	Ullman
	4717	Pascal	3-999	Wirth

# SQL-DDL

---

```
create table Book (  
    ISBN char(10),  
    Title varchar(200),  
    PublisherName varchar(30),  
    primary key (ISBN),  
    foreign key (PublisherName)  
        references Publishers (PublisherName) )
```

# Queries

---

## Basic principles

- Relational algebra as well as
- Tuple or range calculus.

# Relational Algebra

---

$$\sigma_{\text{Name}='Brown'}(r(\text{Borrowing}))$$
$$\pi_{\text{Title}}(r(\text{Book}))$$
$$\pi_{\text{InventoryNo,Title}}(r(\text{Book})) \bowtie \sigma_{\text{Name}='Brown'}(r(\text{Borrowing}))$$

# Alteration Component

---

This component of a database system enables,

- to insert tuples,
- to delete tuples and
- to update/change tuples.

# Languages and Views: SQL

---

```
select Book.InventoryNo, Title, Name
from   Book, Borrowing
where  Name = 'Brown' and
        Book.InventoryNo = Borrowing.InventoryNo
```

```
update Employees
set    Salary = Salary + 1000
where  Salary < 5000
```

```
insert into Book values
    (4867, 'Wissensbanken', '3-876', 'Karajan')
```

```
insert into Customer
    ( select LName, LAdr, 0 from Provider )
```

# Language Integration

---

```
exec sql declare CurBook cursor for  
    select ISBN, Title, PublisherName  
    from Book  
    for update of ISBN, Title;
```

```
exec sql commit work;
```

```
exec sql rollback work;
```

# Views in SQL

---

```
create view Browns as  
select Book.InventoryNo, Title, Name  
from Book, Borrowing  
where Name = 'Brown' and  
        Book.InventoryNo = Borrowing.InventoryNo
```

# Overview of Components (discussed in this lecture)

---

- *Optimizer*
- *Data file organisation and access paths*
- *Organisation of the secondary storage*
- *Transaction management (concurrency control)*
- *Recovery component*

# Optimizer

---

- Equivalence of algebra terms
  1.  $\sigma_{A=const} ( REL1 \bowtie REL2 )$  und  $A$  aus  $REL1$
  2.  $\sigma_{A=const} (REL1) \bowtie REL2$
- common strategy: Selections as early as possible  $\Rightarrow$  decreases the amount of tuples in relations
- Example:  $REL1$  100 tuples,  $REL2$  50 tuples  
internal: tuples are stored sequential
  1.  $5000 (\bowtie) + 5000 (\sigma) = 10000$  operations
  2.  $100 (\sigma) + 10 \cdot 50 (\bowtie) = 600$  operations  
if 10 tuples in  $REL1$  fulfill the condition  $A = const$

# Joins

---

- Merge-Join: *Join by merging* of  $R_1$  and  $R_2$ 
  - ◆ particularly efficient, if one (or both) relation(s) are ordered by the join attributes, i.e., for join attributes  $X$  has to apply:  $X := R_1 \cap R_2$
  - ◆  $r_1$  and  $r_2$  will be ordered by  $X$
  - ◆ Merging of  $r_1$  and  $r_2$ , i.e. , both relations are traversed sequentially and matched pairs are added to the result
- Nested-Loops-Join: twofold loop over  $R_1$  and  $R_2$ 
  - ◆ availability of access path for  $X$  (for one relation)  $\Rightarrow$  replacing inner loop by access via access path

# Complexity of Operations

---

## ■ Selection

- ◆ Hash-based data structures:  $O(1)$
- ◆ Sequential traversal:  $O(n)$
- ◆ Generally (tree-based access paths):  $O(\log n)$

## ■ Join

- ◆ Ordered tables exist:  $O(n + m)$  (Merge Join)
- ◆ Else: up to  $O(n * m)$  (Nested Loop Join)

## ■ Projection

- ◆ Existing access path or projection on key:  $O(n)$
- ◆ Duplicate elimination (by sorting):  $O(n \log n)$

# Types of Optimization

---

- *Logical optimization:*
    - ◆ Uses only algebraic attributes of operations
    - ◆ *NO* information about storage structures and access paths
    - ◆ Usage of heuristic rules instead of exact optimization
    - ◆ Examples:
      - Removal of redundant operations
      - Move/Shift of operations, so that Selections are executed as early as possible
- ↪ *Algebraic optimization*

# Types of Optimization II

---

- *Internal optimization:*
  - ◆ Usage of information about existing storage structures
  - ◆ Choice of implementation strategies of single operations (Merge Join vs. Nested-Loops-Join)
  - ◆ Examples:
    - Order of Joins by means of size and support of the relations by access paths
    - Order of Selections by means of selectivity of attributes and the existence of access paths

# Algebraic Optimization

---

- *Removal of redundant operations* ( $r \bowtie r = r$ )

$$r(\text{BookAway}) = \\ r(\text{Book}) \bowtie \pi_{\text{ISBN}, \text{Date}}( \\ \dots \sigma_{\text{Date} < '31-12-1990'}(r(\text{Borrowing})))$$

- ◆ Query to View:

$$\pi_{\text{Title}}(r(\text{Book}) \bowtie r(\text{BookAway}))$$

- ◆ Insertion of view definition:

$$\pi_{\text{Title}}(r(\text{Book}) \bowtie r(\text{Book}) \bowtie \pi_{\dots}(\dots))$$

# Algebraic Optimization II

---

## ■ *Shift of Selections*

$$\sigma_{\text{Author}='Vossen'}(r(\text{Book}) \bowtie \pi_{\text{ISBN,Date}}(\dots))$$

Join on downsized intermediate results:

$$(\sigma_{\text{Author}='Vossen'}(r(\text{Book}))) \bowtie \pi_{\text{ISBN,Date}}(\dots)$$

Commutate Selection and Join

# Algebraic Optimization III

---

- *Order of Joins*

$$(r(\text{Publisher}) \bowtie r(\text{Borrowing})) \bowtie r(\text{Book})$$

Drawback: first Join degenerates into cartesian product, since there are no shared attributes

$$r(\text{Publisher}) \bowtie (r(\text{Borrowing}) \bowtie r(\text{Book}))$$

$\bowtie$  associative and commutative

# Data Organisation and Access Paths

---

<b>Conceptual Level</b>		<b>Internal Level</b>		<b>Disk</b>
Relations	→	Files	→	
Tuples	→	Records	→	Blocks
(Attribute) Values	→	Fields	→	

# Access Paths

---

- Primary versus Secondary index
- Sequential files, B-Trees, hashing
- One-dimensional versus multidimensional
- Specific applications

# Transactions und Recovery

---

- **Atomicity**
  - ◆ Transaction (TA) is executed as a whole or not at all
- **Consistency**
  - ◆ The state after the execution of a transaction fulfills the integrity conditions
- **Isolation**
  - ◆ Result of a transaction has to match with an isolated execution of this TA (even if there are other, concurrent TAs)
- **Durability**
  - ◆ Results (of a TA) have to be stored persistently in the database

# Transactions II

---

The result of a transaction should look like as if it has been executed according to the ACID principle.

# Data Items and Locks

---

Locks:

```
T1 : lock A;  
      read A;  
      A := A + 1;  
      write A;  
      unlock A;
```

*Deadlocks*:

```
T1 : lock A;  
      ... i  
      lock B;  
      ... i  
      unlock A;  
      unlock B;  
T2 : lock B;  
      ... i  
      lock A;  
      ... i  
      unlock B;  
      unlock A;
```

# Serial Schedules

---

$T_1$ : <b>read</b> $A$ ;	$T_2$ : <b>read</b> $B$ ;
$A := A - 10$ ;	$B := B - 20$ ;
<b>write</b> $A$ ;	<b>write</b> $B$ ;
<b>read</b> $B$ ;	<b>read</b> $C$ ;
$B := B + 10$ ;	$C := C + 20$ ;
<b>write</b> $B$ ;	<b>write</b> $C$ ;

$T_1; T_2$  as well as  $T_2; T_1$  are serial

# Concept of Serializability

---

A schedule is called *serializable*, if the result is equivalent to the result of a serial schedule.

Methods:

- Serializability graph
- Two-Phase-Lock-Protocol (2PL)
- time-stamp procedure

# Different Schedules

Schedule $S_1$		Schedule $S_2$		Schedule $S_3$	
$T_1$	$T_2$	$T_1$	$T_2$	$T_1$	$T_2$
<b>read</b> $A$		<b>read</b> $A$		<b>read</b> $A$	
$A - 10$			<b>read</b> $B$	$A - 10$	
<b>write</b> $A$		$A - 10$			<b>read</b> $B$
<b>read</b> $B$			$B - 20$	<b>write</b> $A$	
$B + 10$		<b>write</b> $A$			$B - 20$
<b>write</b> $B$			<b>write</b> $B$	<b>read</b> $B$	
	<b>read</b> $B$	<b>read</b> $B$			<b>write</b> $B$
	$B - 20$		<b>read</b> $C$	$B + 10$	
	<b>write</b> $B$	$B + 10$			<b>read</b> $C$
	<b>read</b> $C$		$C + 20$	<b>write</b> $B$	
	$C + 20$	<b>write</b> $B$			$C + 20$
	<b>write</b> $C$		<b>write</b> $C$		<b>write</b> $C$

# Cascading Aborts

$T_1$	$T_2$
<b>lock</b> $A$ <b>read</b> $A$ $A := A - 1$ <b>write</b> $A$ <b>lock</b> $B$ <b>unlock</b> $A$	<b>lock</b> $A$ <b>read</b> $A$ $A := A \times 2$  <b>write</b> $A$ <b>unlock</b> $A$
<b>abort</b> $T_1 \rightarrow$	<b>commit</b> $T_2$
$B := B/A \downarrow$	

# Recovery

---

- *Volatile* vs. non-volatile storage
- Log book / journal
- *Backward Recovery*: *undo* changes  $\rightsquigarrow$  UNDO
- *Forward Recovery*: doing changes afterwards  $\rightsquigarrow$  REDO
- Non-addressable memory („shadow storage“)