

# TDT4225

## Chapter 3 – Storage and Retrieval

Svein Erik Bratsberg

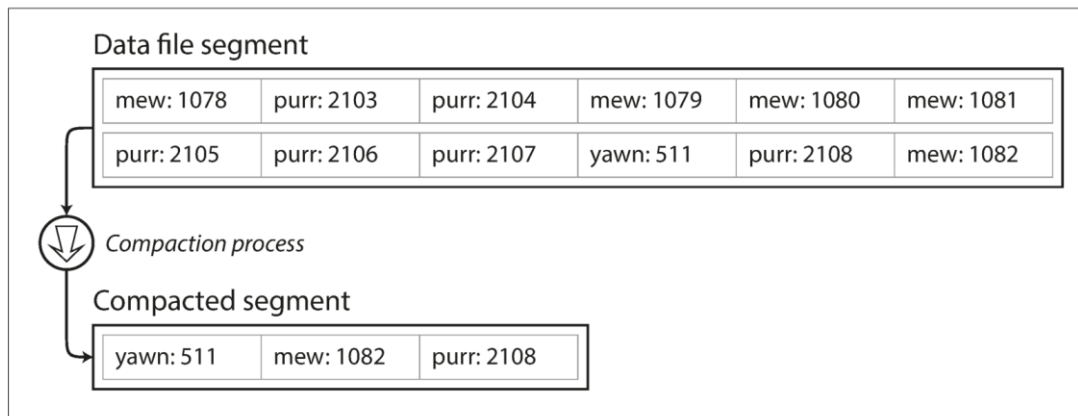
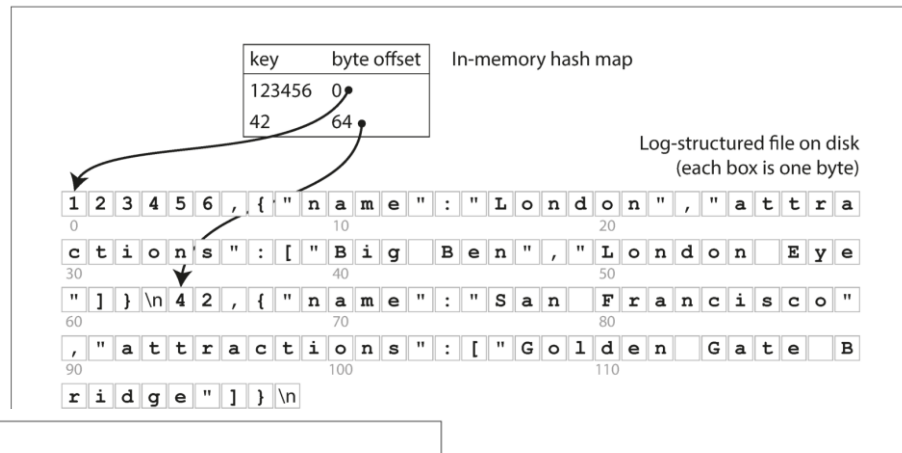
Department of Computer Science (IDI), NTNU

# Data structures that power your database

- The basic function of a database is to store data and later you could retrieve it
- Basic storage + indexes (heap file + index)
- To speed up retrieval you use indexes
- Indexes speed up queries, but slow down writes
- Thus, the application developer or the database administrator choose which indexes to be created based on the application and query pattern

# Hash indexes

- In-memory hash map + disk
- + compactions



in a CSV-like format, indexed with an in-

Figure 3-2. Compaction of a key-value update log (counting the number of times each cat video was played), retaining only the most recent value for each key.

# Merging + compaction

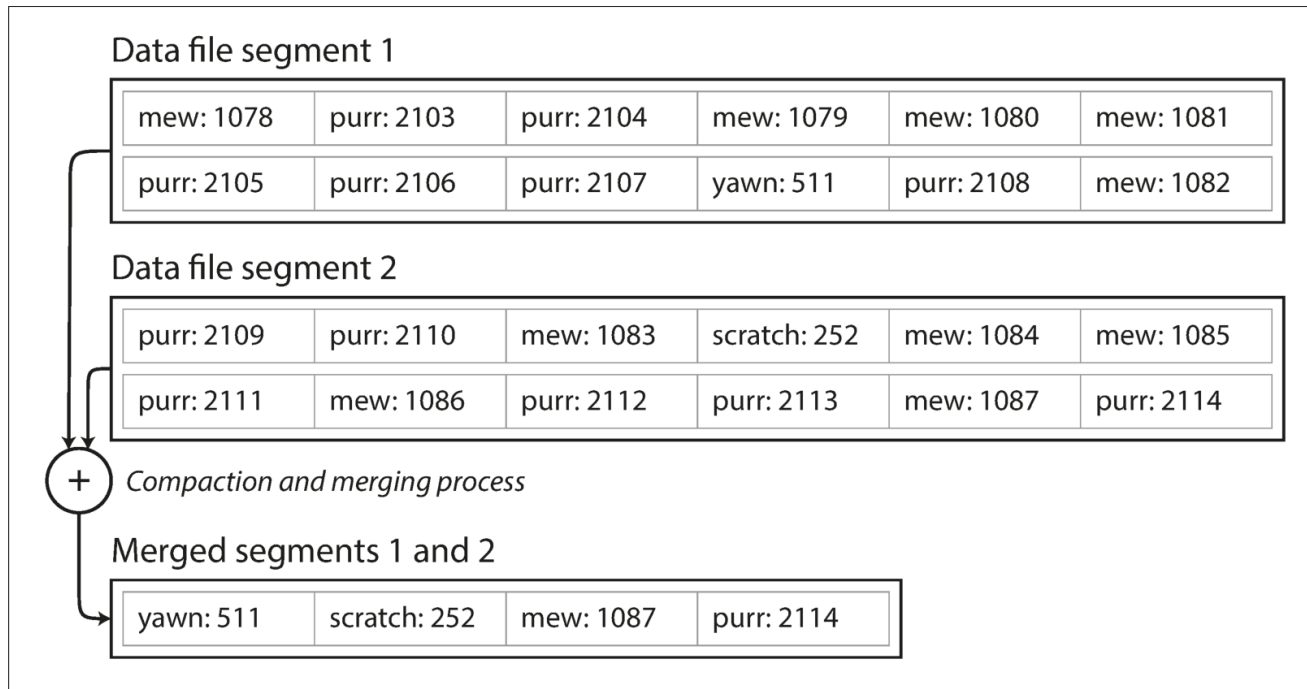


Figure 3-3. Performing compaction and segment merging simultaneously.

# Details

- File format: binary formats? Better than cvs.
- Deleting records (tombstones)
- Crash recovery: rebuild hash index. *Bitcask* stores a snapshot of the hashmap on disk
- Partially written records. Use checksums. Flip/flop on blocks.
- Concurrency control. Append only helps.
- Append only (vs in-place update)
  - Sequential writes
  - Concurrency and crash recovery become easier
  - Merging prevents fragmentation
- Hashmap must fit in-memory and range queries are problematic

# SSTables and LSM-Trees

- Sorted String Table (SSTable)
- Sorted tables
- Efficient merge, like merge-sort

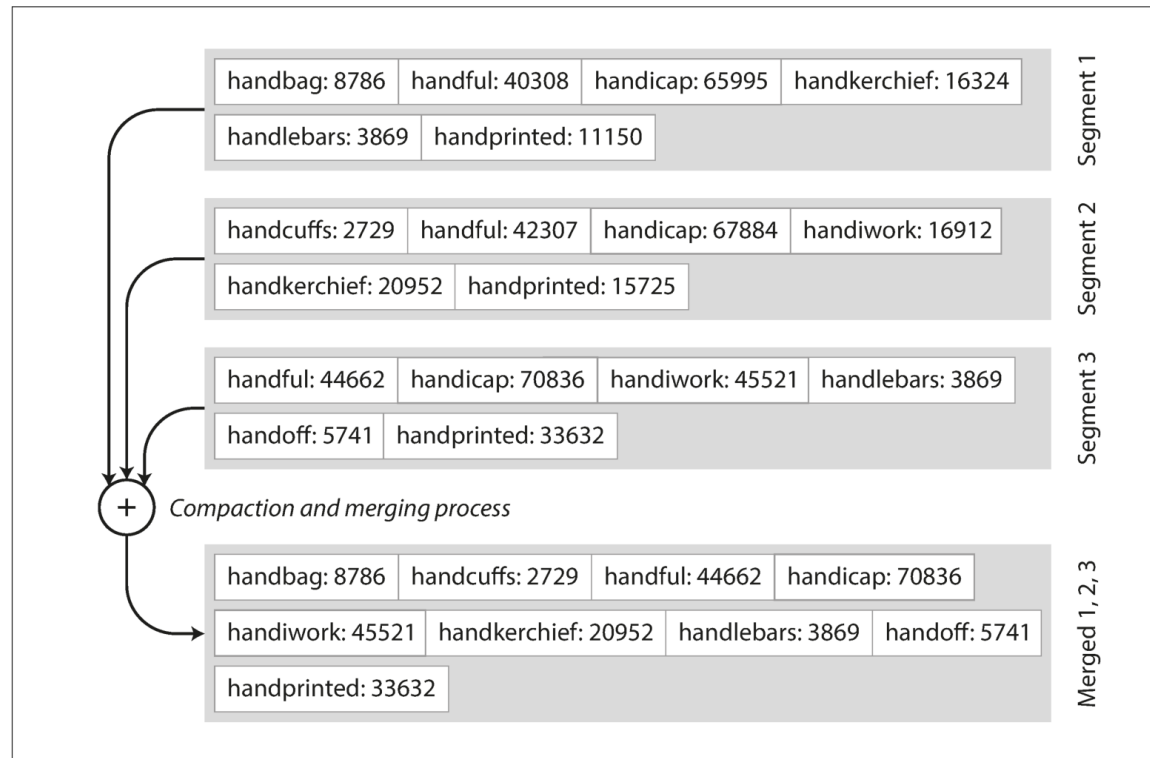


Figure 3-4. Merging several SSTable segments, retaining only the most recent value for each key.

# SSTables and indexes

- An index within each SSTable, e.g. SkipLists
- Bloom filter as well
- Compression is used as well

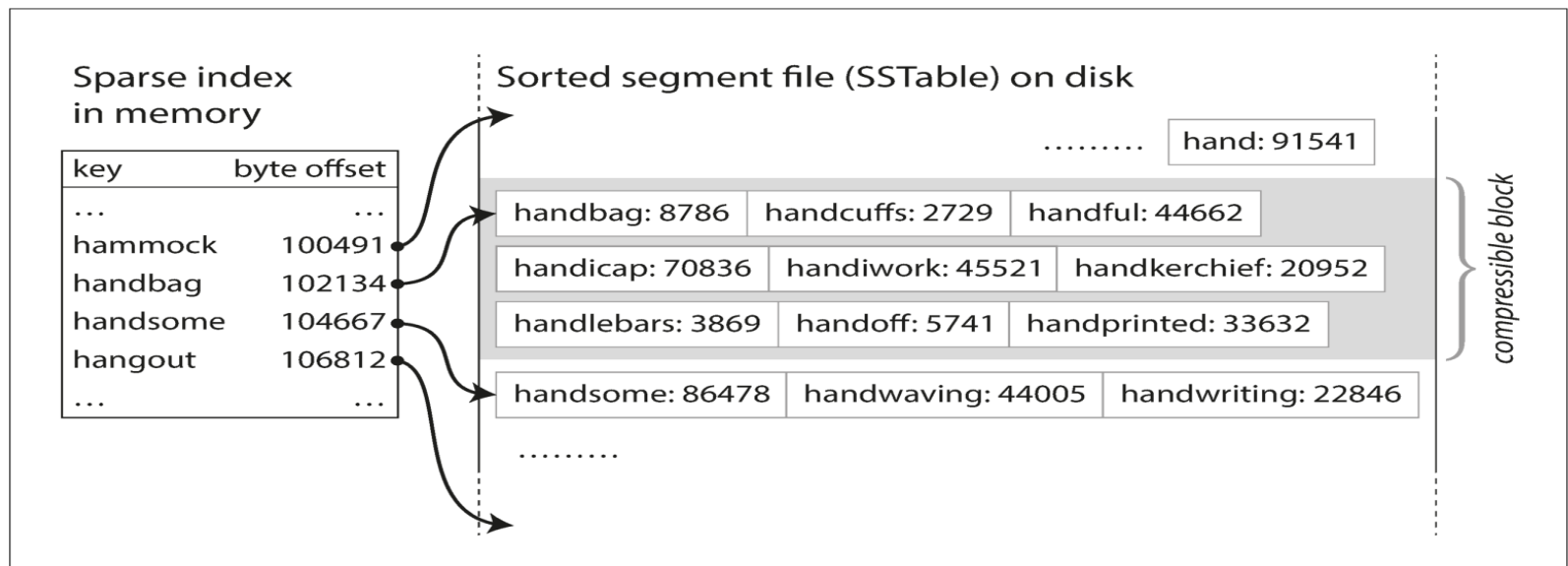


Figure 3-5. An SSTable with an in-memory index.

# Constructing and maintaining SSTables

- Sort the records in MemTable using some type of tree, e.g. SkipLists as in LevelDB/RocksDB.
- When MemTable gets bigger than some threshold, write it as an SSTable to disk.
- While writing MemTable, a new MemTable is used for new inserts.
- At read, try MemTable, then SSTables at disk ...
- Merge and compact SSTables.
- Use a write-ahead log to use in case of crash (and MemTable is lost)
- Log may be discarded when MemTable is written to disk



# LSM-trees

- LevelDB (Google) and RocksDB (Facebook) are LSM-based key-value storage engines
- Riak may use RocksDB as an alternative to Bitcask.
- Cassandra and HBase use LSM-trees
- Idea created by O'Neil & O'Neil in 1996 inspired by log-structured file systems (Ousterhout)
- Google made BigTable
- Bloom filters are used within each MemTable and SSTable to easily filter out requests to non-existing keys.
- Compaction and merge: Size-tiered and level-tiered

# B+-trees

- The standard database method
- Height-balanced tree with blocks as nodes
- All «user records» are at leaf level («bottom»)
- Typical height: 2, 3 or 4.
- Minimum 50 % filldegree in blocks
- Average 67 % filldegree in blocks
- Records are sorted on the key, and the tree supports
  - Direct search on key
  - Range search
  - Sequential, sorted scans
  - Good for most uses, except heavy write (insert) load

# B+-trees vs LSM-trees

- Writes faster for LSM-trees due to low write amplification
- Reads faster for B+-trees?
- B+-trees must write pages even if only a few bytes are updated
- LSM trees can provide a higher write throughput due to larger write units and sequential writes
- LSM trees can be compressed better due to larger units
- Compaction in LSM trees may interfere with ongoing reads and writes
- There is no quick and easy rule: You have to test which is best for you.

# Secondary indexes, heap files and clustered indexes

- Secondary indexes may be created. Often, they are not unique. Must store multiple values per search key.
- A value here is either a pointer (RecordPtr) or another key.
- Heap files store records in insert order, and indexes are used to provide uniqueness and quick access
- Often, records are stored within the index itself, i.e. Clustered index
- MySQL InnoDB and tables in SQL Server use clustered B+-trees

# Multi-column indexes

- Concatenated index, e.g. lastname + firstname
- Multiple dimensions (spatial index)

```
SELECT * FROM restaurants WHERE latitude > 51.4946 AND latitude < 51.5079  
AND longitude > -0.1162 AND longitude < -0.1004;
```

- Regular B-trees don't support spatial indexes
- Space-filling-curves (Z/Morton/Hilbert) could be stored within a B+-tree
- R-trees have multiple dimensions built-in
- Could be used for multiple dimensions in general, e.g. age and location

# Full-text search and fuzzy indexes

- Inverted index to support full-text search
- Many search engine support database-like functionality, e.g. schemas
- They often have a very coarse-grained update policy, e.g. merge in a new index into a bigger one.
- Lucene allows searches with edit-distance, e.g. misspelled words and other linguistic tricks (synonyms etc.)

# Keeping everything in-memory

- Disks require careful layout to become performant
- In-memory databases: Memcached, VoltDB, MemSQL, Oracle TimesTen.
- RAMCloud, Redis and Couchbase are memory-oriented
- They use a memory-based layout of data, and not encoded for disk
- Anti-caching: Store everything in memory, but evicting the least recently used data item (record) to disk
- Non-volatile memory (NVM) could change the architecture of databases

# Transaction processing or analytics

- Traditionally, databases were used for business processing.
- OLTP -- Online transaction processing is used for any access pattern where you need answers quickly
- OLAP – Online analytic processing is used for access patterns which do analysis of large datasets

*Table 3-1. Comparing characteristics of transaction processing versus analytic systems*

Property	Transaction processing systems (OLTP)	Analytic systems (OLAP)
Main read pattern	Small number of records per query, fetched by key	Aggregate over large number of records
Main write pattern	Random-access, low-latency writes from user input	Bulk import (ETL) or event stream
Primarily used by	End user/customer, via web application	Internal analyst, for decision support
What data represents	Latest state of data (current point in time)	History of events that happened over time
Dataset size	Gigabytes to terabytes	Terabytes to petabytes



# Data warehousing

- An enterprise may have dozens/hundreds of OLTP systems
- Most of these are operated by separate teams
- A data warehouse integrates such for analytic purposes

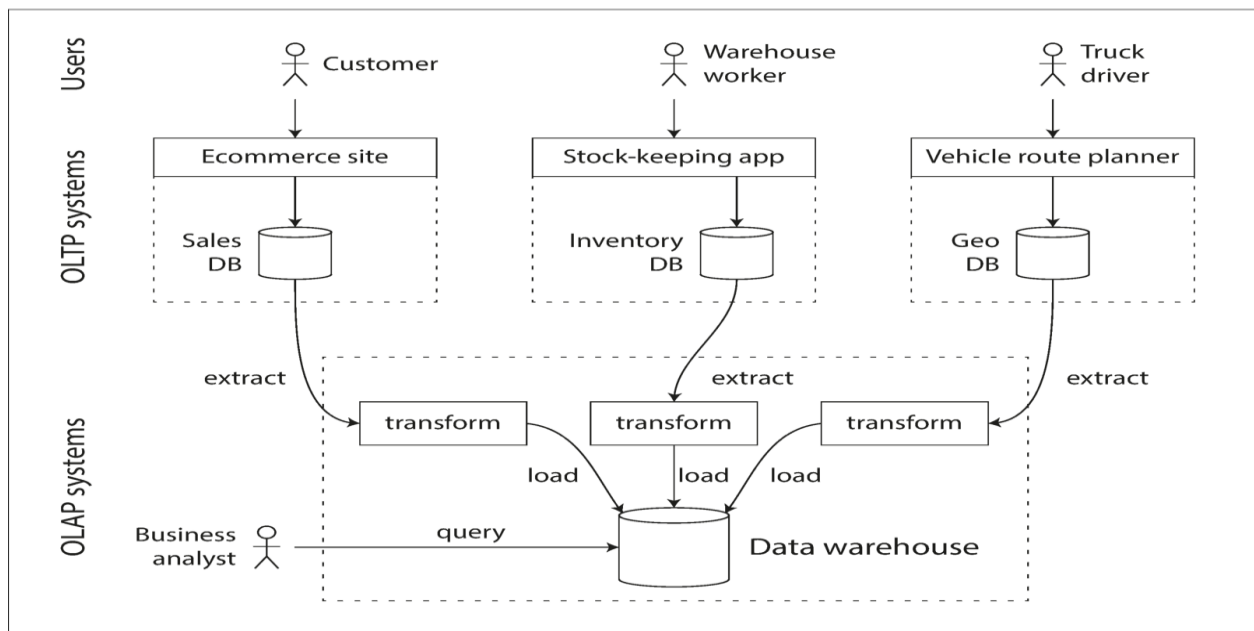


Figure 3-8. Simplified outline of ETL into a data warehouse.

# OLTP vs data warehouses

- Data warehouses use SQL, but may have other storage backends using other types of indexes.
- Column stores vs. row stores
- MS SQL server and SAP Hana support both OLTP and OLAP
- Terradata, Vertica, ParAccel, SAP Hana sell commercial data warehouses
- Apache Hive, Spark SQL, Cloudera Impala, Facebook presto, etc. are open source alternatives

# Star Schemas and Snowflakes

- *Star schemas* or dimensional modeling
- *Snowflakes* are schemas, where dimensions may be normalized

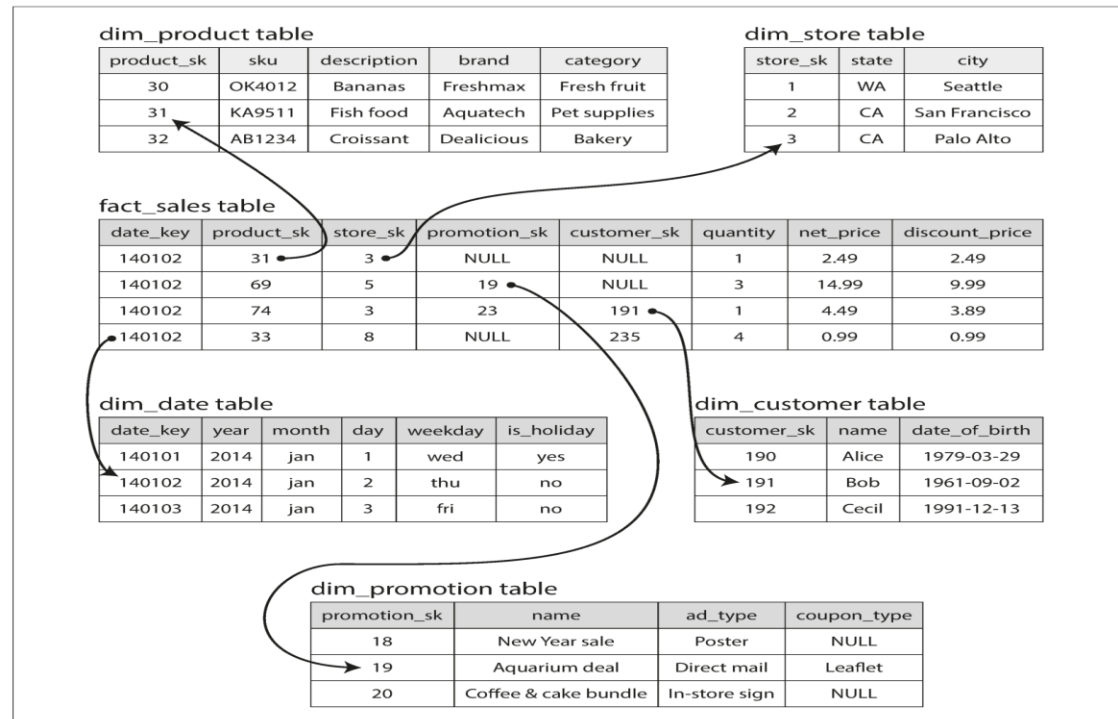


Figure 3-9. Example of a star schema for use in a data warehouse.

# Column stores (1)

- Row storage for OLTP and column storage for OLAP

*Example 3-1. Analyzing 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 = 2013 AND
    dim_product.category IN ('Fresh fruit', 'Candy')
GROUP BY
    dim_date.weekday, dim_product.category;
```

# Column stores (2)

- Store all columns of the same table in same (row) order

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

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

- Bitmap encoding: Well suited for «where c in (69, 31)»

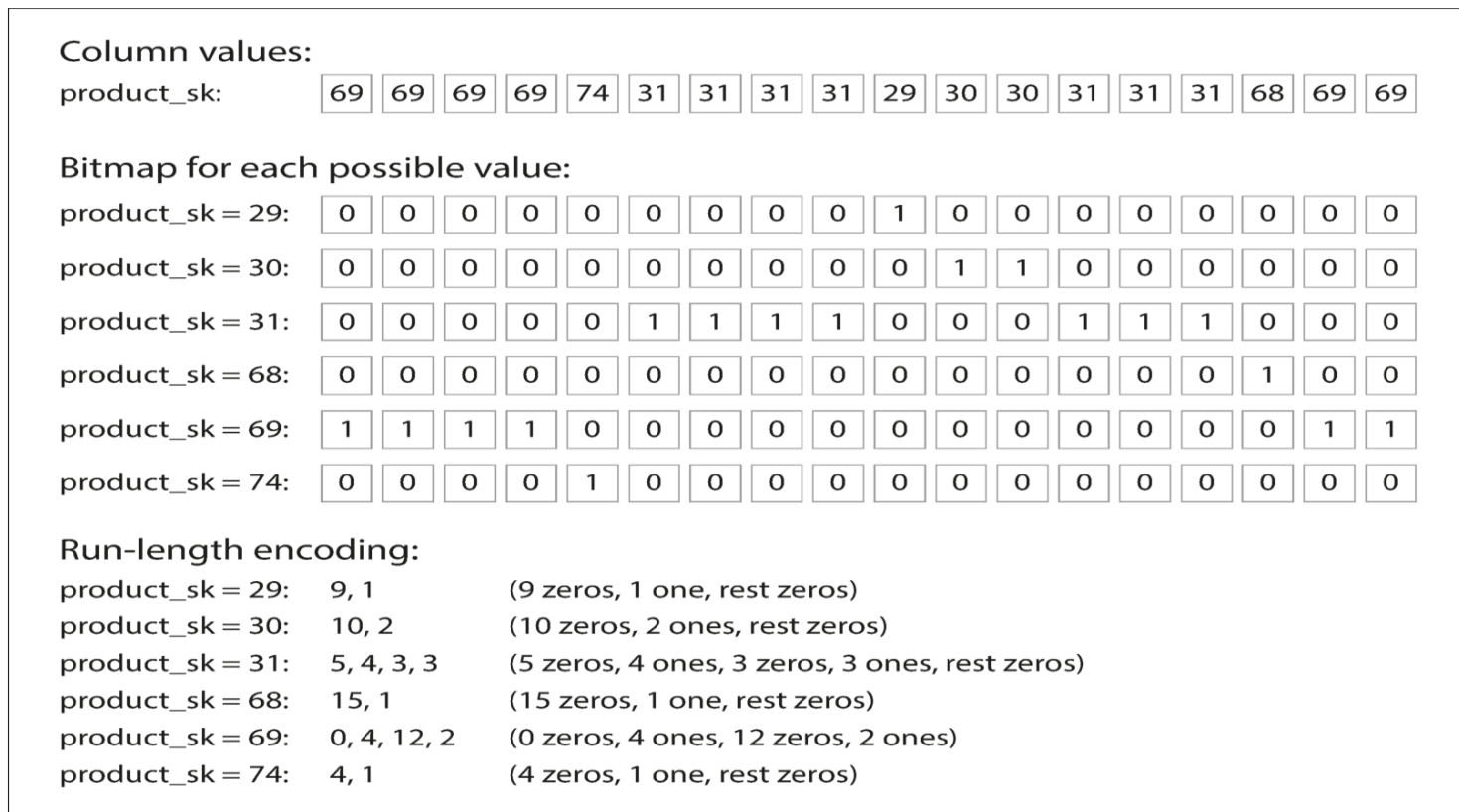


Figure 3-11. Compressed, bitmap-indexed storage of a single column.

# Various data warehouse techniques

- Vectorized processing: Avoid branching, Looping over L1 cache. Bitwise operations, etc.
- Sorting rows according to e.g. date? Typical for queries.
- Store the same data in different sort orders (Vertica)
- Writing to a column store: Use a write store, which further distributes data to columns through merge operations
- Materialized aggregates: COUNT, SUM, AVG, MAX, etc
- Materialized views: Used a lot in read-heavy environments

# Data cubes / OLAP cubes

- Aggregates in several dimensions: Making some queries faster

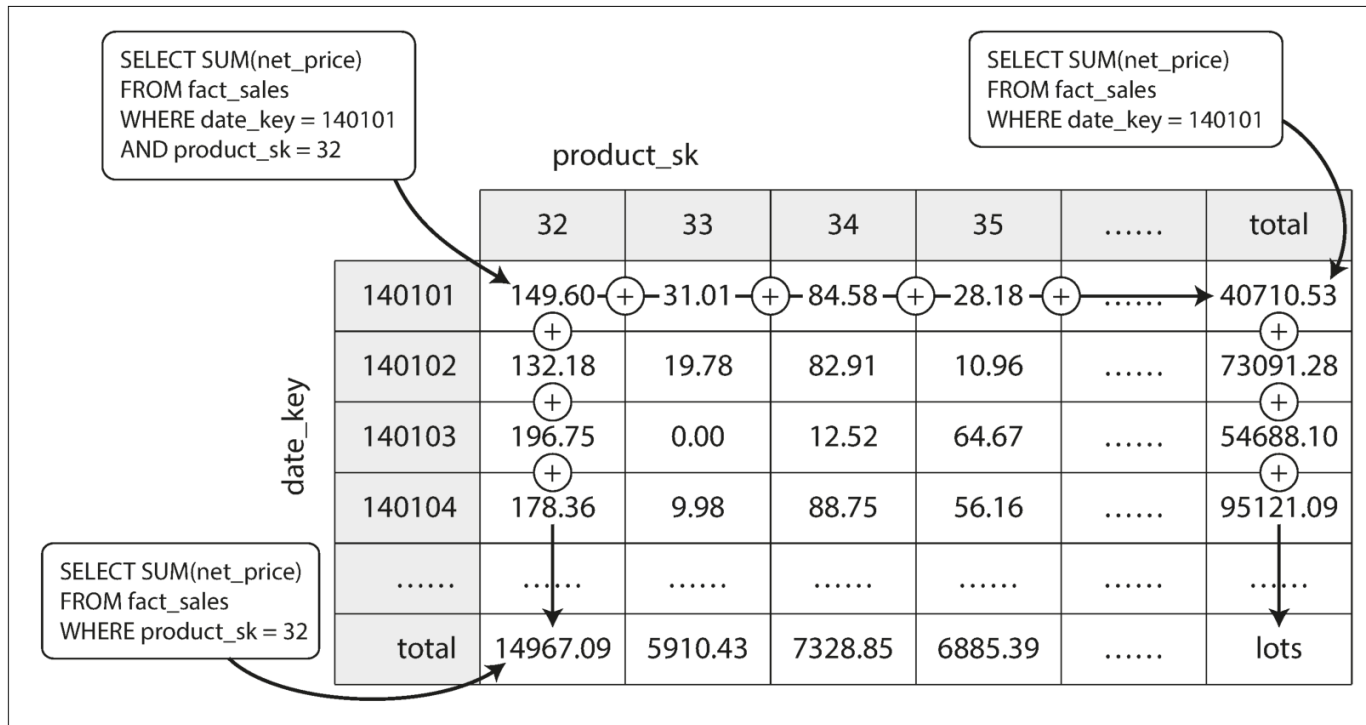


Figure 3-12. Two dimensions of a data cube, aggregating data by summing.



# Summary

- OLTP databases: Online users
  - Log structured: LSM trees
  - Update in place: B+-trees
- OLAP databases: System analytics
  - Column stores
  - Data cubes