Datenbanksysteme II: Overview and General Architecture

Ulf Leser
Table of Content

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

1 – 8 Byte
Compiler

Few MB
Cache-Controller

1-16 GB
Operating system

512GB – 1TB
Disc controller, OS

“Infinite”
Roboter

Tape

Disk

Main Memory

Cache

Register

Really expensive

Very expensive

≈ 200 € / GB

≈ 1 € / GB

< 1€/GB
Storage Hierarchy Today

- Really expensive
  - Register
  - Compiler

- Very expensive
  - Cache
  - Few MB Cache-Controller

- ~ 15 € / GB
  - Main Memory
  - 16-256 GB Operating system

- ~ 0.04 € / GB
  - Disk
  - 1-16 TB Disc controller, OS

- Tape
  - “Infinite” Roboter
Costs Drop Faster than you Think

Source: http://analystfundamentals.com/?p=88
Price versus Access Time

- **Really expensive**
  - Register
  - 1-10ns / byte

- **Very expensive**
  - Cache
  - 10-60ns / cache line

- **Medium expensive**
  - Main Memory
  - 100-300ns / block

- **Inexpensive**
  - Disk
  - 10-20 ms / block

- **Cheapest**
  - Tape
  - sec - min

- **Prices**
  - ~ 15 € / GB
  - ~ 0.04 € / GB

Differences

- Really expensive
- Very expensive
- ~ 15 € / GB
- ~ 0.04 € / GB

Difference ~10^4
New Players

- Really expensive
  - Register: $1-10$ ns / byte
- Very expensive
  - Cache: $10-100$ ns / cache line
  - Main Memory: $60-300$ ns / block
- Solid-State Disks (SSD): $1$ ms / block
  - Approximate cost: $1$ € / GB
- Disk: $10-20$ ms / block
  - Approximate cost: $0.04$ € / GB
- Tape: sec – min
  - Approximate cost: $15$ € / GB
New Players

Really expensive

Very expensive

~ 15 € / GB

~ 1 € / GB

~ 0.04 € / GB

Characteristics

Quelle: http://blog.laptopmag.com/faster-than-an-ssd-how-to-turn-extra-memory-into-a-ram-disk
NUMA

Quelle: http://ixbtlabs.com/articles2/cpu/rmma-numa2.html
Table of Content

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

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

- **Data Model**
  - Conceptual: Relations, tuple, attributes, values

- **Logical Access**
  - Logical: DB objects, records, untypes fields

- **Data Structures**

- **Buffer Management**

- **Operating System**
  - Physical: Files, blocks, sectors
Operations

- SQL: select ... from ... Where Grant access to ... Create index on ...
- RECORD auf Seiten Aktualisierung Zugriffspfade Indexstrukturen
- Disc driver MOVE head ...

OPEN – FETCH – CLOSE
STORE Record

READ page
WRITE page

Data Model
Logical Access
Data Structures
Buffer Management
Operating System
Tasks

Data Model

Logical Access

Data Structures

Buffer Management

Operating System

Query optimization
Access control
Integrity constraints

Physical record manager
Index manager
Lock manager
Log / Recovery

External memory

Sort
Transaction processing
Cursor management

Block management
Caching

Schnittstellen

- Mengenorientierte Schnittstelle: Zugriff auf Tupelmengen über deklarative Sprache
- Satzorientierte Schnittstelle: Zugriff auf typisierte Tupel und logischer Zugriffspfade
- Interne Satzschnittstelle: Zugriff auf Bytearrays und reale Zugriffspfade
- Systempufferschnittstelle: Zugriff im virtuellem Adressraum
- Dateischnittstelle: Physikalische Blöcke
  - Software RAID
- Geräteschnittstelle: Festplatten
  - Cache des Controllers, Prefetching
  - Hardware RAID
Idealisierte Darstellung

• Schichten können zusammengefasst werden
  – Z.B. Schicht satzorientierte und interne Satzschnittstelle
• Manche Themen nicht klar zuordenbar
  – Z.B. Recovery, Optimierung
• Schichten müssen durch andere „hindurch greifen“
  – Caching benötigt für Prefetching Informationen über aktuelle Workload, nicht nur über das aktuelle Tupel
    • Von Schicht 4 zu Schicht 1/2
    • Eventuell sogar von Schicht 5 zu Schicht 1/2
  – Optimierer benötigt Informationen über physische Verteilung von Blöcken auf Disc
    • Von Schicht 1 zu Schicht 4/5
• Verletzung des „Information Hiding“ Prinzips
Table of Content

- Storage Hierarchy
- 5-Layer Architecture
- Overview: Layer-by-Layer
Magnet- bzw. Festplattenspeicher

a) seitliche Ansicht

b) Draufsicht

Zugriffskamm Arm Kopf Spindel Platte

Zylinder

Sektor Spur Kopf Arm
RAID 1: Mirroring

- Datensicherheit und Geschwindigkeit durch Redundanz
- Einfache Redundanz schlecht – wer hat Recht bei Fehlern?
- Doppelter Speicherbedarf
- Lastbalancierung beim Lesen
- Beim Schreiben müssen beide Kopien geschrieben werden
  - Kann aber parallel geschehen – kein Zeitverlust
Bottom-Up

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

Records, Blocks, Files
Access Methods: Sequential 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?
Index 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
  - DELETE(TID): Seek TID and remove; possibly restructuring
  - REPLACE(TID): Seek TID and write
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
Caching = Buffer Management

• Which blocks should be cached – for how long?
• Caching data blocks? Index blocks? Temporary space?
• Allocation of “space per usage” may have large effects
Record Addressing

• How to address a record?
  - Absolute addressing: TID = <PageId, Offset>
  - Absolute addressing + search: TID = <PageId>
Life is complex

- **Oracle procedure for finding free space**
- Free space administered 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
Hashing

- 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
- Structured according to one attribute value only
R-Baum Beispiel

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

Data Model

Logical Access

Data Structures

Buffer Management

Operating System

Query optimization
Drei Schichten-Modell

- **Sichten**
  - **Logisches Modell** (Tabellen, Attribute, …)
  - **Physisches Modell** (Indexierung, Speicherung)

- **Konzeptionelles Schema**
- **Internes Schema**
- **Datenmodellebene**
- **Logischer Zugriff**
- **Speicherstrukturen**
Anfragen

Umschreiben der Anfragen; Viewexpansion

Übersetzung der Anfrage Optimierung (Zugriffspfade, Indexwahl, Joinreihenfolge, )

Benutzung Zugriffspfade / operatoren / ... zur Ausführung des Plans
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)**

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

• Both must be semantically equivalent
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
    END IF
END FOR

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

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

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

...
Query optimization

- Task: Find the (hopefully) **fastest QEP**
- Two levels: Plan structure, physical operators
  - Different **query formulations** by rewriting algebra terms
    - P1: $\sigma_{Name=Bond}(Account \bowtie Customer)$
    - P2: $Account \bowtie \sigma_{Name=Bond}(Customer)$
  - Different **query execution plans** by different op implementations
    - P1': Access by scan, hash-join
    - P1'': Access by index, nested-loop-join
- Enumerate and evaluate (some? all?) QEPs
- Optimization goal: Minimize **size of intermediate results**
  - Might miss optimality
    - 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
  - Prefer sort-merge join
  - ...

- Does not use information about current size of relations and indexes
- Does not use or expected effects of operators in the query (**selectivity**)
- Does not help for join order
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%, $|\text{Customer}|=10,000$, $|\text{Account}|=12,000$
  • Performs …
    • Join: 10,000 * 12,000 = 120M comparisons for join
    • Produces ~12,000 result tuples
    • Filters down to ~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

- Let selectivity of $\sigma_{\text{Name}=\text{Bond}}$ be 1%, $|\text{Customer}|=10.000$, $|\text{Account}|=12.000$
- Performs
  • Reads some index blocks to find 100 customers
    - But these are read using random access
  • Join: $100 \times 12.000 = 1.2M$ comparisons
  • Produces 120 results
Join methods

- Suppose the previous query would contain no selection
- Can’t we do better than “Join: 120M comparisons”
- **Nested loop join** has complexity $O(m*n)$
- Other methods
  - **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 not
  - **Hash join**, ...
- **Note**: Complexity here measure **number of comparisons**
  - This is “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
  - Virtual Private Database, Label-based access control
  - Access to data without DBMS
  - Complement with file protection, encryption of data
Transactions (TA)

- **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 updates**
  - **Isolation**: No influence on result by concurrent executions
  - **Durability**: Updates 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

<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>$T_1$</td>
<td>$T_1$</td>
</tr>
<tr>
<td>read $A$</td>
<td>read $A$</td>
<td>read $A$</td>
</tr>
<tr>
<td>$A - 10$</td>
<td>$A - 10$</td>
<td>$A - 10$</td>
</tr>
<tr>
<td>write $A$</td>
<td>write $B$</td>
<td>write $B$</td>
</tr>
<tr>
<td>read $B$</td>
<td>read $B$</td>
<td>write $B$</td>
</tr>
<tr>
<td>$B + 10$</td>
<td>$B + 10$</td>
<td>$B + 10$</td>
</tr>
<tr>
<td>write $B$</td>
<td>write $B$</td>
<td>write $B$</td>
</tr>
<tr>
<td></td>
<td>read $C$</td>
<td>read $C$</td>
</tr>
<tr>
<td></td>
<td>$C + 20$</td>
<td>$C + 20$</td>
</tr>
<tr>
<td></td>
<td>write $B$</td>
<td>write $B$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$C + 20$</td>
</tr>
<tr>
<td></td>
<td>write $C$</td>
<td>write $C$</td>
</tr>
</tbody>
</table>
Synchronization and locks

- When are two schedules „fine“?
  - When they are serializable
  - I.e., when they are equivalent to a serial schedule
  - Proof serializability of schedules
- Checking after execution is wasteful
  - Synchronization protocols
  - Guarantee 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

- LMS  Lock manager (only clustered dbs)
- RECO Recovery of distributed transactions
- PMON Control and restart of all processes
- SMON Recovery at start-up after failure
- CKPT Checkpointing
- ARC0 Archiving of Redo-Log data
- DBW Writing of database blocks
- LGW Writing of Redo-Log blocks
- D Dispatcher für multithreaded servers