Foundations of Information Systems (WS 2008/09)

– Chapter 5 –

Database Management Systems
Main components of a database management system (DBMS) and their interaction:
Storage manager basics

**User Interface**

- **Schema Manager**
- **Query Manager**
- **Transaction Manager**

**Storage Manager**

- **Data Dictionary**
- **Database**
- **Log**
Storage hierarchy of a computer

Access times (approx.):

- 1-10 ns
- 10-100 ns
- 100-1000 ns
- 10 ms
- sec

1 ms = 1/1000 sec
1 μs = 1/1000 ms
1 ns = 1/1000 μs

„Access gap“: factor $10^5$

This is where queries are executed!

This is where data are stored!
• Operations on data are always performed **in main memory only**!
• **Consequence**: Data to be manipulated have to be read into main store before modification („buffering“) and have to be written back sometimes later to background storage media by the DBMS.
• For the DBMS, the machine‘s main memory thus serves as a **buffer**.
General principle of **mapping relations to pages** (on disc):

- **per relation**: A file subdivided into several subsequent pages
- **per page**: An internal record table containing pointers to all tuples/records on this page
- **Adressing** individual tuples via a **tuple identifier (TID)**
- **TID** consists of two parts:
  - A page number (absolute address)
  - A pointer in the record table pointing to a particular position within the page where the respective tuple starts (relative address)
- Indirection is useful during **reorganisation** of the page.
Access paths and indexes

- DBMS normally make use of various internal auxiliary data structures in order to speed up access to larger tables. Such additional data structures are called access paths.

- The most important such access aids are called indexes, special forms of search trees offering direct access to values of particularly important and frequently accessed attributes of a table (rather than sequentially searching all fields of all rows). Often, at least the primary key attribute(s) of a table are supported by an index for rapid access.

- The notion 'index' is motivated by the concept of index pages of books. Here, an additional part at the end of the book contains keyword-page reference lists for avoiding to search for certain keywords sequentially.

- SQL offers special DDL-commands for creating and deleting indexes to one or several columns of a table:

  \[
  \text{CREATE INDEX } \text{<index-name> ON } \text{<table-name> <list-of-column-names>}
  \]

  \[
  \text{DROP INDEX } \text{<index-name>}
  \]
• Per relation, one index normally supports the primary key (often created automatically):

• In addition, several secondary indexes may exist for other frequently used columns, even if these columns/attributes do not possess the key property.

• For implementing indexes, special data structures are used by the DBMS, the choice of which can be influenced by specially authorized DB administrators in large commercial systems:
  • Search trees:
    • ISAM-files
    • B-trees and variants
    • R-trees and variants (higher-dimensional index structures)
  • Hash-tables (tables associated with special access functions)

• When using an index, there is always a „tradeoff“ between resources:
  A reduction in access time has to be measured against an increase in administration time for index maintenance and in storage used by the extra index data.
simplest form of indexing:

- Each entry on the index pages consists of values of the indexed attribute (search key S). Each entry on the data pages is a value of another attribute (data D).

- On the index pages, search key values alternate with pointers to corresponding data pages.

- Search within a page is done sequentially! Modifications on index pages may lead to high administration overhead (rebalancing of neighbour pages, redirection of pointers, creation of new pages, moving of key values).
B-tree: Idea

Generalization of the ISAM-idea to multiple levels of indexing plus mixing data and index values:

B-Tree (Bayer, McCreight 1970)

- Each node corresponds to one page in store
- Tree is balanced: equally long branches
- More than 50% of each page filled with data
A database physically consists of a selection of files accessed solely by the storage manager of the DBMS.

These files reside on large, persistent storage media, normally on discs. This part of a computer’s store is called secondary storage or background store, as opposed to its comparatively small, but fast primary or main store.

There is no such thing as rows or columns in files, but just sequences of individual symbols (actually just bits!). Thus, the DBMS has to
- code relational tuples/fields as bit sequences when storing away
- decode them again when fetching them for manipulation

There are a few more basic principles of physical structure of a database:
- The beginning of each tuple (stored as a bit sequence) is marked by a reference called tuple identifier (TID).
- Tuples are summarized into larger units called pages – when accessing the database no tuples, but entire pages are fetched.
- Pages in turn are organized in form of search trees, called indexes.
Query manager basics

User Interface

Schema Manager

Query Manager

Transaction Manager

Storage Manager

Data Dictionary

Database

Log
Query processing: Overview

- One of the most important tasks of a DBMS is efficient processing of queries.

- In principle, queries are treated like programs written in a program language, i.e., they are given to a compiler.

- The most important phase of the compilation process is the attempt of improving the efficiency of the final execution phase, called query optimization.
Query optimization: Phases

- The optimization phase tries to improve efficiency of execution, even though it not always may reach an „optimal“ solution (thus the notion is misleading).

- From an initial internal representation of the query in RA, an equivalent but more efficiently executable algebra expression is generated by equivalence preserving transformations:

  **Logical optimization**

- Afterwards a suitable implementation variant is chosen for each RA-operator. This is done by exploiting several special parameters of the operand relations such as size, selectivity, sorting, indexing etc. – often cost models are used, too.

  **Physical optimization**

- At the end of the optimization phase a query execution plan (short: QEP) is generated which can be further improved individually („by hand“) by a DB specialist when using a commercial DBMS ("database tuning").

- Strategies and heuristics used by a commercial query optimizer are often very badly documented as they constitute a „well hidden" secret of the resp. vendor.
Example of logical optimization (1)

In which semester are those students enrolled which attend courses read by professor Sokrates?

SELECT DISTINCT s.Semester
FROM Students s,
     Attends a,
     Courses c,
     Professors p
WHERE p.Name = 'Sokrates'
  AND p.PersNr = c.ReadBy
  AND a.CourseNr = c.CourseNr
  AND a.MatrNr = s.MatrNr

in SQL:
Example of logical optimization (2)

Operator tree in canonical representation of the SQL-query:

\[
\pi_{s.\text{Semester}} \\
\sigma_{p.\text{Name} = 'Sokrates' \land p.\text{PersNr} = c.\text{ReadBy} \land a.\text{CourseNr} = c.\text{CourseNr} \land a.\text{MatrNr} = s.\text{MatrNr}} \\
\times \\
\times \\
\times \\
\times \\
\times \\
\times \\
\times \\
\times
\]
Splitting and pushing selections

\( \pi \)  
\( s.\text{Semester} \)

\( \sigma \)  
\( p.\text{Name} = 'Socrates' \land \)  
\( p.\text{PersNr} = c.\text{ReadBy} \land \)  
\( a.\text{CourseNr} = c.\text{CourseNr} \land \)  
\( a.\text{MatrNr} = s.\text{MatrNr} \)

\( \text{nicht verschiebbar!} \)
Example of logical optimization (4)

Forming joins from products and selections

π

σ

p. PersNr = c. ReadBy

σ

a. CourseNr = c. CourseNr

σ

p. Name = 'Sokrates'

σ

a. MatrNr = s. MatrNr

© 2009 Prof. Dr. Rainer Manthey
Foundations of IM
Example of logical optimization (5)

Reordering joins

- $\pi$\(\sigma\)
  - $p.\text{PersNr} = c.\text{ReadBy}$
  - $a.\text{CourseNr} = c.\text{CourseNr}$
  - $a.\text{MatrNr} = s.\text{MatrNr}$

Smallest operand: Should participate in 1st join!
Example of logical optimization (6)

... all other joins are then ordered automatically.

\[
\pi \quad s.\text{Semester}
\]

\[
\sigma \quad p.\text{Name} = 'Sokrates'
\]

\[
\sigma \quad c.\text{Name} = 'Sokrates'
\]

\[
a.\text{CourseNr} = c.\text{CourseNr}
\]

\[
a.\text{MatrNr} = s.\text{MatrNr}
\]

\[
p.\text{PersNr} = c.\text{ReadBy}
\]

\[
s
\]

\[
a
\]

\[
c
\]
• **Justification** for reorderings during logical optimization:
  In each step **equivalence preserving transformations** are applied.

  e.g.: \[ \text{if } \text{attr}(p) \subseteq \text{attr}(R_1) \]
  \[ \text{and } \text{attr}(p) \cap \text{attr}(R_2) = \emptyset \]

  \[ \sigma_p(R_1 \times R_2) \Rightarrow (\sigma_p(R_1)) \times R_2 \]

• **Always:**
  • Answer set is **identical** over each DB-state before and after transformation („equivalence”).
  • Evaluation of the transformed expression is often (in many cases, not always) more efficient than before the transformation.

• **But:**
  • Transformations do **not** come with any guarantee for **efficiency gains**!
  • Transformation rules are **heuristics** („rules of thumb”), which have to be applied in a „clever“ way.

• **Therefore:**
  • Notion „optimization" is **misleading**!
  • If at all, only certain **improvements** of efficiency can be achieved, rarely a (theoretically imaginable) optimal solution will be reached.
Main tasks of physical optimization:

- Choosing suitable algorithms for physically realizing RA-operators: physical operators, physical algebra
- Exploiting existing access paths to storage structures: indexes, B-trees, hash-tables
- Sorting of intermediate results
- Creating temporary indexes and hash-tables
- Planning of intermediate storage usage
- Estimating costs of different implementation variants based on predefined cost models
Physical optimization: Example

Result of logical optimization:

\[ \pi_{s.\text{Semester}}(a.\text{MatrNr} = s.\text{MatrNr}) \]

\[ \sigma_{p.\text{Name} = 'Sokrates'}(p.\text{PersNr} = c.\text{ReadBy}) \]

QEP after physical optimization:

\[ \text{IndexDup} \]

\[ \text{Hash} \]

\[ \text{Project} s.\text{Semester} \]

\[ \text{IndexJoin} \]

\[ h.\text{MatrNr} = s.\text{MatrNr} \]

\[ \text{MergeJoin} s \]

\[ h.\text{CourseNr} = v.\text{CourseNr} \]

\[ \text{Sort} h.\text{CourseNr} \]

\[ \text{Sort} c.\text{CourseNr} \]

\[ \text{IndexJoin} a \]

\[ p.\text{PersNr} = c.\text{ReadBy} \]

\[ \text{Select} p.\text{Name} = 'Sokrates' \]
Operators of physical algebra

Each operator of RA ("logical algebra") is associated with one or more procedures realizing this operation in physical storage: Operators of physical algebra

- **Projection** without duplicate elimination:

- **Union** without duplicate elimination:

- **Duplicate elimination**:
  - "naive" basic version:
  - using sorting:
  - using an Index:

- **Selection**:
  - without using an index:
  - with using an index:
Operators of physical algebra (2)

- **Join** (analogously: Difference, intersection):
  - "naive" basic version:
  - on **sorted** input relations:
  - using an **index**:
  - using a **hash-table** in main memory:

- **Intermediate storing**:
  - with temporary **materialisation** of results relations in memory:
  - M. with **sorting** of result relation:
  - M. with **indexing** the result relation:

- **Sorting** of intermediate result relations:
Discussion of sample evaluation plan

Temporary hash-table in main memory

Primary index on s

Only part of primary key on a ⇒ no IndexJoin

No index on 'Name'

Secondary index on c

IndexDup

Hash

Project

IndexJoin

MergeJoin

Sort

IndexJoin

Select

p.Name = 'Sokrates'

p

c

s

a

CourseNr = c.CourseNr

Sort

CourseNr = c.MatrNr

CourseNr = c.CourseNr

CourseNr = a.CourseNr

IndexJoin

a.MatrNr = s.MatrNr

IndexJoin

IndexDup

Page dimensions: 595.2x842.0

© 2009 Prof. Dr. Rainer Manthey

Foundations of IM

26
Transcation manager basics

- User Interface
- Schema Manager
- Query Manager
- Transaction Manager
- Storage Manager
- Data Dictionary
- Database
- Log
Every change (update) of a database is performed in special units comprising one or more update statements (possibly combined with queries):

Transactions in SQL are marked by special keywords:
- BEGIN_OF_TRANSACTION, for starting a transaction,
- COMMIT, for successful completion, or
- ROLLBACK, for aborting a failed transaction explicitly.

Transactions are „packages“ of individual steps treated with „particular care“ by the DBMS: Transactions are always guaranteeing some form of logical and physical consistency of the database!

Individual update statements are treated like „mini-transactions“ implicitly, even without enclosing them into BEGIN-COMMIT.

The transaction manager is one of the most important components of a DBMS.
Transactions take place in many areas of real life all the time, e.g. when using a cash machine for receiving cash money or when transferring money by home banking.

**e.g.**:: Transfer 50 € from account A to account B!

```
begin_of_transaction; read(A, a); a := a -50; write(A, a);
read(B, b); b := b+50; write(B, b); commit
```

**Initial state** (consistent) **Intermediate state** (inconsistent) **Final state** (consistent)
Transaction management often takes place in a context where several competing transactions \textit{concurrently} try to access a central, jointly used database.

\textit{e.g.}: Book a seat on the next flight from Cologne to Delhi!

\begin{itemize}
  \item[$T_A$] \begin{verbatim}
begin_of_transaction;
read(free, a);
write(occupied, a);
commit
\end{verbatim}
  \item[$T_B$] \begin{verbatim}
begin_of_transaction;
read(free, b);
write(occupied, b);
commit
\end{verbatim}
\end{itemize}

What happens if there is only one seat left? – Who gets it? - What does the other one see?
Sequences of updates are reasonably handled only if they are executed entirely and without observable interruption.

Transactions expect
- Consistency of the final state (i.e., satisfaction of all integrity constraints)
- Persistence of the resulting changes
- Resistance of the DBMS against machine errors and concurrent access

If any transaction has reached a state where any of these expectations can no longer be satisfied, it has to be interrupted by the DBMS and the initial state from which this particular transaction has started has to be reconstructed:

If a transaction had to be rolled back due to external errors, the DBMS will automatically try to restart it at the next possible moment in time.

However, if the transaction has been stopped due to integrity violations it has caused itself, rollback is performed, but no restart is attempted for this transaction.
Transactions thus are characterised by the following four properties addressed together by the notion of „ACID-paradigm“:

- **Atomicity:**
  Transactions are indivisible, i.e., they are either executed entirely, or not at all („all or nothing-principle“).

- **Consistency:**
  Transactions lead from consistent initial states to consistent final states (but possibly via inconsistent intermediate states).

- **Isolation:**
  Concurrently performed transactions on the same database (in multi-user mode) do not influence each other, even though their individual steps may be interleaved by the DBMS for improving efficiency.

- **Durability:**
  The effect of a successfully completed transaction is maintained persistently in the DB, unless explicitly reversed later on by follow-up transactions.
Components of a transaction manager

Resulting from the ACID-paradigm:
three main components of a transaction manager

• **Consistency checker:**
  - Checks all affected integrity constraints after each individual step in a transaction (IMMEDIATE) or at the end of a transaction (DEFERRED).
  - Stops transaction on integrity violations and initiates rollback.

• **Scheduler:**
  - Determines a proper („non-interferring“) sequence of steps of concurrent transactions („interleaving“) using resources efficiently.
  - Maintains the „illusion“ of a sequential execution (serializability).

• **Recovery manager:**
  - Records individual steps of each transaction in a special protocol storage area, called the **DB log** („logging“).
  - Manages rollback and restart after system failures according to the log entries.
"Lost update problem"

Expl. of two unsynchronized transactions, the execution of which causes changes to be "lost":

<table>
<thead>
<tr>
<th>Step</th>
<th>(T_1)</th>
<th>(T_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>read((A,a_1))</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>(a_1 := a_1 - 300)</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>(a_2 := a_2 \times 1.03)</td>
<td>read((A,a_2))</td>
</tr>
<tr>
<td>4.</td>
<td>(b_1 := b_1 + 300)</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>write((A,a_1))</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>write((A,a_2))</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>read((B,b_1))</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>(b_1 := b_1 + 300)</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>write((B,b_1))</td>
<td></td>
</tr>
</tbody>
</table>

\(T_1\): Transferring 300 € from account A to B
\(T_2\): Paying 3% interest to account A

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>5300 €</td>
<td>4700 €</td>
</tr>
<tr>
<td>Step 2</td>
<td>5459 €</td>
<td></td>
</tr>
<tr>
<td>Step 3</td>
<td></td>
<td>5000 €</td>
</tr>
<tr>
<td>Step 4</td>
<td></td>
<td>5000 €</td>
</tr>
</tbody>
</table>
There would have been two intuitively correct solutions:

1. case: $T_1$ executed before $T_2$

<table>
<thead>
<tr>
<th>Step</th>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>read($A,a_1$)</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>$a_1 := a_1 - 300$</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>write($A,a_1$)</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>read($B,b_1$)</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>$b_1 := b_1 + 300$</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>write($B,b_1$)</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td></td>
<td>read($A,a_2$)</td>
</tr>
<tr>
<td>8.</td>
<td></td>
<td>$a_2 := a_2 \times 1.03$</td>
</tr>
<tr>
<td>9.</td>
<td></td>
<td>write($A,a_2$)</td>
</tr>
</tbody>
</table>

$T_1$: Transferring 300 € from account A to B
$T_2$: Paying 3% interest to account A

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5300 €</td>
<td>4700 €</td>
</tr>
<tr>
<td>2</td>
<td>5000 €</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5150 €</td>
<td></td>
</tr>
</tbody>
</table>
Another semantically „clean" solution:

2. case: \( T_2 \) executed before \( T_1 \)

<table>
<thead>
<tr>
<th>Schritt</th>
<th>( T_1 )</th>
<th>( T_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td></td>
<td>read(( A, a_2 ))</td>
</tr>
<tr>
<td>2.</td>
<td></td>
<td>( a_2 := a_2 \times 1.03 )</td>
</tr>
<tr>
<td>3.</td>
<td></td>
<td>write(( A, a_2 ))</td>
</tr>
<tr>
<td>4.</td>
<td>read(( A, a_1 ))</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>( a_1 := a_1 - 300 )</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>write(( A, a_1 ))</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>read(( B, b_1 ))</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>( b_1 := b_1 + 300 )</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>write(( B, b_1 ))</td>
<td></td>
</tr>
</tbody>
</table>

\( T_1 \): Transferring 300 € from account A to B
\( T_2 \): Paying 3% interest to account A
An „interleaved" execution of both transactions producing a „correct“ result:

<table>
<thead>
<tr>
<th>Step</th>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td></td>
<td>$\text{read}(A,a_2)$</td>
</tr>
<tr>
<td>2.</td>
<td>$\text{read}(B,b_1)$</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>$b_1 := b_1 + 300$</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td></td>
<td>$a_2 := a_2 \times 1.03$</td>
</tr>
<tr>
<td>5.</td>
<td>$\text{write}(B,b_1)$</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td></td>
<td>$\text{write}(A,a_2)$</td>
</tr>
<tr>
<td>7.</td>
<td>$\text{read}(A,a_1)$</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>$a_1 := a_1 - 300$</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>$\text{write}(A,a_1)$</td>
<td></td>
</tr>
</tbody>
</table>

- **$T_1$:** Transferring 300 € from account A to B
- **$T_2$:** Paying 3% interest to account A

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5300 €</td>
<td>4700 €</td>
</tr>
<tr>
<td>2</td>
<td>5459 €</td>
<td>5000 €</td>
</tr>
<tr>
<td>3</td>
<td>5159 €</td>
<td>5159 €</td>
</tr>
</tbody>
</table>

**Account Balances:**

- Account A: **5300 €**
- Account B: **4700 €**

**Updated Account Balances:**

- Account A: **5459 €**
- Account B: **5000 €**

- Account A: **5159 €**
There is much, much more to be learnt about databases and database management, ...

... but: This lecture cannot provide more than just a „taster“ of this exciting and very relevant area of information processing.

Time is over, folks!

Next week, we‘ll do a last round of „exam training“ – make sure everybody attends:

I want each of you to pass straightaway!!