Physical storage in DBMS

Chapter content:

- Storage : DBMS adapts to hardware (Reminder)
- Reads vs Writes: storage architectures
- Techniques for handling updates
- Physical storage in RDBMS

Table of content

New DB technologies and trends

Data storage and processing evolutions:

- New hardware (FPGA, GPGPU, SSD, NVRAM)
- In-memory DB/ Column storage
- NoSQL: (shared nothing) massively parallel architecture (Map/Reduce, key/value)

Trends emphasize

- supporting both *analytical* (OLAP, mining) and *prediction* queries (statistical ML)
- ACID in distributed data processing
- trust management – so far, more of an issue for transactions rather than storage/analysis (blockchain)
- integrating new (external) types of data (text, feeds, images, video...)
- building data summaries (data too large to be stored): sampling, etc.
- real-time DW

Hardware: data storage

The hard drive

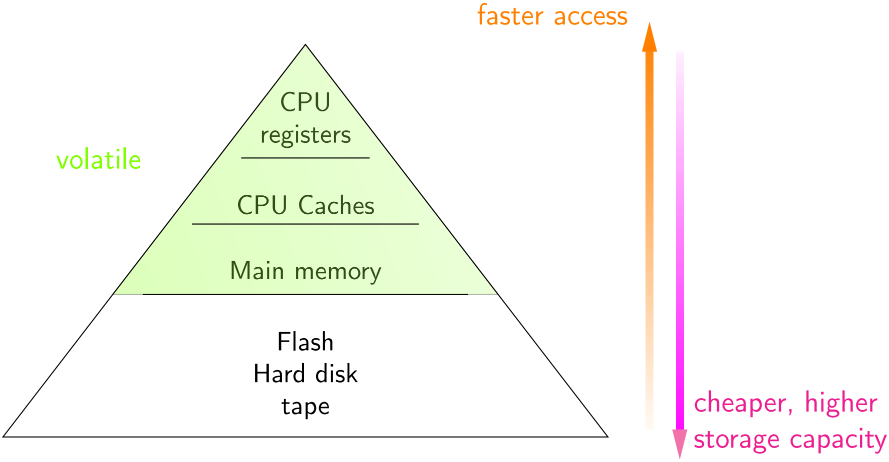
Created in 1956 (IBM).
Multiple disks (platters) spinning rapidly together (thousands rpm/min).
Ferromagnetic. Multiple tracks per platter. 1 head per platter, on air cushion.

$\approx$ 100€
a few TB
100gr.

Main storage device (secondary storage) since 1960.
The support for which DBMS were devised.

Hardware: storage

Memory hierarchy:

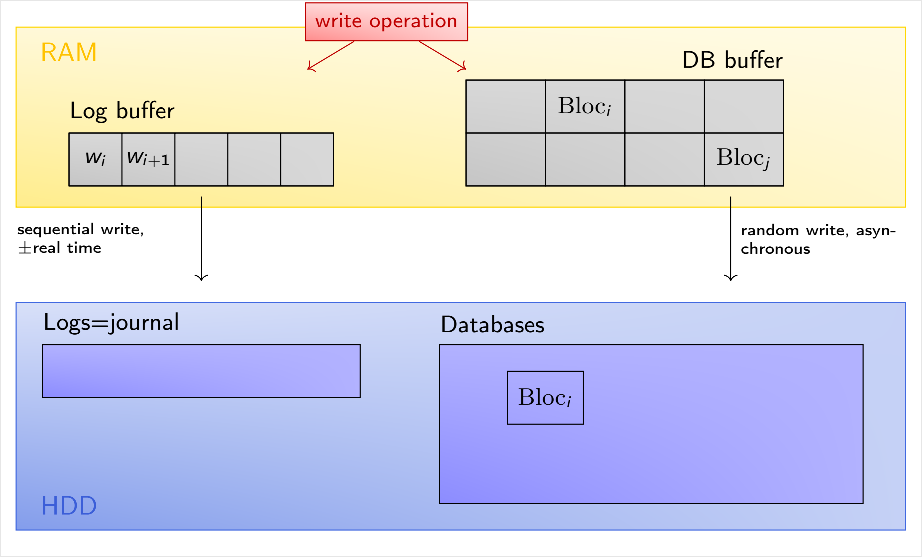


On early computers, CPU frequency $\approxeq$ memory bus and memory access.
But CPU became much faster than memory.

Table of content

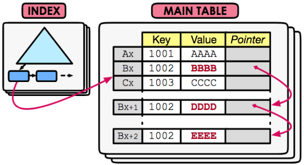
Physical storage in DBMS

Handling writes in DBMS

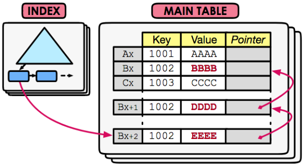


Architecture types (MVCC) in terms of tables

O2N : oldest-to-newest
N2O : newest-to-oldest

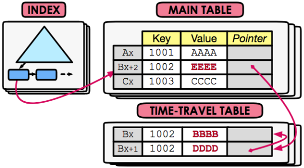


(a) Append-only (O2N)



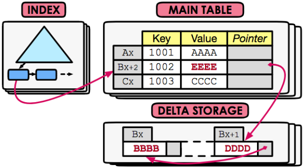
(b) Append-only (N2O)

NUODB



(c) Time-travel Storage

SAP



(d) Delta Storage

ORACLE MariaDB

http://www.vldb.org/pvldb/vol10/p781-Wu.pdf (VLDB'17)

Table of content

How storage architectures deal with updates

- *update-in-place* storage: most relational DB. Limitations: versioning (SCD 2) results in fragmentation. Lock on page during update.
  Strategy often used with B-tree index. Provides optimal reads (≊1 seek per read/1 per scan if unfragmented). But updating index slower.

- *log-structured* storage: all updates appended as a log (no "main").
  Limitations: scans get expensive: must read all logs. But writes faster than in B-tree (and no lock).
  A popular version: *LSM-tree* (Log-Structured Merge tree).
- *delta* with main store: main store is read-optimized, updates recorded in write-optimized buffer *(SAP Hana/Sans Souci, and other column stores)*. Buffer merged from time to time.

↪ **Rationale for log-structured storage: instead of writing a full page on each update, defer and process them in batch. Idea of *deferred writes* common with delta approach.**

LSM-tree

Storage optimized for write-throughput (as every log-structured storage):

- LSM-tree consists of 2 or more layers $C_0, C_1(, C_2 \ldots, C_k)$.
- C_0 in-memory, can be tree, hash table...
- others $(C_1, \ldots)$ are B-trees, recorded on disk/slower memory.
- updates written only in C_0
- $\forall i$ when C_i is full, we merge it into C_{i+1} (C_i is emptied)
  $\hookrightarrow$ $\forall i$ versions in C_i always more recent than in C_{i+1}
  $\hookrightarrow$ at most r versions of record co-exist.
  $\hookrightarrow$ a Read must search layers $C_0, C_1 \ldots$ until item found.

...but we did not detail what "full" means:

Original version: [O'Neill et al. 1996]

- recommend using geometric progression: $|C_{i+1}| = r \times |C_i|$
  $\hookrightarrow$ total number of unique keys: $N = O(r^k)$.
- but claimed inefficient (does not exploit write locality)

LSM-tree: assets and drawbacks (compared to Btree)

Buffered writes makes writes faster.

Pros:

- ✔ write operations are faster (sequential)
- ✔ lower fragmentation (so range queries get more efficient)

Weaknessess:

- ✘ accessing a given value may be slow

cassandra mongoDB APACHE HBASE SQLite LEVELDB

References

Rappels simples sur le stockage dans les SGBDs

- *http://sys.bdpedia.fr/*

MVCC storage

- *http://www.vldb.org/pvldb/vol10/p781-Wu.pdf*

LSM trees

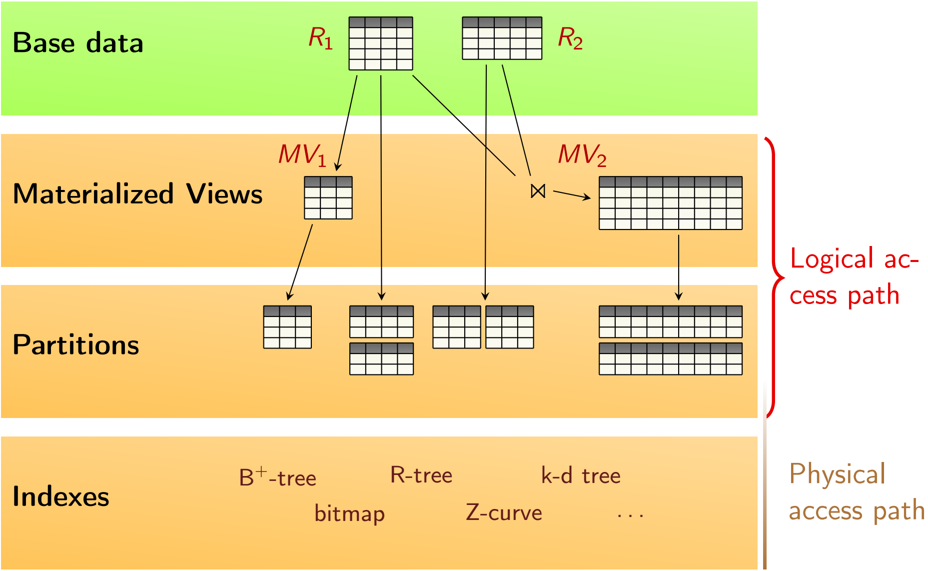
- *http://www.vldb.org/pvldb/vol10/p1526-bocksrocker.pdf*
- *http://www.eecs.harvard.edu/~margo/cs165/papers/gp-lsm.pdf*
- *https://www.quora.com/How-does-the-Log-Structured-Merge-Tree-work*

Table of content

3 levels of design

Conceptual design	User-oriented description, independant of implementation *E-R, UML*
Logical design	Logical description, independant of DBMS *Relational model: table schema*
Physical design	Actual database structures *materialized views, partitions, indexes*

Accessing data



: logical schema (ANSI SPARC architecture)

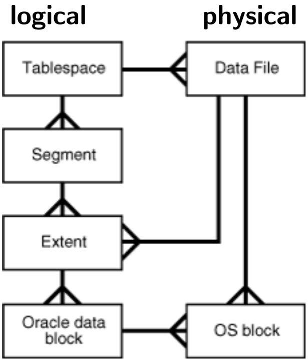
: physical schema (ANSI SPARC architecture)

Physical storage in DBMS (reminder)

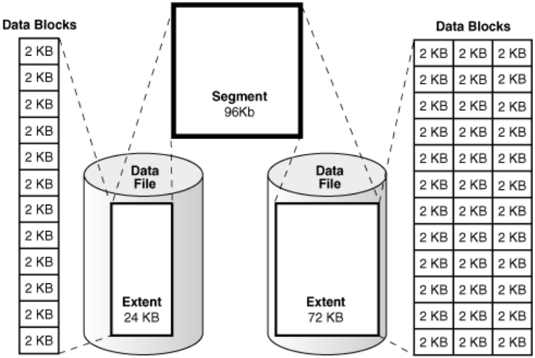
- physical address of record provided by ROWID
- Records stored in pages=blocks.
- Read/Write operations require the corresponding data be brought in buffer (main memory).
- DBMS must minimize pages I/O for performance.

larger block size: more tuples edited per page I/O, but fewer pages in main-memory.

Physical storage: Oracle (1)



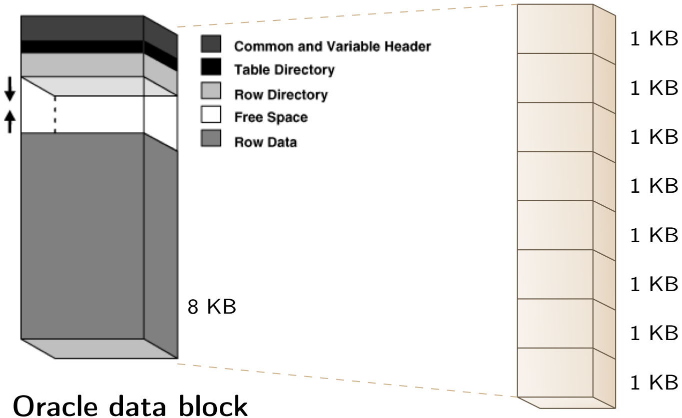
E-R diagram of storage structures



logical storage

[*Oracle Database Concepts*]

Physical storage: Oracle (2)



Oracle data block

O.S. blocks

References

Oracle Database Concepts
https://docs.oracle.com/database/121/CNCPT/logical.htm

https://en.wikibooks.org/wiki/Oracle_and_DB2,_Comparison_and_Compatibility/Storage_Model/Physical_Storage/Summary

ftp://ftp.software.ibm.com/software/data/db2/9/labchats/20110331-slides.pdf
(a bit outdated: 2011...)

http://www.postgresql.org/docs/9.4/static/storage-page-layout.html