In-memory databases

Contents

- In-memory database technology
- Commercial In-memory Databases
- SQL Server Memory Optimized Tables
- Exercise

Contents

- **In-memory database technology**
- Commercial In-memory Databases
- SQL Server Memory Optimized Tables
- Exercise

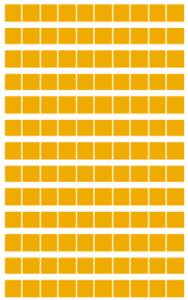
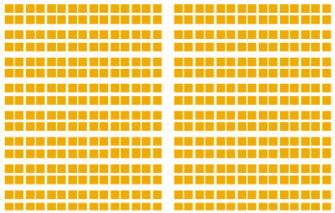
Classical Database Architecture

- Classical Database management systems (DBMS) are designed late 1970's for:
  - Delivering performance on hardware with
    - limited main memory

    and
    - slow disk I/O

    as the main bottleneck
- The focus was on optimizing disk access
  - for example by minimizing the number of disk pages to be read into main memory when processing a query (see chapter about indexing)
- Sequential processing paradigm
  - Data tables are fetched from the database
  - Row by row processing
  - Write-back to the database

Hardware revolution

20 Years Ago		Now	Near Future
			
Memory 1GB	X 6,000	**Memory** 6 TB	**Memory** 48 TB
CPU 4 x 50 Mhz	X 1,800	**CPU** 120 x 3 GHz	**Cores** 480 (8 x 4 x 15)
Transistors (CPU) ~ 1 million		**Transistors (CPU)** 2.6 Billion	

Note: Figures are for single servers

In-memory database technology

The basic idea is that memory is much faster than disk

- a modern CPU has a **memory bandwidth** of **20 GB/sec** and higher
  - a **single disk** is around:
    - **550 Mbyte/sec for SSDs**
    - **180 Mbyte/sec for hard disk drives**
  - => difference by **factor 36 and 110**.
- If a program is using the same memory frequently, it is cached inside the CPU's L1 or L2 cache, speeding up the memory bandwidth by another factor of 10.
- At the same time
  - the disk speed of 180MB/sec is **for sequential read access** only
  - **random access would be times worse** for a disk system

In-memory database technology

- In-memory means that all the data is stored in the RAM memory
- No time waste in reading from disk
- Quick access from CPU to data

In-memory database technology

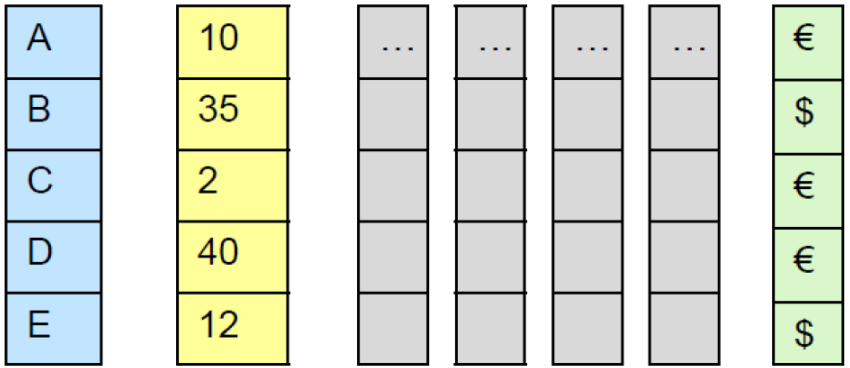
- Pro’s:
  - Fast updating
  - Fast inserting
  - Fast reading
- Cons:
  - When the power is gone => all data are gone
  - Cost:
    - 1 TB , 2 TB server is affordable
    - 64 Tb server is expensive
    - 1000 TB server does not exist

Storing tabular data: two ways

Table of Information

A	10	… many columns	€
B	35		$
C	2		€
D	40		€
E	12		$

… by column

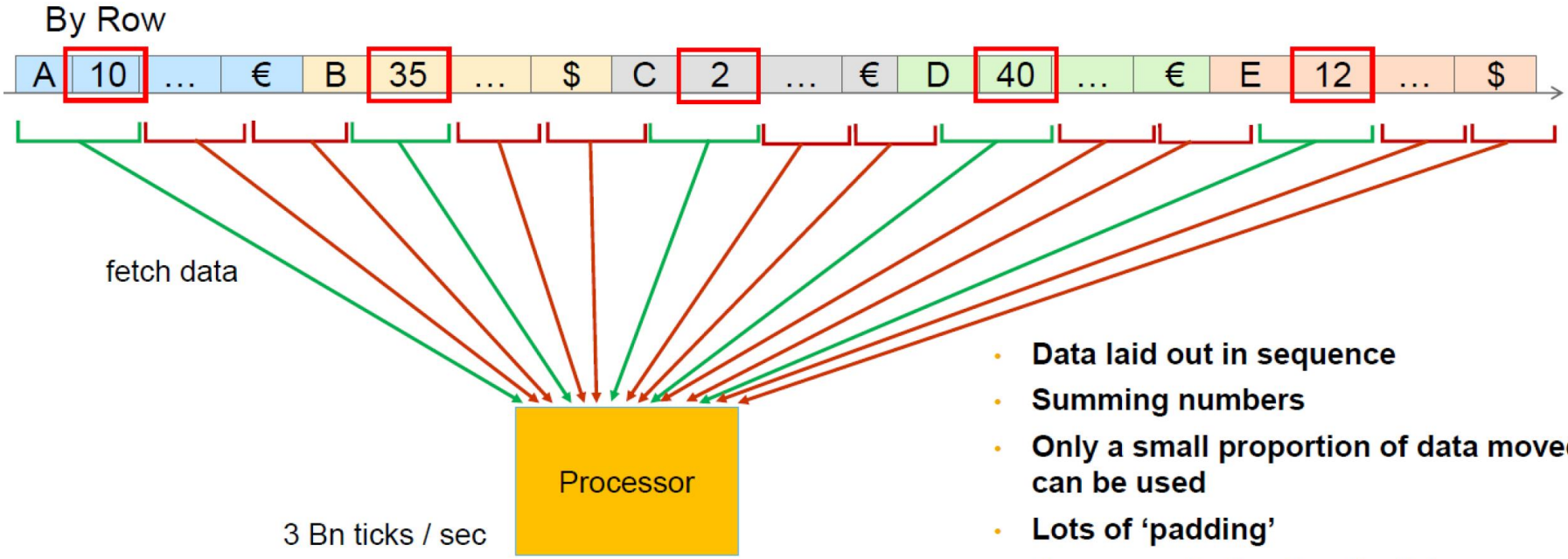


… or by row

A	10	…	€	B	35	…	$	C	2	…	€	D	40	…	€	E	12	…	$

Store by row

For rows, data is moved to the processor in ‘chunks’ but only a tiny proportion of those ‘chunks’ is useful



- **Data laid out in sequence**
- **Summing numbers**
- **Only a small proportion of data moved can be used**
- **Lots of ‘padding’**
- **Processor ‘spins its wheels’**

Columnar storage

Row-based Storage

Order	Country	Product	Sales
456	France	Corn	$1000
457	Italy	Wheat	$2000
458	Italy	Corn	$100
459	Italy	Peas	$110
460	Italy	Ferrari Lug Nut	$200
461	Spain	Rice	$700
462	Germany	Beer	$300

456
457
458
459
460
461
462

France
Italy
Italy
Italy
Italy
Spain
Germany

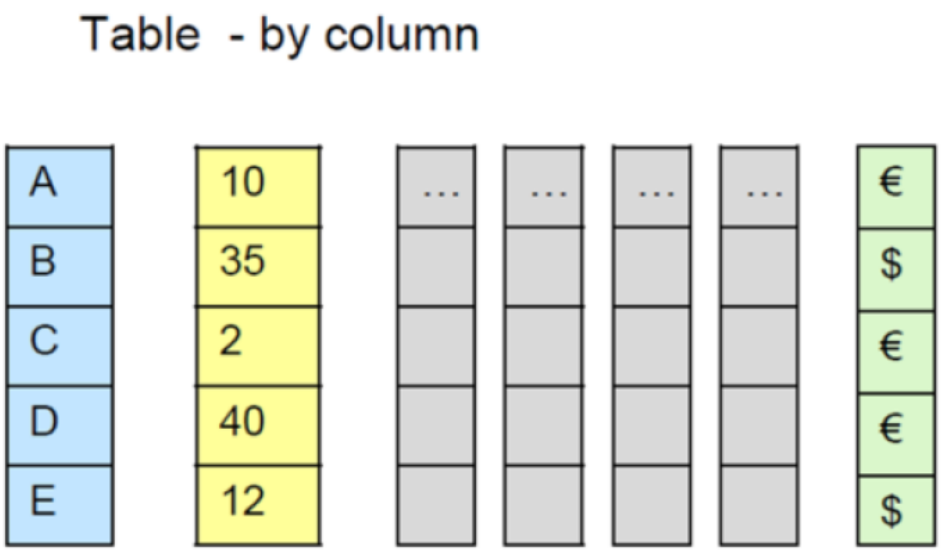
Corn
Wheat
Corn
Peas
Ferrari Lug Nut
Rice
Beer

$1000
$2000
$100
$110
$200
$700
$300

Column-based Storage

- To read a row store, we would have to go through each record one by one
- Columnar Storage will instead store each column separately with its own, separate index
- Allows to easily exclude entire columns not included in the query:
- Ex. queries for Country and sales will ignore Order and Product, thereby increasing the speed and efficiency of the query.

Columnar storage: benefits



- Highly dense data structures
- Compresses nicely
- Multiple values for aggregate calculations available at once
- Easy to add new columns => no need to rewrite table
- Only fetch columns that are asked in query
- Makes optimal use of modern CPU architectures

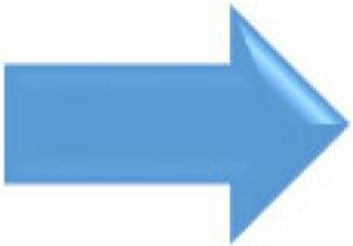
Columnar storage: compression

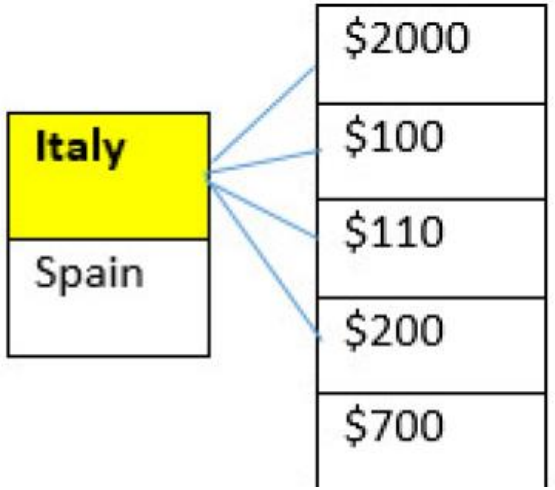
- Basic idea: a single CPU (and RAM) is much faster than the disk
  - Compression of data in order to reduce the amount of data is beneficial as long as the overhead of that is not too huge
- Every DB technology uses compression
  - But compression and decompression ask a toll (time)
  - Most obvious overhead:
    - Update data in a database block
      - Uncompress DB block
      - Merge change data in block
      - Compress DB block again

Columnar storage: compression

- Each column is stored separately => greater rates of compression

Italy	$2000
Italy	$100
Italy	$110
Italy	$200
Spain	$700





- In the example records indicating ‘Italy’ are compressed 4 to 1

Columnar storage with in-memory

- Conclusion: columnar storage
  - reduces required size of data
  - has faster access
  - is only useful for in-memory storage

Contents

- In-memory database technology
- **Commercial In-memory Databases**
- SQL Server Memory Optimized Tables
- Exercise

Commercial in-memory databases

- SAP Hana: completely redesigned database concept
- Oracle: IM (in-memory) column store:
  - “maintains copies of tables, partitions, and individual columns in a special compressed columnar format that is optimized for rapid scans.”
- Microsoft: “memory-optimized tables”

Contents

- In-memory database technology
- Commercial In-memory Databases
- **SQL Server Memory Optimized Tables**
- Exercise

SQL Server Memory Optimized Tables

- Up to 30x performance gain according to Microsoft
- It is not (only) fast because it is in-memory; it is fast because it is optimized around the data being in-memory (see previous slides)
- Data lives in-memory does not mean you lose it when there is a failure.
  - By default, all transactions are fully durable.
  - As part of transaction commit, all changes are written to the transaction log on disk.
  - If there is a failure at any time after the transaction commits, your data is there when the database comes back online.

Memory optimized tables: usage scenarios (1/2)

- High-throughput and low-latency transaction processing
  - Core scenario.
  - E.g.: stock trading, sports betting, mobile gaming.
- Data ingestion
  - Ingesting large volumes of data from different sources at same time.
  - E.g.: IoT sensor readings and events.

Memory optimized tables: usage scenarios (2/2)

- Caching and session state
  - E.g. ASP.NET session state
- Tempdb object replacement (MS-SQL Server)
  - Replace temporary tables (#table, ##table) and table variables (@table)
- ETL (Extract Transform Load)
  - ETL workflows often include load of data into a staging table, transformations of the data, and load into the final tables.
  - Use non-durable memory-optimized tables (option: `DURABILITY=SCHEMA_ONLY)` for the data staging. They completely remove all IO, and make data access more efficient.

Memory optimized tables: syntax

See script “MemoryOptimizedTables.sql”

Restrictions:

- Clustered indexes, which are the default for primary keys, are not supported with memory-optimized tables. Specify a NONCLUSTERED index instead.
- Computed columns are not supported with memory-optimized tables.
- The feature ROWGUIDCOL is not supported with memory-optimized tables.
- **Tables that are migrated to memory can’t have foreign key relationships with other tables, so they first have to be removed.** Foreign key relationships also can’t be restored after migration of a single table, so in practice often the complete database has to be migrated to memory if you want to restore foreign keys if all tables are connected through foreign keys.

Contents

- In-memory database technology
- Commercial In-memory Databases
- SQL Server Memory Optimized Tables
- **Exercise**

Migrating tables to In-Memory: exercise

Database xtreme

- Migrate table Supplier to memory in a durable way.
  - Describe the different steps
  - Check the result

References

“High Performance SQL Server”, 2016, Benjamin Nevarez, Apress

“Expert Performance Indexing in SQL Server 2019”, Jason Strate, 2019, Apress

https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/overview-and-usage-scenario