Datenbanksysteme II: Overview and General Architecture

Ulf Leser
Table of Content

• Storage Hierarchy
• 5-Layer Architecture
• Overview: Layer-by-Layer
### 2010: Price versus speed

<table>
<thead>
<tr>
<th>Storage Type</th>
<th>Speed</th>
<th>Price per GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>1-10 ns/byte</td>
<td></td>
</tr>
<tr>
<td>Cache</td>
<td>10-60 ns/cache line</td>
<td></td>
</tr>
<tr>
<td>Main Memory</td>
<td>100-300 ns/block</td>
<td>~ 200 €/GB</td>
</tr>
<tr>
<td>Disk</td>
<td>10-20 ms/block</td>
<td>~ 1 €/GB</td>
</tr>
<tr>
<td>Tape</td>
<td>&gt; 1 ms/block</td>
<td>&lt; 1 €/GB</td>
</tr>
</tbody>
</table>

**Difference:**
- ~10^5
- ~10^4
## 2010: Storage Hierarchy

<table>
<thead>
<tr>
<th>Storage Type</th>
<th>Memory Size</th>
<th>Cost per GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>1-4 byte</td>
<td></td>
</tr>
<tr>
<td>Cache</td>
<td>1-4 MB</td>
<td>~ 200 € / GB</td>
</tr>
<tr>
<td>Main Memory</td>
<td>1-16 GB</td>
<td>~ 1 € / GB</td>
</tr>
<tr>
<td>Disk</td>
<td>512 GB – 1TB</td>
<td>&lt; 1€/GB</td>
</tr>
<tr>
<td>Tape</td>
<td></td>
<td>“Infinite” tape robots</td>
</tr>
</tbody>
</table>
2016: Storage Hierarchy

<table>
<thead>
<tr>
<th>Really expensive</th>
<th>Register</th>
<th>1 - 32 byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very expensive</td>
<td>Cache</td>
<td>1-16 MB</td>
</tr>
<tr>
<td>~ 7 € / GB</td>
<td>Main Memory</td>
<td>16-256 GB</td>
</tr>
<tr>
<td>~ 0,04 € / GB</td>
<td>Disk</td>
<td>1-16 TB</td>
</tr>
<tr>
<td>Tape</td>
<td></td>
<td>“Infinite” tape robots</td>
</tr>
</tbody>
</table>
Costs Drop Faster than you Think

Source: http://analystfundamentals.com/?p=88
New Players

Really expensive

Very expensive

~ 7 € / GB

~ 1 € / GB

~ 0,04 € / GB

Solid-State Disks (SSD)

1 ms / block

Disk

10-20 ms / block

Main Memory

60-300ns / block

Cache

10-100ns / cache line

Register

1-10ns / byte

Tape

sec – min
New Players

- Really expensive Register
  - 1-10 ns / byte

- Very expensive Cache
  - 10-100 ns / cache line
  - ~ 15 € / GB
  - ~ 1 € / GB
  - ~ 0.04 € / GB

- Solid-State Disks (SSD)
  - 60-300 ns / block
  - ~ 0.04 € / GB

- Disk
  - 10-20 ms / block

- Tape
  - sec – min

Characteristics

- random access \(!=\) sequential

**Hard Drive**

<table>
<thead>
<tr>
<th></th>
<th>Read [MB/s]</th>
<th>Write [MB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>121.3</td>
<td>109.3</td>
</tr>
<tr>
<td>Seq</td>
<td>41.69</td>
<td>48.05</td>
</tr>
<tr>
<td>512K</td>
<td>0.543</td>
<td>0.693</td>
</tr>
<tr>
<td>4K</td>
<td>1.004</td>
<td>0.698</td>
</tr>
</tbody>
</table>

**SSD**

<table>
<thead>
<tr>
<th></th>
<th>Read [MB/s]</th>
<th>Write [MB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>477.9</td>
<td>235.7</td>
</tr>
<tr>
<td>Seq</td>
<td>402.7</td>
<td>248.9</td>
</tr>
<tr>
<td>512K</td>
<td>30.49</td>
<td>64.67</td>
</tr>
<tr>
<td>4K</td>
<td>200.3</td>
<td>233.2</td>
</tr>
</tbody>
</table>

**RAM Disk**

<table>
<thead>
<tr>
<th></th>
<th>Read [MB/s]</th>
<th>Write [MB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>5766</td>
<td>7760</td>
</tr>
<tr>
<td>Seq</td>
<td>5649</td>
<td>7172</td>
</tr>
<tr>
<td>512K</td>
<td>657.0</td>
<td>554.8</td>
</tr>
<tr>
<td>4K</td>
<td>631.9</td>
<td>544.7</td>
</tr>
</tbody>
</table>

Storage Area Networks (SAN)

- **Dedicated subsystem** providing storage (and only storage)
- **Virtualization** of resources
- Facilitates management, storage assignment, backup etc.
Prize of Main Memory

• 2014: 1TB DRAM ~ 5000€
• 2016: Laptops with 16GB, desktops with 32GB, servers with 128GB
• Guess: 99% of all commercial databases are smaller than 100GB
New: Multi-Core with NUMA

- Modern CPUs can easily have **4-8 cores**, each 2 threads
- 4 CPUs in one server is standard
- Add hyper-threading
- **128 hardware threads**
- Future: Servers with 1000+ threads (exascale)
  - Network on a chip: Caching, routing, ...

Quelle: http://ixbtlabs.com/articles2/cpu/rmma-numa2.html
Consequences

• Dealing with memory hierarchy is core concern of DBMS
  - Speed of access
  - Durability of changes

• This lecture will mostly focus on disk versus RAM

• Similar problems for cache-RAM, disk-SSD, ...

• Differences exist
  - Block sizes
  - Heterogeneous pattern: Read/write, random-access/sequential
  - Durability
  - Error rates, long-evity
  - ...
  - Very active area of research
Table of Content

- Storage Hierarchy
- 5-Layer Architecture
- Overview: Layer-by-Layer
Overview

- Databases are complex software artifacts
- Need to be sliced into layers
- Hardware-induced layers: Memory hierarchy
- **Abstraction-induced layers:** Tuple – array – byte stream
  - Conceptual – logical – physical
  - Separation of concern
  - Information hiding
Five Layer Architecture

- **Conceptual**
  - Data Model
    - Logical Access
      - Logical
      - Data Structures
        - Buffer Management
          - Operating System
            - Physical
  - Schema, SQL
    - Records, transactions
      - Virtual blocks, arrays, locks
        - Blocks (pages)
          - Disks, blocks
Tasks

Data Model

Query optimization
Access control
Integrity constraints

Logical Access

Physical record manager
Index manager
Lock manager
Log / Recovery

Data Structures

Buffer Management

External memory

Operating System

Sort
Transaction processing
Cursor management

Block management
Caching

Transaction processing
Cursor management

External memory

Sorting
Transaction processing
Cursor management

Block management
Caching
Operations

SQL: select ... from ... Where
Grant access to ...
Create index on ...

RECORDs in pages
access paths, indexes

Disc driver
MOVE head ...

OPEN – FETCH – CLOSE
STORE Record

READ page
WRITE page

Data Model

Logical Access

Data Structures

Buffer Management

Operating System
Interfaces

Set-based access using declarative language

Record-based access using logical access path

Record-based access using physical data structures

Byte access in virtual address space

Block access (software RAID)

Disc controller (Caching, Prefetching, Hardware RAID)
Note: Idealized Representation

- Layers **may be merged**
  - E.g. logical and internal record-based layers
- Not all functionality can be assigned to exactly one layer
  - E.g. recovery, optimization
- Layers sometimes must **access non-neighboring layers**
  - Prefetching needs to know the query
    - Layer 4 to Layer 1/2
  - Optimizer needs to know about physical data layout
    - Layer 1 to layer 4/5
  - Breaks **information hiding** principle
Table of Content

• Storage Hierarchy
• 5-Layer Architecture
• Overview: Layer-by-Layer
Bottom-Up

- Data Model
- Logical Access
- Data Structures
- Buffer Management
- Operating System
Classical Discs

a) seitliche Ansicht

Zugriffs-kamm
Arm
Kopf
Spindel
Platte
Zylinder

b) Draufsicht

Sektor
Spur
Kopf
Arm
• Redundancy: **Fail-safety and access speed**
  - Increased read performance, write perf. not affected (parallel write)
  - Disc crash (one) can be tolerated
  - Be careful about dependent components (controller, power, …)

• **Drawbacks**
  - Which value is correct in *case of divergence* in the two copies?
  - Space consumption doubles

**RAID 1: Mirroring**
Access Methods: Sequential Unsorted Files

- Access to records by record/tuple identifier (RID or TID)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1522</td>
<td>Bond</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>Mason</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1754</td>
<td>Miller</td>
<td>...</td>
</tr>
</tbody>
</table>

- Operations
  - INSERT(Record): Move to end of file and add, $O(1)$
  - SEEK(TID): Sequential scan, $O(n)$
    - FIRST (File): $O(1)$
    - NEXT (File): $O(1)$
    - EOF (File): $O(1)$
  - DELETE(TID): Seek TID; flag as deleted, $O(n)$
  - REPLACE(TID, Record): Seek TID; write record, $O(n)$
    - What happens if records have variable size?
Access Methods: Sequential sorted Files

- Operations
  - **SEEK( TID):** Bin search, $O(\log(n))$
    - But a lot of random access
    - Might be slower than scanning the file
  - **INSERT( Record):** seek(TID), *move subsequent records* by one
    - This is terribly expensive – $O(n)$ reads and writes
  - ...
Indexed Files

- Operations
  - SEEK( TID): Using order in TIDs; O(log(n))
    - Only if tree is balanced; only if tree is ordered by the right value
  - INSERT( TID): Seek TID and insert; possibly restructuring
  - ...

Root
Internal Node
Leaves

Indexed Files
Storage in Oracle

- Data files are assigned to **tablespaces**
  - May consist of multiple files
  - All data from one object (table, index) are in one tablespace
  - Backup, quotas, access, …
- Extents: **Continuous sequences** of blocks on disc
- Space is allocated in extents (min, next, max, …)
- Segments logically group all extents of an object
Managing space in Oracle

Tablespace (gestrichelter Bereich)

Datenbankdateien  
Objekte (Segmente)
Bottom-Up

- Data Model
- Logical Access
- Data Structures
- Buffer Management
- Operating System

Virtual - physical blocks, access paths
Bottom-Up

Data Model

Logical Access

Data Structures

Buffer Management

Operating System

Virtual - physical blocks, access paths
Caching = Buffer Management

- Which blocks should be cached – for how long?
- Caching data blocks? Index blocks?
- **Competition**: Intermediate data, data buffers, sort buffer, …
From Buffers to Records

• Absolute addressing: TID = <PageId, Offset, ID>
  - Pro: Fast access
  - Con: Records cannot be moved

• Absolute addressing + search: TID = <PageId, ID>
  - Pro: Records can be moved within page
  - Con: Slower access
Free Space, TX, and Concurrent Processes

- Oracle procedure for finding free space
- Free space managed at the level of segments
  - Logical database objects
- Explanation
  - TFL: transaction free list
  - PFL: process free list
  - MFL: master free list
  - HWM: High water mark
Records - Blocks

- Records can be placed *arbitrarily within blocks*
  - TID need to encode the position (block …)
  - **Pro:** *Flexibility*; moving records is comparably simple
  - **Con:** Finding a record by value requires **scanning the entire file**

- Record values can determine the block in which they are stored
  - Underspecified: Which value?
  - **Pro:** Finding a record by the *distinguished value* is faster
  - **Con:** *Space management* becomes much more difficult
    - Almost empty blocks, expensive re-organizations, …
Hash-based Files

- Hash file consists of
  - Set of \( m \) buckets (one or more blocks)
  - A hash function \( h(K) = \{0, \ldots, m-1\} \) on a set \( K \) of keys;
  - A hash table (bucket directory) with pointers to buckets

- Pro: **Easier to handle** than sorted file, faster than raw file
- Contra: **Unpredictable performance**, one attribute rules
Multidimensional Shapes: R-Trees

Quelle: Geppert, Data Warehousing, VL SoSe 2002
Bottom-Up

- Data Model
- Logical Access
- Data Structures
- Buffer Management
- Operating System

Query optimization
The ANSI/SPARC Three Layer-Model

- **View**
  - Logical model: Tables, attributes, constraints, ...

- **Conceptual Schema**
- **Internal Schema**
  - Physical model: Data structures, indexes, ...

- **Data Model**
- **Logical Access**
- **Data Structures**
Query Execution

Query rewriting, view expansion

Query execution plan generation and optimization: Access paths, join order, ...

Execution of operators, pipelining
Query Processing

• **Declarative** query
  
  ```sql
  SELECT Name, Address, Checking, Balance
  FROM customer C, account A
  WHERE Name = “Bond” and C.Account# = A.Account#
  ```

• Translated in procedural **Query Execution Plan (QEP)**

  ```
  FOR EACH c in CUSTOMER DO
    IF c.Name = “Bond” THEN
      FOR EACH a IN ACCOUNT DO
        IF a.Account# = c.Account# THEN
  ```

• **Semantically equivalent**: Always compute the same result, irrespectively of the DB content
One Query – Many QEPs

SELECT     Name, Address, Checking, Balance
FROM     customer C, account A
WHERE     Name = “Bond” and C.Acco# = A.Acco#

FOR EACH c in CUSTOMER DO
    IF c.Name = “Bond” THEN
        FOR EACH a IN ACCOUNT DO

FOR EACH a in ACCOUNT DO
    FOR EACH c IN CUSTOMER DO
        IF a.Acco# = c.Acco# THEN

FOR EACH c in CUSTOMER WITH Name=“Bond” BY INDEX DO
    FOR EACH a IN ACCOUNT DO

FOR EACH c in CUSTOMER WITH Name=“Bond” BY INDEX DO
    FOR EACH a IN ACCOUNT with a.Acco#=c.Acco# BY INDEX DO

...
Query optimization

• Task: Find the (hopefully) fastest QEP
• Two interdependent levels: Best plan, best impl.
  - Different QEPs by algebraic rewriting
    • P1: $\sigma_{\text{Name}=\text{Bond}}(\text{Account} \bowtie \text{Customer})$
    • P2: $\text{Account} \bowtie \sigma_{\text{Name}=\text{Bond}}(\text{Customer})$
  - Different QEPs by different operator implementations
    • P1’: Access by scan, hash-join
    • P1’’: Access by index, nested-loop-join

• Plan space:Enumerate and evaluate (some? all?) QEPs
• Optimization goal: Minimize size of intermediate results
  - Might miss optimality in terms of runtime
    • Expansive subplan with sorted result
    • Cheap subplan with unsorted result
Rule-Based Optimizer

• Use rules-of-thumbs
  - Push selections as far as possible
  - Push projections as far as possible
  - Use indexes whenever possible
  - Always prefer sort-merge join
  - Order joins: Tables with more selections first
  - …

• Does not use information about current size of relations and indexes or distribution of values

• Does not use expected effects of operators in the query (selectivity)
Cost-Based Optimizer

• Use statistics on current state of relations
  – Size, value distribution, fragmentation, cluster factors, ...

  FOR EACH a in ACCOUNT DO
      FOR EACH c IN CUSTOMER DO
          IF a.Account# = c.Account# THEN
              IF c.Name = “BOND” THEN ...

  – Let selectivity of $\sigma_{\text{Name}=\text{Bond}}$ be 1%, $|\text{Customer}| = 10.000$, $|\text{Account}| = 12.000$, Customer:Account is 1:N
  – Performs ...
    • Join: 10.000 * 12.000 = 120M comparisons
    • Produces $\sim 12.000$ intermediate result tuples
    • Filters down to $\sim 120$ results
Cost-Based Optimizer

• Use statistics on current state of relations
  - Size, value distribution, fragmentation, cluster factors, ...

  FOR EACH c in CUSTOMER WITH Name="Bond" BY INDEX DO
    FOR EACH a IN ACCOUNT DO
      IF a.Account# = c.Account# THEN
  
  - Same setting
  - Performs
    • Reads some index blocks to find 100 customers
      - But these are read using random access
    • Join: 100*12.000 = 1.2M comparisons
    • Produces 120 results
Join methods

• Suppose the previous query would contain no selection
• Can’t we do better than “Join: 120M comparisons”
• Join methods
  - Nested loop join: $O(m \times n)$ key comparisons
  - Sort-merge join
    • First sort relations in $O(n \times \log(n) + m \times \log(m))$
    • Merge results in $O(m + n)$
    • Sometimes better, sometimes worse
  - Hash join, index-join, grace-join, zig-zag join, ...
• Note: Complexity here measures number of comparisons
  - This is a “main-memory” viewpoint
  - Must not be used for IO tasks
Data Dictionary

- Query execution needs metadata: **Data dictionary**
  - Semantic parsing of query: Which relations exist?
  - Which indexes exist?
  - Cardinality estimates of relations?
  - Size of buffer for in-memory sorting?
  - ...

<table>
<thead>
<tr>
<th>Table_name</th>
<th>Att_name</th>
<th>Att_type</th>
<th>size</th>
<th>Avg_size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>Name</td>
<td>Varchar2</td>
<td>100</td>
<td>24</td>
</tr>
<tr>
<td>Customer</td>
<td>account#</td>
<td>Int</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Customer</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Access Control

• Read and write access on objects
• Read and write access on system operations
  - Create user, kill session, export database, ...
• GRANT, REVOKE Operations
• Example:
  
  \[
  \text{GRANT ALL PRIVILEGES ON ACCOUNT TO Freytag WITH GRANT OPTION}
  \]
• No complete protection
  - Granularity of access rights usually relation/attribute – not tuple
    - Use views, label-based access control
    - Access to data without DBMS (at OS level)
    - Complement with file protection, encryption of data
Bottom-Up

- Transactions, serializability, recovery
- Data Model
  - Logical Access
  - Data Structures
  - Buffer Management
  - Operating System
Transactions (TX)

• Transaction: “Logical unit of work”
  
  Begin_Transaction
  
  UPDATE ACCOUNT
  
  SET Savings = Savings + 1M
  SET Checking = Checking - 1M
  WHERE Account# = 007;
  
  INSERT JOURNAL <007, NNN, "Transfer", ...>

  End_Transaction

• ACID properties
  - Atomic execution
  - Consistent DB state after commits
  - Isolation: No influence on result by concurrent TX
  - Durability: After commit, changes are reflected in the database
Lost Update Problem

Deposit $1,000
Read account value
5,000
Add $1,000
6,000
Write back

Deposit $2,000
Read account value
5,000
Add $2,000
7,000
Write back
Synchronization and schedules

\begin{align*}
T_1 & : \text{read } A; \\
A := A - 10; & \\
\text{write } A; & \\
\text{read } B; & \\
B := B + 10; & \\
\text{write } B; & \\
T_2 & : \text{read } B; \\
B := B - 20; & \\
\text{write } B; & \\
\text{read } C; & \\
C := C + 20; & \\
\text{write } C; &
\end{align*}

<table>
<thead>
<tr>
<th>Schedule $S_1$</th>
<th>Schedule $S_2$</th>
<th>Schedule $S_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>$T_2$</td>
<td>$T_1$</td>
</tr>
<tr>
<td>read $A$</td>
<td></td>
<td>read $A$</td>
</tr>
<tr>
<td>$A - 10$</td>
<td></td>
<td>$A - 10$</td>
</tr>
<tr>
<td>write $A$</td>
<td></td>
<td>read $B$</td>
</tr>
<tr>
<td>$B + 10$</td>
<td></td>
<td>$B - 20$</td>
</tr>
<tr>
<td>write $B$</td>
<td></td>
<td>write $B$</td>
</tr>
<tr>
<td>read $B$</td>
<td></td>
<td>read $C$</td>
</tr>
<tr>
<td>$B - 20$</td>
<td></td>
<td>$B + 10$</td>
</tr>
<tr>
<td>write $B$</td>
<td></td>
<td>$C + 20$</td>
</tr>
<tr>
<td>read $C$</td>
<td></td>
<td>write $C$</td>
</tr>
<tr>
<td>$C + 20$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>write $C$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Synchronization and locks

• When is a schedule „fine“?
  – When it is serializable
  – I.e., when it is equivalent to a serial schedule
  – Proof serializability of schedules
• Strategy: Blocking everything is dreadful
• Strategy: Checking after execution is wasteful
• Synchronization protocols
  – Guarantee to produce only serializable schedules
  – Require certain well-behavior of transactions
    • Two phase locking, multi-version synchronization, timestamp synchronization, ...
• Be careful with deadlocks
Recovery – Broad Principle

- Store data redundantly: Save old values
- Different formats for different access characteristics
So many managers ...
Oracle processes

<table>
<thead>
<tr>
<th>Process</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMS</td>
<td>Lock manager (only clustered dbs)</td>
</tr>
<tr>
<td>RECO</td>
<td>Recovery of distributed transactions</td>
</tr>
<tr>
<td>PMON</td>
<td>Control and restart of all processes</td>
</tr>
<tr>
<td>SMON</td>
<td>Recovery at start-up after failure</td>
</tr>
<tr>
<td>CKPT</td>
<td>Checkpointing</td>
</tr>
<tr>
<td>ARC0</td>
<td>Archiving of Redo-Log data</td>
</tr>
<tr>
<td>DBW</td>
<td>Writing of database blocks</td>
</tr>
<tr>
<td>LGW</td>
<td>Writing of Redo-Log blocks</td>
</tr>
<tr>
<td>D</td>
<td>Dispatcher für multithreaded servers</td>
</tr>
</tbody>
</table>