Data Warehousing

Saptarshi Pyne
Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology Jodhpur, Rajasthan, India 342030

CSL4030 Data Engineering Lectures 27, 28, 29
October 13th, 20th, 26th, 2023

What we discussed in the last class

- Hashing and Indexing
  - Hash maps or hash tables
  - B-trees
  - Sorted strings tables (SSTables)
  - Log-structured merge trees (LSM-trees)
  - Bloom filters

Transaction processing vs. batch processing

Transaction processing = Executing jobs that are needed to be performed with low latency i.e. in near real-time.

In the context of the term ‘transaction processing’, a ‘transaction’ could either be a database transaction with the ACID properties or a single set of read/write operations with the BASE properties.

Batch processing = Executing jobs that are scheduled to be performed periodically such as every night at 12 am.

Online transaction processing (OLTP) vs. online analytic processing (OLAP)

OLTP and OLAP are two broad categories of **storage engines**.

Storage engine = In a DBMS software, the component that manages how data is stored in main memory and on disk [1].

OLTP = Storage engines optimized for storing everyday operational data.

OLAP = Storage engines optimized for handling business analytics queries.

[1] https://www.mongodb.com/docs/manual/core/storage-engines/

ETL (Extract-transform-load)

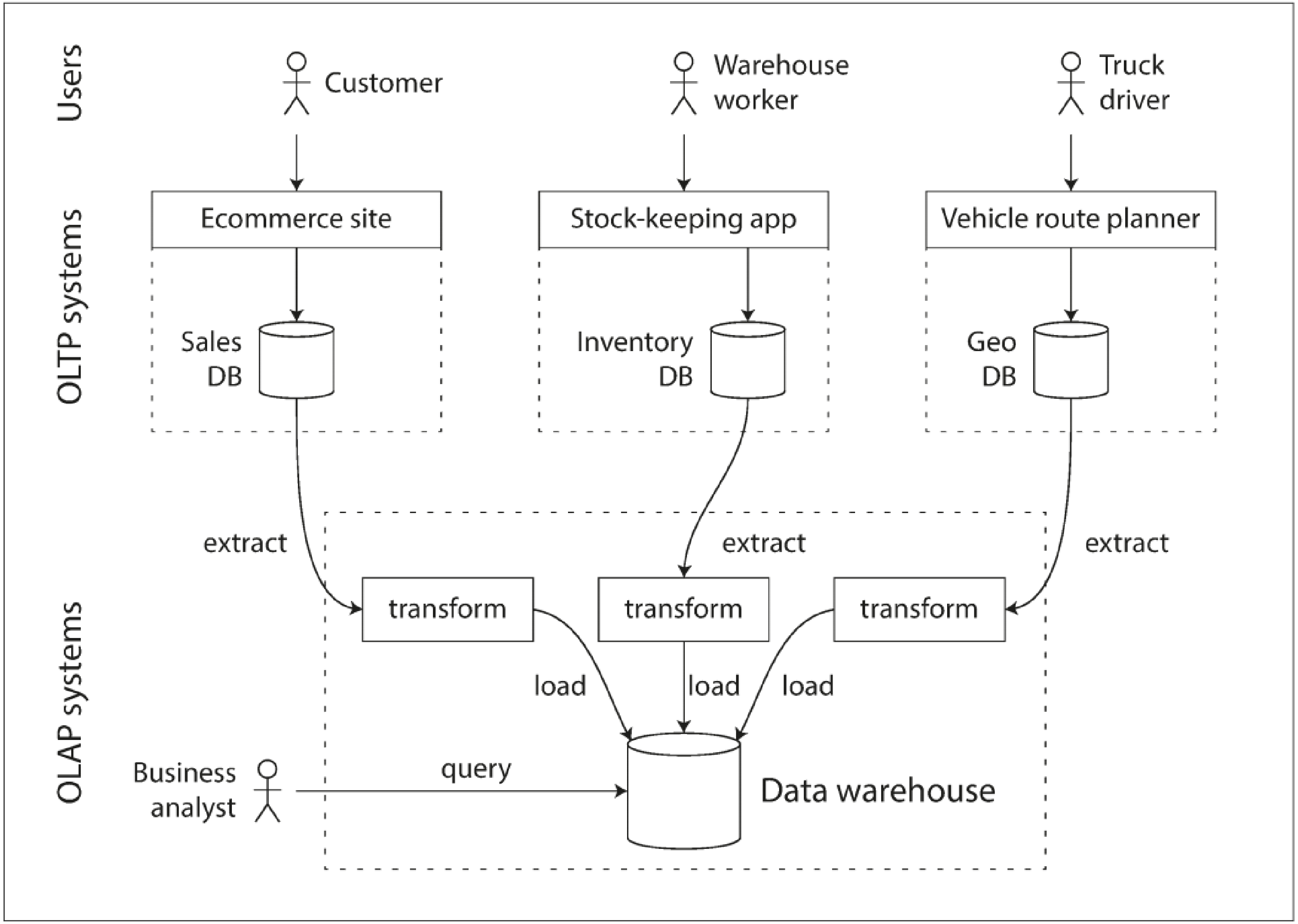
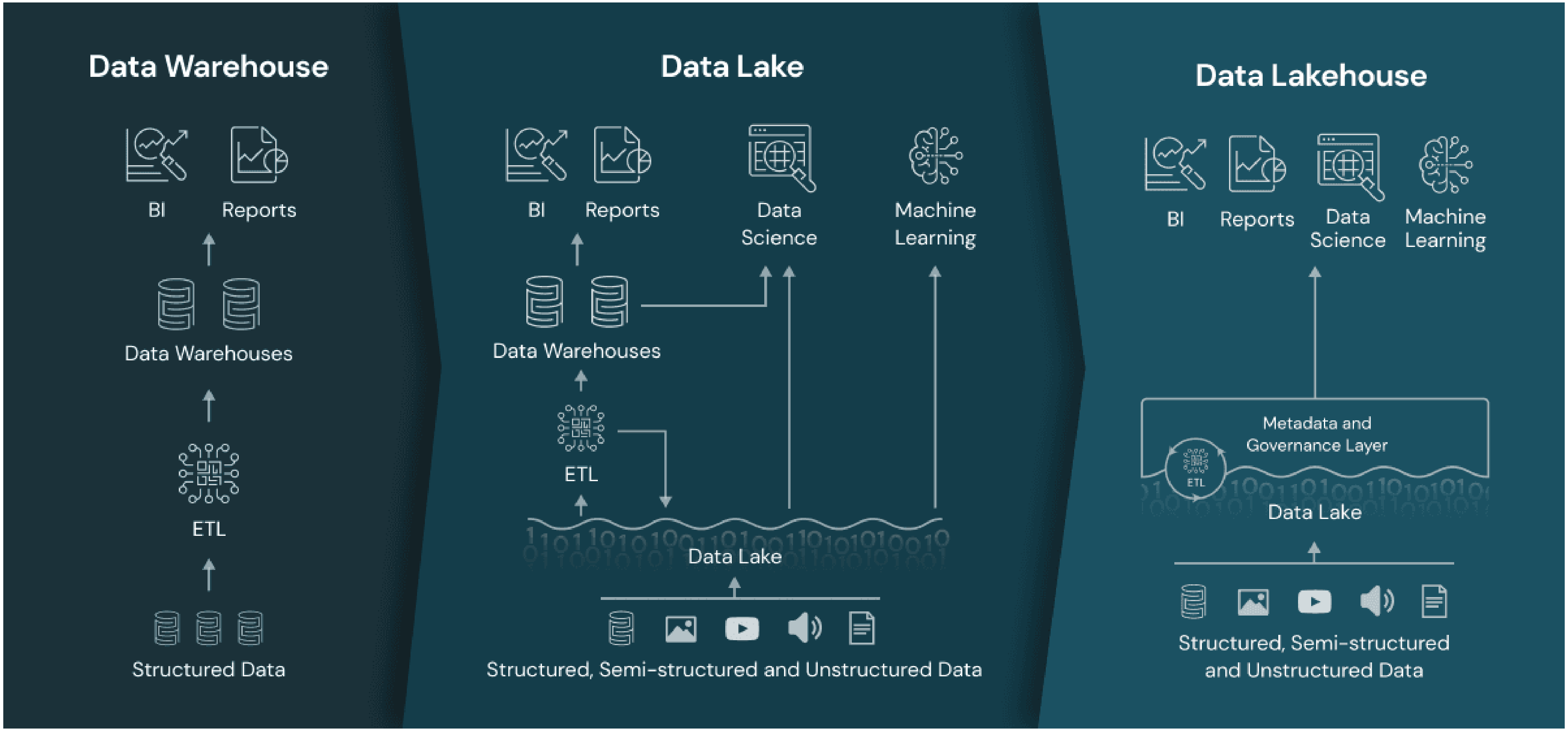


Figure 3-8. Simplified outline of ETL into a data warehouse.

Data warehouse vs. data lake vs. data lakehouse



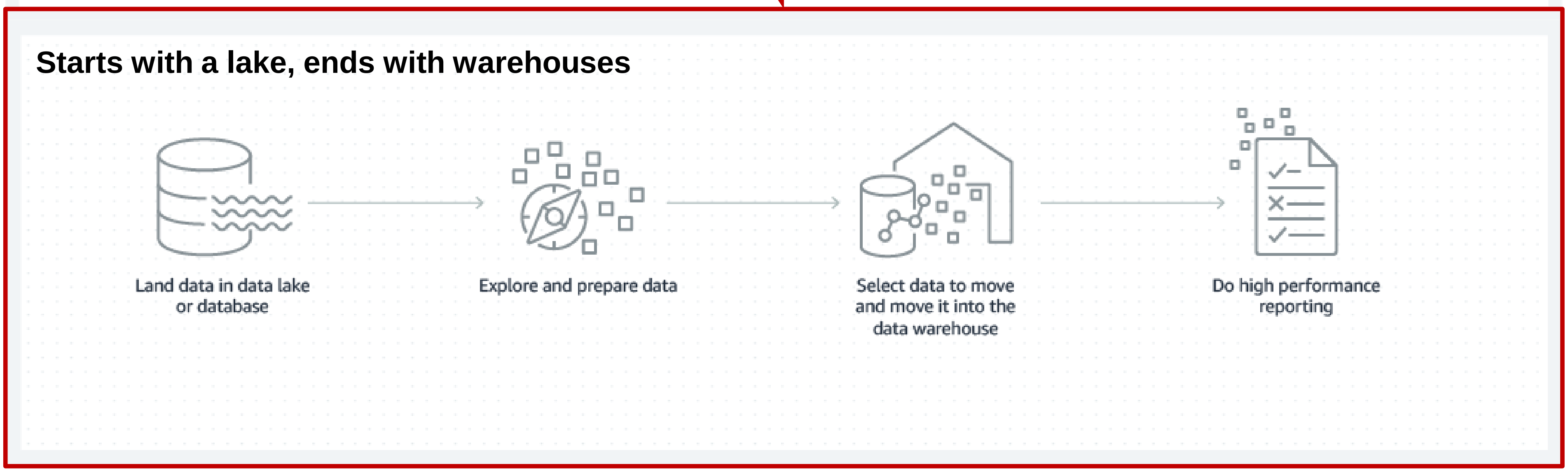
databricks

A ‘Data Lakehouse Architecture and **AI Company**’.
It is a 4 billion dollar data engineering and AI startup founded by the Apache Spark founders in 2016 at UC Berkeley.

https://www.databricks.com/glossary/data-lakehouse

Data lakehouse = lake + warehouse

Amazon Redshift is a **lakehouse** service offered by AWS.



https://aws.amazon.com/data-warehouse/

Types of data warehouse schemas

- The star schema
  - Centre = A fact table with one event per row
  - Periphery = Dimensional tables each representing different dimensions of the fact table
- The snowflake schema

An example of a star schema

dim_product table

product_sk	sku	description	brand	category
30	OK4012	Bananas	Freshmax	Fresh fruit
31	KA9511	Fish food	Aquatech	Pet supplies
32	AB1234	Croissant	Dealicious	Bakery

dim_store table

store_sk	state	city
1	WA	Seattle
2	CA	San Francisco
3	CA	Palo Alto

fact_sales table

date_key	product_sk	store_sk	promotion_sk	customer_sk	quantity	net_price	discount_price
140102	31	3	NULL	NULL	1	2.49	2.49
140102	69	5	19	NULL	3	14.99	9.99
140102	74	3	23	191	1	4.49	3.89
140102	33	8	NULL	235	4	0.99	0.99

The fact table

dim_date table

date_key	year	month	day	weekday	is_holiday
140101	2014	jan	1	wed	yes
140102	2014	jan	2	thu	no
140103	2014	jan	3	fri	no

dim_customer table

customer_sk	name	date_of_birth
190	Alice	1979-03-29
191	Bob	1961-09-02
192	Cecil	1991-12-13

dim_promotion table

promotion_sk	name	ad_type	coupon_type
18	New Year sale	Poster	NULL
19	Aquarium deal	Direct mail	Leaflet
20	Coffee & cake bundle	In-store sign	NULL

Figure 3-9. Example of a star schema for use in a data warehouse.

The fact table can have more or less than five **rays of star** i.e. any number of **dimension tables**.

Types of data warehouse schemas (contd.)

- The star schema

- The snowflake schema
  - More normalized than the star schema
  - Demerit: Requires more joins for OLAP. More suitable for OLTP.

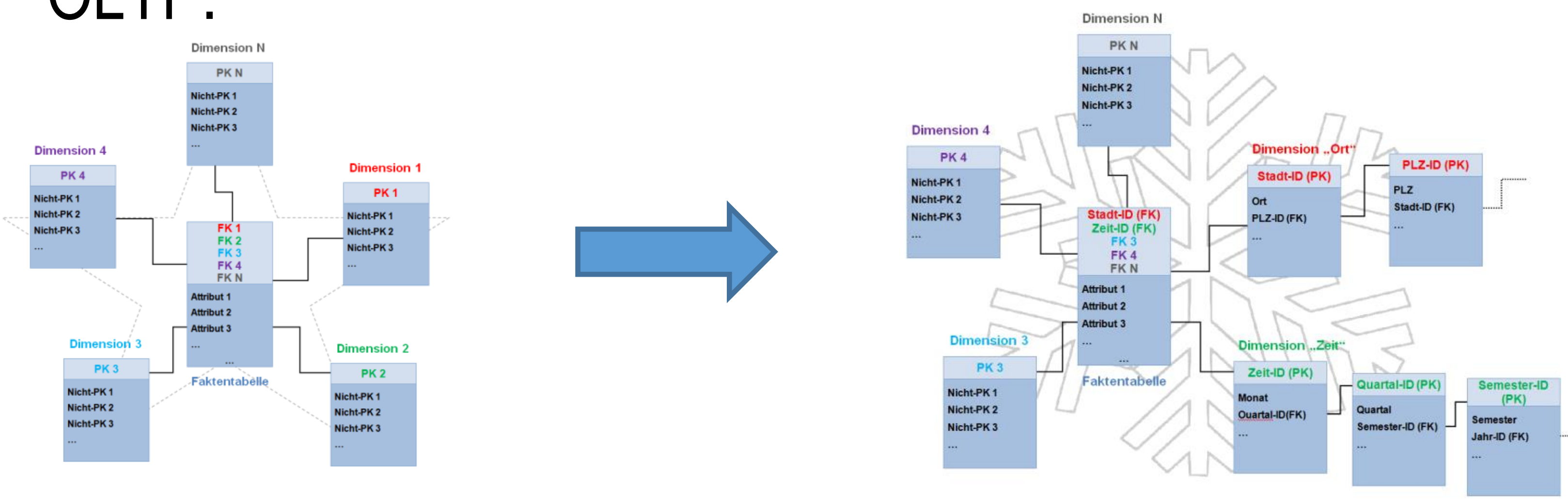


Figure courtesy:
https://en.wikipedia.org/wiki/Star_schema#/media/File:Star_Schema.png
https://en.wikipedia.org/wiki/Snowflake_schema#/media/File:Snowflake_schema.png

How do OLTP storage engines store data on disk?

OLTP usually handles a continuous stream of read/write operations **each affecting a small number of records**.

E.g.,

```
UPDATE fact_sales
SET quantity = 1
WHERE product_sk = 69 AND store_sk = 3;
```

Hence, the efficiency lies in finding the target records as fast as possible. For this reason, an efficient indexing scheme is paramount.

How do OLTP storage engines store data in disk? (contd.)

- The (classical) update-in-place paradigm
  - Update a single log file
  - Indexing data structures: B-tree, B^+ tree
  - Software: Most OLTP storage engines
- The (modern) log-structured paradigm
  - Never update a block, append to the existing block, write a new block
  - Indexing data structures: SSTables, LSM-trees,
  - Software: Bitcask, LevelDB, Cassandra, HBase, Lucene

Challenges of OLTP storage engines

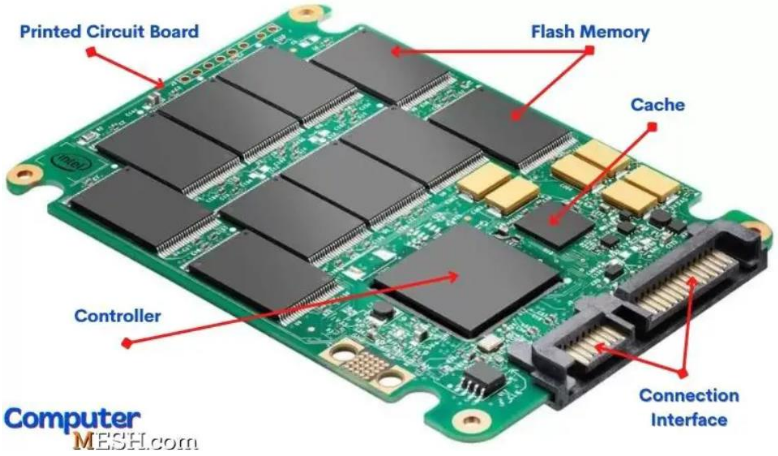
Disk seek time is a limiting factor due to continuous disk read/write operations.

- Resolution 1:
  - Replace random-access writes with sequential-access writes i.e. log-structured writes. The read/write head of the hard disk can perform random access by moving its head from one location to a distant location. Such head movements are time consuming. One the other hand, sequential access allows the read/write head to continuously move to the next word. Such sequential movement is much less time consuming.

Challenges of OLTP storage engines (contd.)

Disk seek time is a limiting factor due to continuous disk read/write operations.

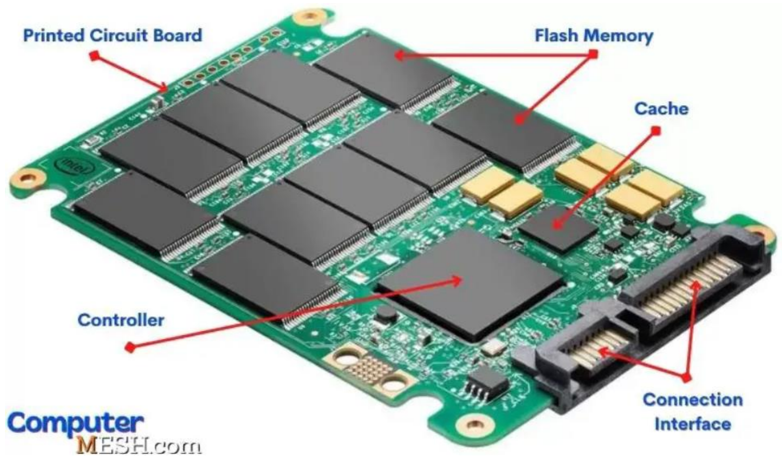
- Resolution 2:
  - Utilize flash-based solid state drive (SSD) disks.



Challenges of OLTP storage engines (contd.)

SSDs store data as electrical voltage in **flash memory cells** that have random access time in microseconds.

Additionally, modern SSDs utilize **parallelization** by dividing the memory cells into partitions and then having one controller for each partition.



Section 'Solid-State Drive', Chapter 6 'Storage', J. Reis, M. Housley (2022), Fundamentals of Data Engineering, O'Reilly Media, Inc.,ISBN: 9781098108304

Challenges of OLTP storage engines (contd.)

Disk Type	Max. Read Speed	Max. Write Speed
SATA HDD	80 Mbps	160 Mbps
SATA SSD	200 Mbps	550 Mbps
NVMe (Non-volatile memory express) PCIe SSD	7.3 Gbps	6.35 Gbps

How do OLAP storage engines store data in disk?

OLTP usually generates business intelligence reports by performing queries **each reading a small number of columns from a large chunk of related records**.

E.g., (Analyze whether people are more inclined to buy fresh fruit or candy, depending on the day of the week)

SELECT **dim_date.weekday**, **dim_product.category**, SUM(**fact_sales.quantity**) AS quantity_sold

FROM fact_sales

JOIN dim_date ON fact_sales.date_key = dim_date.date_key

JOIN dim_product ON fact_sales.product_sk = dim_product.product_sk

WHERE

dim_date.year = 2014 AND dim_product.category IN ('Fresh fruit', 'Candy')
GROUP BY dim_date.weekday, dim_product.category;

Hence, the efficiency lies in compressing column data.

The column storage strategy

A row-oriented storage (a fact table usually have trillions of records and 100+ columns)

fact_sales table

date_key	product_sk	store_sk	promotion_sk	customer_sk	quantity	net_price	discount_price
140102	69	4	NULL	NULL	1	13.99	13.99
140102	69	5	19	NULL	3	14.99	9.99
140102	69	5	NULL	191	1	14.99	14.99
140102	74	3	23	202	5	0.99	0.89
140103	31	2	NULL	NULL	1	2.49	2.49
140103	31	3	NULL	NULL	3	14.99	9.99
140103	31	3	21	123	1	49.99	39.99
140103	31	8	NULL	233	1	0.99	0.99

A column-oriented storage (each column is stored in a separate file)

Columnar storage layout:

date_key file contents:	140102, 140102, 140102, 140102, 140103, 140103, 140103, 140103
product_sk file contents:	69, 69, 69, 74, 31, 31, 31, 31
store_sk file contents:	4, 5, 5, 3, 2, 3, 3, 8
promotion_sk file contents:	NULL, 19, NULL, 23, NULL, NULL, 21, NULL
customer_sk file contents:	NULL, NULL, 191, 202, NULL, NULL, 123, 233
quantity file contents:	1, 3, 1, 5, 1, 3, 1, 1
net_price file contents:	13.99, 14.99, 14.99, 0.99, 2.49, 14.99, 49.99, 0.99
discount_price file contents:	13.99, 9.99, 14.99, 0.89, 2.49, 9.99, 39.99, 0.99

Figure 3-10. Storing relational data by column, rather than by row.

Column compression using bitmaps (one bitmap for each distinct value)

Column values:

product_sk:	69	69	69	69	74	31	31	31	31	29	30	30	31	31	31	68	69	69

Bitmap for each possible value:

product_sk = 29:	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
product_sk = 30:	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
product_sk = 31:	0	0	0	0	0	1	1	1	1	0	0	0	1	1	1	0	0	0
product_sk = 68:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
product_sk = 69:	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
product_sk = 74:	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

The number of bits in each bitmap = The number of rows in the table.

Fast analytics using column bitmaps

Case study:
SELECT quantity FROM fact_sales
WHERE product_sk = 31 AND store_sk = 3;

product_sk values: 69, 69, 69, 74, 31, 31, 31, 31
product_sk = 31: 0 0 0 0 1 1 1 1

store_sk values: 4, 5, 5, 3, 2, 3, 3, 8
store_sk = 3: 0 0 0 1 0 1 1 0

(product_sk = 31 AND store_sk = 3): bitwise AND
0 0 0 0 0 1 1 0
Select only the 6th and 7th rows.

Sparse bitmaps

For a column, when the number of distinct values is small and the values are almost uniformly distributed, the vanilla bitmap encoding is sufficient.

However, when there are a large number of distinct values or the distribution of values is skewed, we will see a lot of zeroes in each bitmap. Such bitmaps are called **sparse bitmaps**.

Sparse bitmaps can be more efficiently stored as **run-length encoded (RLE) bitmaps**.

Run-length encoded (RLE) bitmaps

Column values:

product_sk:	69	69	69	69	74	31	31	31	31	29	30	30	31	31	31	68	69	69

Bitmap for each possible value:

product_sk = 29:	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
product_sk = 30:	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
product_sk = 31:	0	0	0	0	0	1	1	1	1	0	0	0	1	1	1	0	0	0
product_sk = 68:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
product_sk = 69:	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
product_sk = 74:	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Run-length encoding:

product_sk = 29:	9, 1	(9 zeros, 1 one, rest zeros)
product_sk = 30:	10, 2	(10 zeros, 2 ones, rest zeros)
product_sk = 31:	5, 4, 3, 3	(5 zeros, 4 ones, 3 zeros, 3 ones, rest zeros)
product_sk = 68:	15, 1	(15 zeros, 1 one, rest zeros)
product_sk = 69:	0, 4, 12, 2	(0 zeros, 4 ones, 12 zeros, 2 ones)
product_sk = 74:	4, 1	(4 zeros, 1 one, rest zeros)

Run-length encoded bitmaps

9N1Y8N

10N2Y6N

5N4Y3N3Y3N

N = No or zero

Y = Yes or one

However, we lose the ability to do bitwise operations.

Figure 3-11. Compressed, bitmap-indexed storage of a single column.

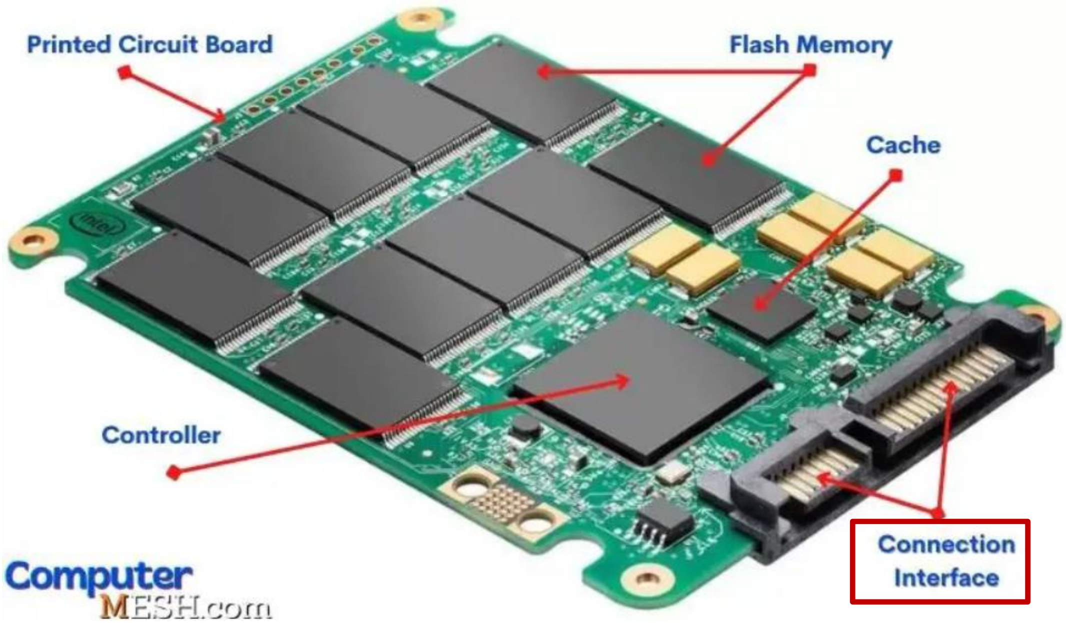
Challenges of OLAP storage engines

Once the to-read values are located, we also need a mechanism to transfer/copy them from the disk to the main memory as fast as possible.

Challenges of OLAP storage engines (contd.)

Disk bandwidth is a limiting factor for transferring large volumes of data from disk to main memory.

- Resolutions: Use SSDs with higher bandwidth connection interfaces, e.g., use NVMe PCIe SSDs instead of SATA SSDs.



Challenges of OLAP storage engines (contd.)

Disk Type	**Max. Data Transfer Rate**
SATA HDD	160 Mbps
SATA SSD	600 Mbps
NVMe (Non-volatile memory express) PCIe SSD	3.5 Gbps

https://computermesh.com/types-of-solid-state-drives-and-connector-type/

Another way to reduce read latency in OLAP

Ready-to-read data (like ready-to-eat food)

- Materialized aggregates
  - Materialized views:
    A materialized view is a database object that contains the results of a query.
    Example:
    CREATE MATERIALIZED VIEW mv1 AS
    SELECT * FROM fact_sales;

https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-MATERIALIZED-VIEW.html#GUID-EE262CA4-01E5-4618-B659-6165D993CA1B

Another way to reduce read latency in OLAP (contd.)

Ready-to-read data (like ready-to-eat food)

- Materialized aggregates
  - Materialized views
    - Special type: Data cubes or OLAP cubes or hypercubes

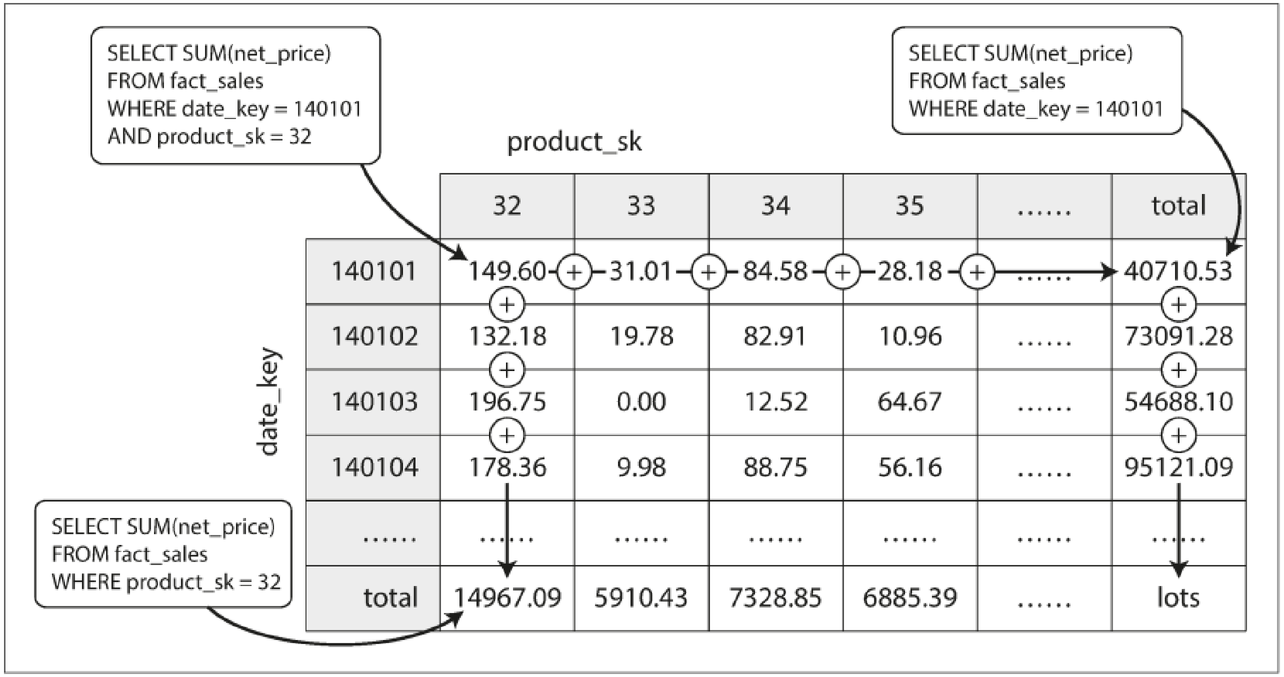


Figure 3-12. Two dimensions of a data cube, aggregating data by summing.

References

- M. KLEPPMANN (2017), Designing Data-Intensive Applications The Big Ideas Behind Reliable, Scalable, and Maintainable Systems, O’Reilly.
  - Pages 90-108, Chapter 3: Storage and Retrieval

Thank you