Column databases

Chapter content:

2018-2019

- In-memory databases: commercial systems: MariaDB
- In-memory databases: commercial systems: Oracle
- In-memory databases: commercial systems: IBM DB2

Column-oriented DB

Column storage vs Row storage

Idea: modify traditional storage for relation R. In consecutive memory, instead of storing a row we store a column (values of one attribute $R.x$).

Assets:

- ✔ only accesses relevant attributes
- ✔ potentially drastic speedup at query-time, esp. aggregation
- ✔ better compression techniques (values from same domain: many identical)
- ✔ allows vectorization (bitwise, SIMD)

Weaknesses:

- ✘ need fast tuple reconstruction
- ✘ slower on `select *`
- ✘ updates (insertions, deletions...) are harder

$\Longrightarrow$ analytical workloads, mostly reads, large data

Column-oriented DB

Dictionary encoding

⚠ *The following slides describe the Sanssouci prototype architecture. Similar ideas (dictionary, buffering updates) apply in other column-oriented DB but with significant variations.*

Objective: reduce main memory operations through (lightweight) compression.

recID	fname
...	...
24	Jean
25	Pierre
26	Marie
27	Ali
28	Marie
...	...

⇒

Dictionary for "fname"

valueID	value
...	...
11	Ali
12	Jean
13	Marie
14	Pierre
...	...

Attribute vector for "fname"

position	valueID
...	...
24	12
25	14
26	13
27	11
28	13
...	...

[Plattner, in-memory databases course]

The dictionary is sorted $\Longrightarrow$ fast lookup of id from value, fast range queries.

Column-oriented DB

Compression

valueID is already a "compressed" representation of value. But we can compress attribute vector (e.g.: RLE).

Column	Cardi-nality	Bits Needed	Item Size	Plain Size	Size with Dictionary (Dictionary + Column)	Compression Factor
First names	5 millions	23 bit	50 Byte	400GB	250MB + 23GB	≈ 17
Last names	8 millions	23 bit	50 Byte	400GB	400MB + 23GB	≈ 17
Gender	2	1 bit	1 Byte	8GB	2b + 1GB	≈ 8
City	1 million	20 bit	50 Byte	400GB	50MB + 20GB	≈ 20
Country	200	8 bit	47 Byte	376GB	9.4kB + 8GB	≈47
Birthday	40000	16 bit	2 Byte	16GB	80kB + 16GB	≈ 1
Totals			**200 Byte**	**≈ 1.6TB**	**≈ 92GB**	**≈ 17**

[Plattner, in-memory databases course]

Dealing with updates

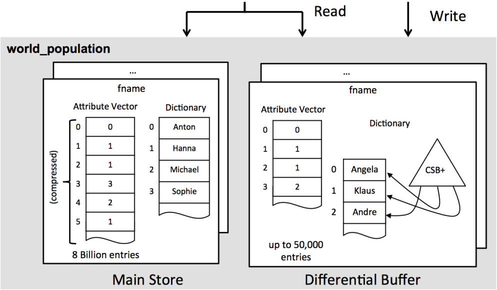
The differential buffer

Inserting a value may:

- have no impact on dictionary
- add a value at the end of dictionnary (#bits may change)
- force a dictionary reorganization (sorted dict)

$\hookrightarrow$may reorganize the whole attribute vector (same for deletions).

$\implies$ we keep the main store *read-only* . Perform insert, update, delete on the differential buffer only.



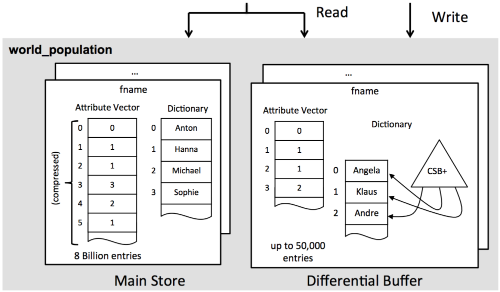
[Plattner, in-memory databases course]

The differential buffer

Queries will check both the *compressed main store* and *differential buffer*.
The differential buffer:

- records updates
- is kept small (periodically merged into the main store and emptied)
- uses column storage but with *unsorted* dictionary
- an index (CSB+-tree) is maintained on the dictionary

A validity attribute is added to tuples (uncompressed bit vector in main store). . .



[Plattner, in-memory databases course]

The differential buffer: updates

Validity attribute: uncompressed bit vector in main store.

Dealing with updates/deletions:

- we update validity attribute in main store
- insert corresponding tuple in differential buffer

recId	fname	lname	gender	country	city	birthday	valid
0	Martin	Albrecht	m	GER	Berlin	08-05-1955	1
1	Michael	Berg	m	GER	Berlin	03-05-1970	0
2	Hanna	Schulze	f	GER	Hamburg	04-04-1968	1
3	Anton	Meyer	m	AUT	Innsbruck	10-20-1992	1
4	Ulrike	Schulze	f	GER	Potsdam	09-03-1977	1
5	Sophie	Schulze	f	GER	Rostock	06-20-2012	1
...	...	...	...	...	...	...	
8×10^9	Zacharias	Perdopolus	m	GRE	Athen	03-12-1979	1

Main Store

recId	fname	lname	gender	country	city	birthday	valid
0	**Michael**	**Berg**	**m**	**GER**	**Potsdam**	**03-05-1970**	**1**

Differential Buffer

Michael Berg moves to Potsdam

[Plattner, in-memory databases course]

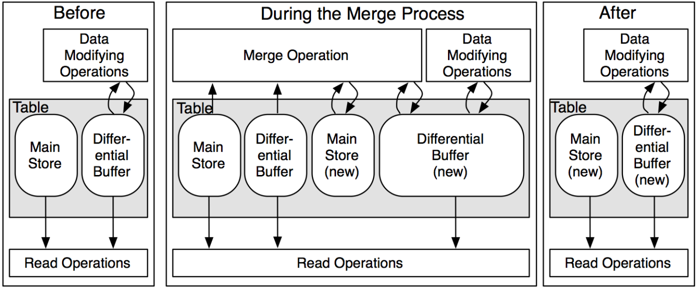
One possible way to deal with deletions: do not delete: keep validity interval (like scd type 2) $\Longrightarrow$ `insert-only` approach.

Merging the differential buffer and main store: architecture

Data in main store takes less space (compression) and benefits from faster reads (sorted) $\Longrightarrow$ keep differential buffer small $\Longrightarrow$ merge process.

We first create a second (empty) differential buffer: updates during (and after) the merge are directed toward that new buffer.

- ✔ Advantage of working on copies: short lock.
- ✘ Drawback: needs dedicated resources ($2\times$ space).



[Plattner, in-memory databases course]

Merge process: 1) combine dictionaries 2) compute new attribute vector.

The Merge process

Merge process: 1) combine dictionaries 2) compute new attribute vector.

Main Store

Dictionaries

valueID	fname	city
0	Albert	Berlin
1	Michael	London
2	Nadja	

Attribute vectors

recID	fname	city
0	2	0
1	1	1
2	0	0

Validity vector

recID	valid
0	0
1	0
2	1

Differential Buffer

Dictionaries

valueID	fname	city
0	Michael	Berlin
1	Nadja	Potsdam
2	Hanna	Dresden

Attribute vectors

recID	fname	city
0	0	0
1	1	1
2	0	1
3	2	2

Validity vector

recID	valid
0	0
1	1
2	1
3	1

The Merge process (2)

Merge process: 1) combine dictionaries 2) compute new attribute vector.

Main Store (new)

Dictionaries

valueID	fname	city
0	Albert	Berlin
1	Hanna	Dresden
2	Michael	London
3	Nadja	Potsdam

Attribute vectors

recID	fname	city
0	3	0
1	2	2
2	0	0
3	2	0
4	3	3
5	2	3
6	1	1

Validity vector

recID	valid
0	0
1	0
2	1
3	0
4	1
5	1
6	1

Differential Buffer (new)

Dictionaries

valueID	fname	city

Attribute vectors

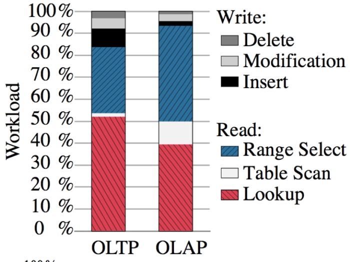
recID	fname	city

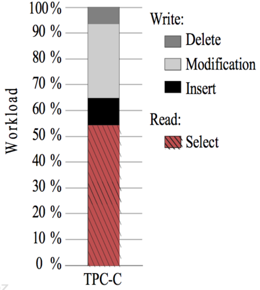
Validity vector

recID	valid

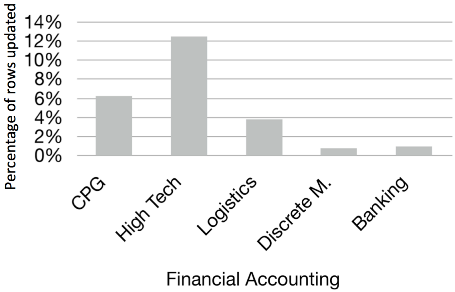
Column-oriented DB

OLTP vs OLAP





Few updates in OLTP



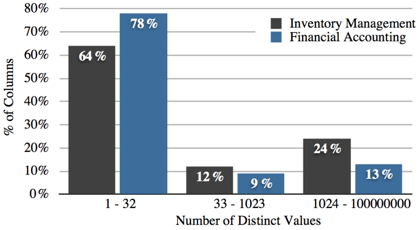
[Plattner, in-memory databases course]

Column-oriented DB

OLTP vs OLAP

55% unused columns per company in average

40% unused columns across all companies



[Plattner, in-memory databases course]

Wide tables with unused columns

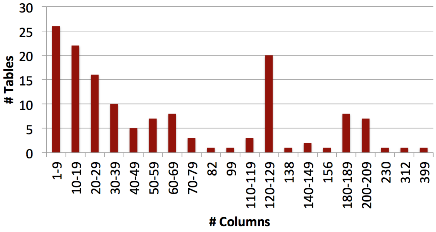


Table of content

2018-2019

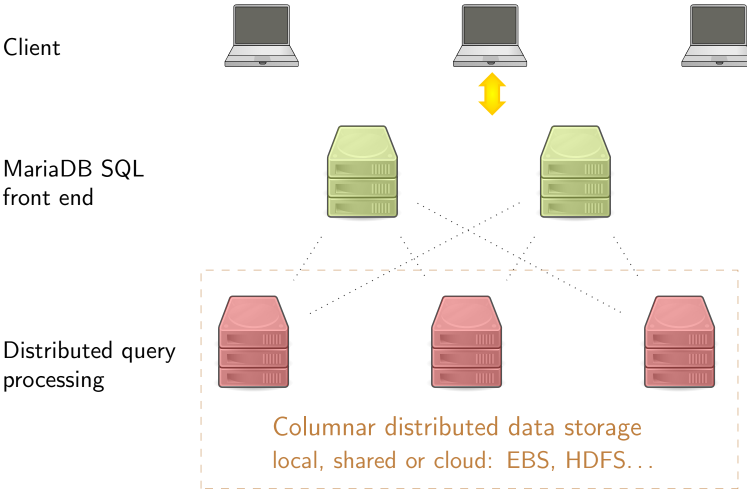
Column databases

- In-memory databases: commercial systems: MariaDB
- In-memory databases: commercial systems: Oracle
- In-memory databases: commercial systems: IBM DB2

MariaDB ColumnStore

A massively parallel column-storage engine ported from InfiniDB.

Recommended over MariaDB row storage when queries process millions-billions rows from billions-trillions rows tables.



MariaDB ColumnStore: storage architecture

Partition	Segment 1 · Segment 2 · Segment 3 · Segment 4
Segment file	Extent 1 · Extent 2
Extent 8M values	Block 1 · Block 2 · … · Block n
Block 8kB of values	Values

Values are fixed-length datatypes, 1-8bytes.
For larger values: pointer to dictionary entry.

MariaDB

Data is compressed using `snappy` library
≥250MB/s compression, 500MB/s decompression on single Core i7, 64bit.

Extent Map catalogs record the min and max value in each extent.
↪ query engine prunes irrelevant extents.

Block-based MVCC for consistency.
A version buffer records in an in-memory hash table the blocks being modified by a transaction.

Dedicated `cpimport` bulk loader.

```
create table t (
id int,
Name varchar(20),
) engine=columnstore;
```

Table of content

2018-2019

Column databases

- In-memory databases: commercial systems: MariaDB
- In-memory databases: commercial systems: Oracle
  - In-memory features
  - IM column store
- In-memory databases: commercial systems: IBM DB2

Oracle 12.2: In-memory features

In-memory aggregation: a query transformation considered by query optimizer, like star transformation, materialized views, or expansion...

For each dimension:

1. compute a dense grouping key for rows satisfying filters
2. compute the vector of grouping keys on the fact table
3. build temporary table mapping grouping keys to attribute values

Then

1. Scan&aggregate the fact table using key vectors: `VECTOR GROUP BY`
2. join back dimension attributes using temporary tables.

In-memory column store: copy of tables, in column format (detailed later)

In-memory aggregation: example

```
SELECT category, country, state, SUM(amount)
FROM   sales s, products p, geography g
WHERE  s.g_id = g.geo_id
AND    s.p_id = p.prod_id
AND    g.state IN ('WA','CA')
AND    p.manuf = 'ACME'
GROUP BY category, country, state
```

Geography

geo_id	city	state	country
2	Seattle	WA	USA
3	Spokane	WA	USA
7	SF	CA	USA
8	LA	CA	USA
...	...	...	France

Products

prod_id	category	manuf
4	sport	Acme
3	sport	Acme
1	food	Acme
8	electric	Acme
...	...	ATOS

temporary tables

dense_gr_key_g	state	country
1	WA	USA
2	CA	USA

dense_gr_key_p	category
1	sport
2	food
3	electric

Sales

p_id	g_id	amount
8	1	100
9	1	150
8	2	100
4	3	110
2	30	130
6	20	400
3	1	100
1	7	120
3	8	130
4	3	200

key vectors

dense_gr_key_p	dense_gr_key_g
3	
3	1
1	1
1	
2	2
1	2
1	1

 https://docs.oracle.com/database/121/TGSQL/tgsql_transform.htm#TGSQL95264

Oracle: IM column store

In-memory column stores. SGA pool that records *copy* of tables, in columnar format. Column store is recorded only in (volatile) memory.

Candidate for column store considered if declared in `CREATE` or `ALTER` statement. Once populated, kept consistent with the copy in row format.

```
ALTER TABLE t INMEMORY -- makes t candidate for populating the IM column store
  MEMCOMPRESS FOR CAPACITY HIGH
  PRIORITY LOW;
```

MEMCOMPRESS FOR

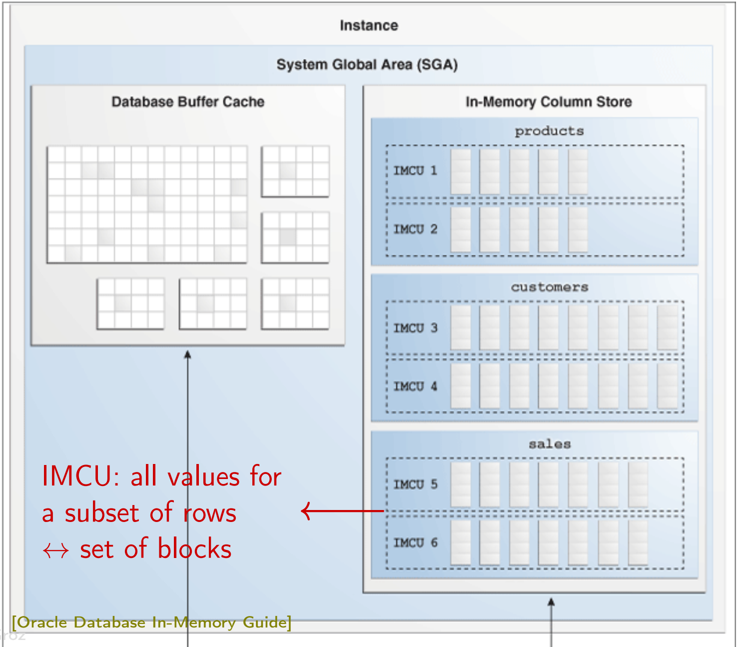
- QUERY LOW : best for queries (default)
- QUERY HIGH : higher compression
- ...
- CAPACITY HIGH : highest compression

PRIORITY

- NONE : populates only when object is scanned (default)
- LOW : populates after higher priority objects
- ...
- CRITICAL : highest priority

ORACLE

Oracle: IM column store architecture



IMCU: all values for
a subset of rows
$\leftrightarrow$ set of blocks

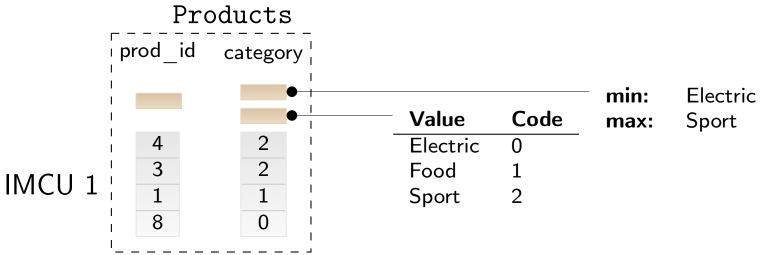
[Oracle Database In-Memory Guide]

Oracle: IM column store architecture

For each IMCU= set of CU.

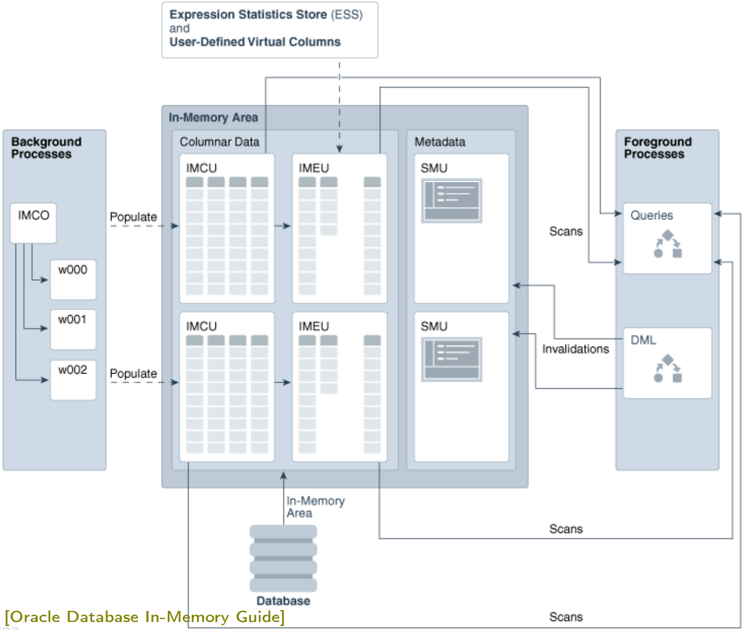
IMCU= In-Memory Compression Unit, CU= Column compression Unit

- header: metadata including
  - min/max value of each local column *(useful for pruning)*
  - (sometimes) local dictionary for local column data, implemented as a sorted list of distinct values with their dictionary code.
- body: the local column data, ordered by ROWID.



$\hookrightarrow$ **min/max** allows IMCU pruning. *Ex:* `SELECT...WHERE prod_id > 9`

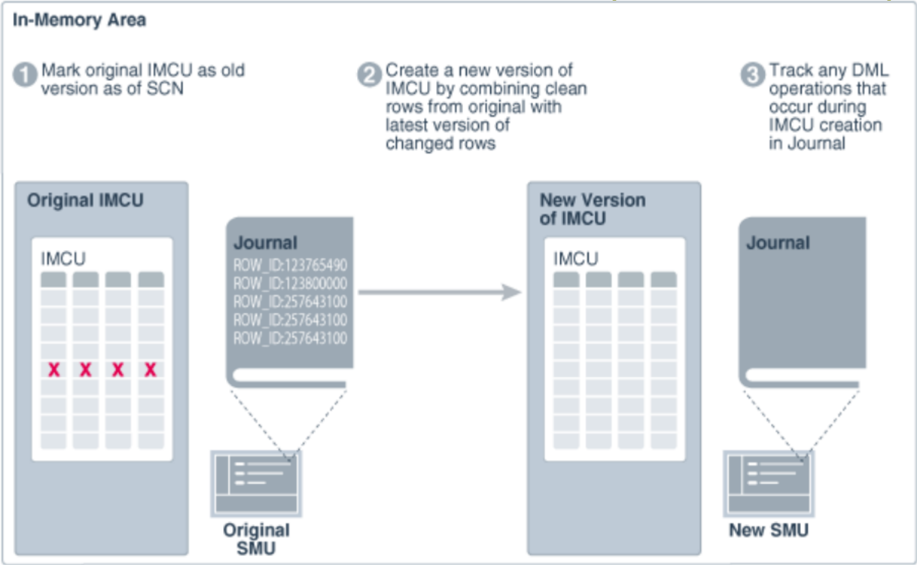
Oracle: IM column store architecture



[Oracle Database In-Memory Guide]

Oracle: IM column store updates

[Oracle Database In-Memory Guide]



- Threshold-based repopulation (number of changes in journal exceeds threshold)
- Trickle repopulation (periodic, updates all IMCU having some stale data)

Oracle: IM column store: integration on hardware

SQL-on-silicon: SPARC M7 chips:
SIMD on traditional CPU devised for graphics.
So devised chip with 8 Database accelerators with 4 pipelines = 32 engines supporting specialized instructions to process columnar data:

- Extract: uncompress data (byte/bit-packed, Huffman, RLE)
- Scan: filter data w.r.t an interval
- Select: filter data according to bit vector (given data vector and bit vector, return vector of selected items)
- Join

Some claims: 160GB/s bandwidth, 6x speedup on Apache Spark queries.

(Exadata) In-memory format in Exadata smart flash cache. Using additional flash cache to extend main memory (faster than disk): used to record db blocks evicted from SGA buffer cache.

Oracle vs SAP

SAP Hana: prototype around 2008, commercial product end 2011. Industry leader on in-memory techniques (though anteriority sometimes discussed)

2/3 of SAP Business suite customers rely on Oracle database.
By 2015, SAP tried to make ERP customers switch to SAP Hana: integrated stack S/4HANA.

SAP pushing toward cloud-based apps.

2016: Oracle 12 integrates In-memory storage. Special care to support SAP BW.

In 2017, SAP signs with Oracle to support Oracle on their ERP till 2025.

[http://www.silicon.fr/5-questions-comprendre-guerre-oracle-sap-in-memory-95002.html]

[http://www.scmfocus.com/saphana/2017/07/09/saps-change-policy-hana-oracle/]

As long as we are comparing

Prix des options Oracle (21 juin 2016) « Processor Licence » :

Active Data Guard	11 500 $
Database In-Memory	23 000 $
Diagnostics Pack	7 500 $
Tuning Pack	5 000 $
Partitionning	11 500 $
Advanced compression	11 500 $
OLAP	23 000 $
Advanced Analytics	23 000 $
Spatial	17 500 $
Multitenant	17 500 $
TOTAL :	139 500 $

[Source: https://blog.developpez.com/sqlpro/p13001/ms-sql-server/oracle-vs-sql-server-les-options-payantes-qui-font-la-difference]

Oracle édition Enterprise : 47 500$
Pricing of DBMS generally a bit opaque (multiple discounts, what should be counted: license cost only or ROI, etc).

Multiple discussions online about Oracle vs Microsoft SQL Server, MariaDB etc.

Table of content

2018-2019

Column databases

- In-memory databases: commercial systems: MariaDB
- In-memory databases: commercial systems: Oracle
- In-memory databases: commercial systems: IBM DB2
  - In-memory features: DB2 12 for z/OS
  - In-memory Column store: DB2 BLU

IBM DB2 12 for z/OS

DB2 12 (2016) emphasizes in-memory computations. Users must provision large RAM to benefit from enhancements.

In-memory contiguous buffer pool. Unlike versions<12, makes sure page stealing only occurs in some 10% overflow area, which results in savings on cache page management.

In-memory index optimization. Reserves an in-memory area to speedup lookups in unique indexes.

Speeding up INSERTS. New insertion algorithm for inserts on non-clustered tables (table with MEMBER CLUSTER attribute). Requires more memory be assigned for table space partitions.

DB12 also kind of increases pool dedicated to sorting operations so they may fit in-memory...

IBM DB2 11.1 LUW with BLU acceleration

DB2 11.1 (2016), but BLU accelerations introduced in DB2 10.5 (2013). BLU emphasizes processing compressed columnar data, parallelization, SIMD, memory management...

For more, see: *DB2 with BLU acceleration*, VLDB'2013

http://db.disi.unitn.eu/pages/VLDBProgram/pdf/industry/p773-barber.pdf

```sql
CREATE TABLE Employee (
  ID SMALLINT NOT NULL,
  NAME VARCHAR(9),
  DEPT SMALLINT,
  SALARY DECIMAL(7,2)
)
 ORGANIZE BY COLUMN;
```

References

Column/In-memory databases (general):

- *The Design and Implementation of Modern Column-Oriented Database Systems*, Abadi et al. Foundations and trends in database, 2012

Column/In-memory databases (SAP):

- *In-Memory Data Management*, H.Plattner, livre disponible B.U. Orsay, et MOOC: https://open.hpi.de/courses/imdb2015
- *Parallel Replication across Formats in SAP HANA for Scaling Out Mixed OLTP/OLAP Workloads, VLDB'2017* *http://www.vldb.org/pvldb/vol10/p1598-han.pdf*
- *SAP HANA Adoption of Non-Volatile Memory, VLDB'2017* *http://www.vldb.org/pvldb/vol10/p1754-andrei.pdf*

Column/In-memory databases (Oracle):

- *https://docs.oracle.com/database/122/INMEM/toc.htm*
- *Query Optimization in Oracle 12c Database In-Memory, VLDB'15* *http://www.vldb.org/pvldb/vol8/p1770-das.pdf*
- *Distributed Architecture of Oracle Database In-memory, VLDB'15* *http://www.vldb.org/pvldb/vol8/p1630-mukherjee.pdf*

Bibliographie

Column/In-memory databases (MariaDB):

- *https://mariadb.com/kb/en/library/mariadb-columnstore/*
- *https://mariadb.com/kb/en/library/columnstore-storage-architecture/*

Column/In-memory databases (IBM):

- *https://en.wikipedia.org/wiki/IBM_BLU_Acceleration*
- *DB2 with BLU acceleration*, VLDB'2013 http://db.disi.unitn.eu/pages/VLDBProgram/pdf/industry/p773-barber.pdf
- *http://www.redbooks.ibm.com/redbooks/pdfs/sg248383.pdf and other redbooks.*

Trends in DWH:

- *Vendors white papers (Microsoft, Oracle, etc), Gartner, tdwi, etc. Ex:* http://download.microsoft.com/download/C/2/D/C2D2D5FA-768A-49AD-8957-1A434C6C8126/The_Microsoft_Modern_Data_Warehouse_White_Paper.pdf

Data processing on modern Hardware:

- http://www.odbms.org/wp-content/uploads/2014/03/Data-Processing-on-FPGAs.pdf
- http://edbticdt2016.labri.fr/downloads/gustavo_alonso_slides.pdf

Columnar storage. . .

4 APACHE Projects on serialization in column format.

Arrow : *in-memory*

- objective: *standard* for in-memory column storage. More about manipulating (volatile, in RAM) data than about defining a file format
- caracteristics: focus on vectorization, zero-copy, random reads

Parquet Parquet : *immutable, on disk*

- caracteristics: focus on compression of columnar data

Kudu : *updatable, on disc (+ buffer)*

- caracteristics: more like a column store than a format. Format similar to Parquet, but optimized for updates.

ORC Apache orc

- objective: optimize Hive and MR
- caracteristics: compression, indexes, predicate pushdown

References

- A post by McKinney answering a post on Arrow vs Parquet: `http://wesmckinney.com/blog/arrow-columnar-abadi/`
- and some additional answer to the author: `http://dbmsmusings.blogspot.fr/2018/03/an-analysis-of-strengths-and-weaknesses.html`