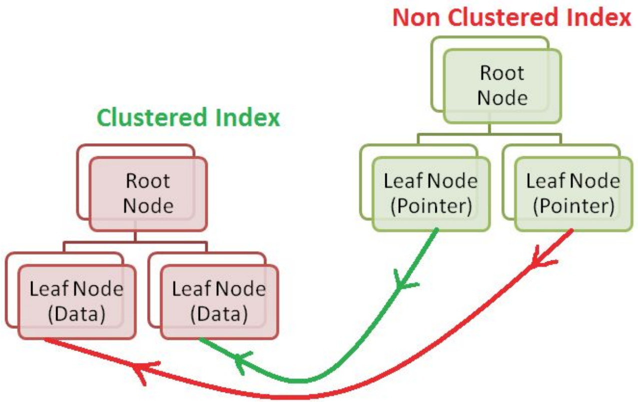
Clustered Index



- **Storing the indexed row directly within an index**
    - Telling the order of the data in the index
- Why?
    - Reads performance can be bad, when
        - the location of the heap file keeps changed, needing additional pointers
- Only one clustered index per table in SQL
    - In some system, primary key is always the clustered index
- Adds overhead on write requiring additional storage and consistency

Image source: https://medium.com/fintechexplained/clustered-vs-non-clustered-index-8efed55ed7b9

Concatenated Index: Multi-column Index

- **Combines several fields into one key**
- Why?
  - A single key is not enough when querying multiple columns simultaneously
- Examples
  - Geospatial data - latitude, longitude
  - Weather observation - date, temperature
  - Selecting colours for product search - red/green/blue
- B-tree or LSM-tree can’t handle these as it is
  - Needs translation process to R-tree

Other Indexing Structures

Full-text search and fuzzy indexes

- Search for similar keys
  - misspelled words, synonyms
- Lucene
  - For the term dictionary, SSTable-like structure with a small in-memory index is used to find the offset
- Techniques for Machine learning

In-memory database

- RAM becomes cheaper
- Big performance improvements claimed
  - Avoiding encoding in-memory data structures to be written in a disk
- Different data models from disk-based index
  - Priority queues & sets of Redis

Data Warehousing

Laine Kim

From <Designing Data-Intensive Applications>
by Martin Kleppmann

OLTP

Online Transaction Processing

- End user, customers
- Read: limited records per query
- Write: random, low-latency from user inputs
- Highly available

OLAP

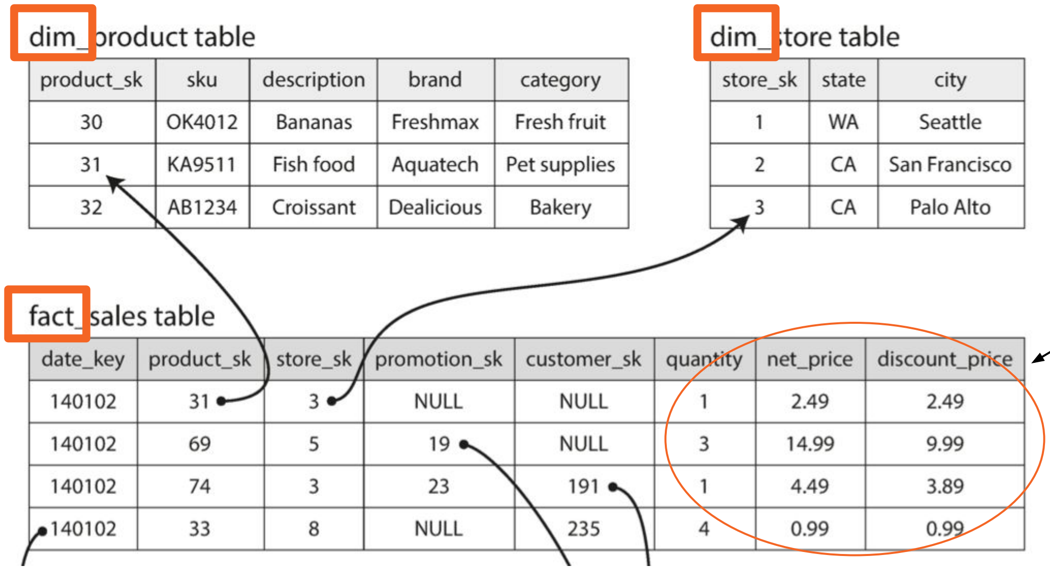
Online Analytic Processing

- Used for business analysis
- Read: Aggregate numerous historical records
- Write: Bulk import
- Larger dataset size than OLTP

Data Warehouse

- **Separate Database** from the OLTP system
- Read-only copy of the OLTP
- Optimized for analytic specific access patterns
- Writing is more difficult but LSM-trees is a solution
  - Not real-time write
  - First write in-memory store, being sorted and prepared for writing to disk until enough writes accumulated
- Most Commonly relational as SQL

Star schema for Analytics



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

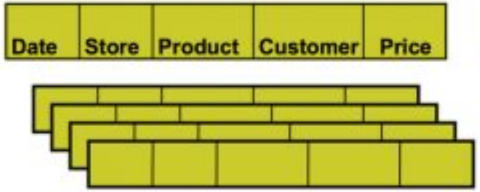
fact_sales table

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

Figure 3-9 Example of a star schema for use in a data warehouse,
<Designing Data-Intensive Applications>

- Fact table - event
  - ○ Often wide with over 100 columns
    - ■ Attribute columns
    - ■ Foreign key references to dim tables
- Dimension table - 6 W
  - ○ Smaller rows than fact tables
  - ○ If subdimensions: Snowflake schema

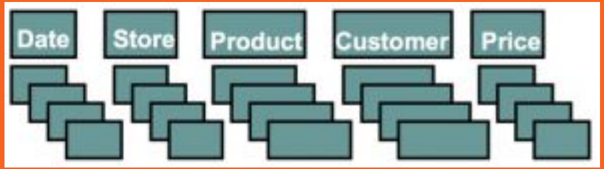
Row-Oriented Storage



- Each row is stored next to each other
- Slow to fetch a million rows for analysis
- Applicable to OLTP

Image source: https://insightsoftware.com/blog/mpp-columnar-databases/

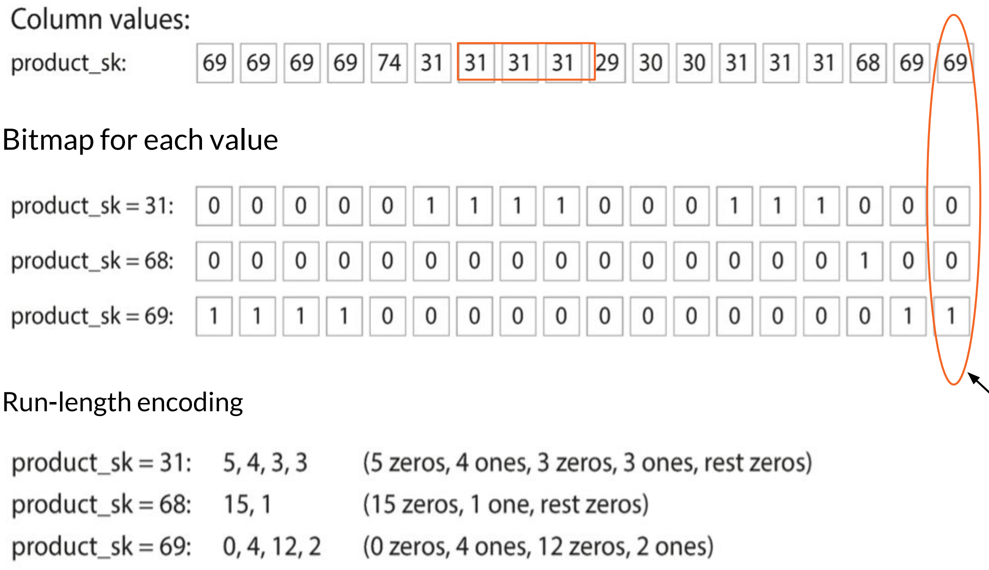
Column-Oriented Storage



- Store values from each column together
    - Rows in the same order
- Efficient use of CPU cycle
    - Compressed column data allows more rows to fit in the CPU's L1 cache

**L1 cache: The smallest and fastest type of cache memory, directly embedded into the CPU with the same speed*

Column Compression - bitmap encoding



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 value

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

Run-length encoding

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)

A row

Figure 3-11. Compressed, bitmap-indexed storage of a single column, <Designing Data-Intensive Applications>

- Example query
  - WHERE product_sk IN (30, 68, 69):
  - Load the three bitmaps with OR operation, efficiently!

Sort Order

- Sort an entire row at a time
    - Otherwise, can't reconstruct the row
- Set sort key to be the most frequently used column
    - Ex) often targeting date rages, *data_key* is the first sort key
- Sorted order helps the compression of the columns

Aggregation with Materialized Views

- Cache often used queries with aggregate functions
    - COUNT, SUM, AVG, MIN, or MAX in SQL
- Needs to be updated as data changes
- Precomputed, thus very fast
- Less complexity than raw data - limited usages

The End