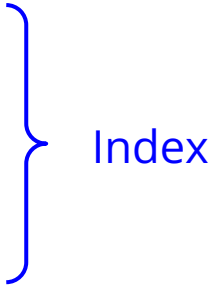
Designing Data-Intensive Applications

Wei Yao
12/11/2022

Chapter 3: Storage and Retrieval

Database Engine solve the fundamental questions:

- Store the data
- Retrieve data



This chapter is aiming to answer what is storage engine doing under the hood?

OLTP vs OLAP

这两类主流需求的差异，决定了所采用的 Database engine差异。OLTP重在小而快的大量 读写需求，OLAP重在庞大数据的整体层面上的分析需求。

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

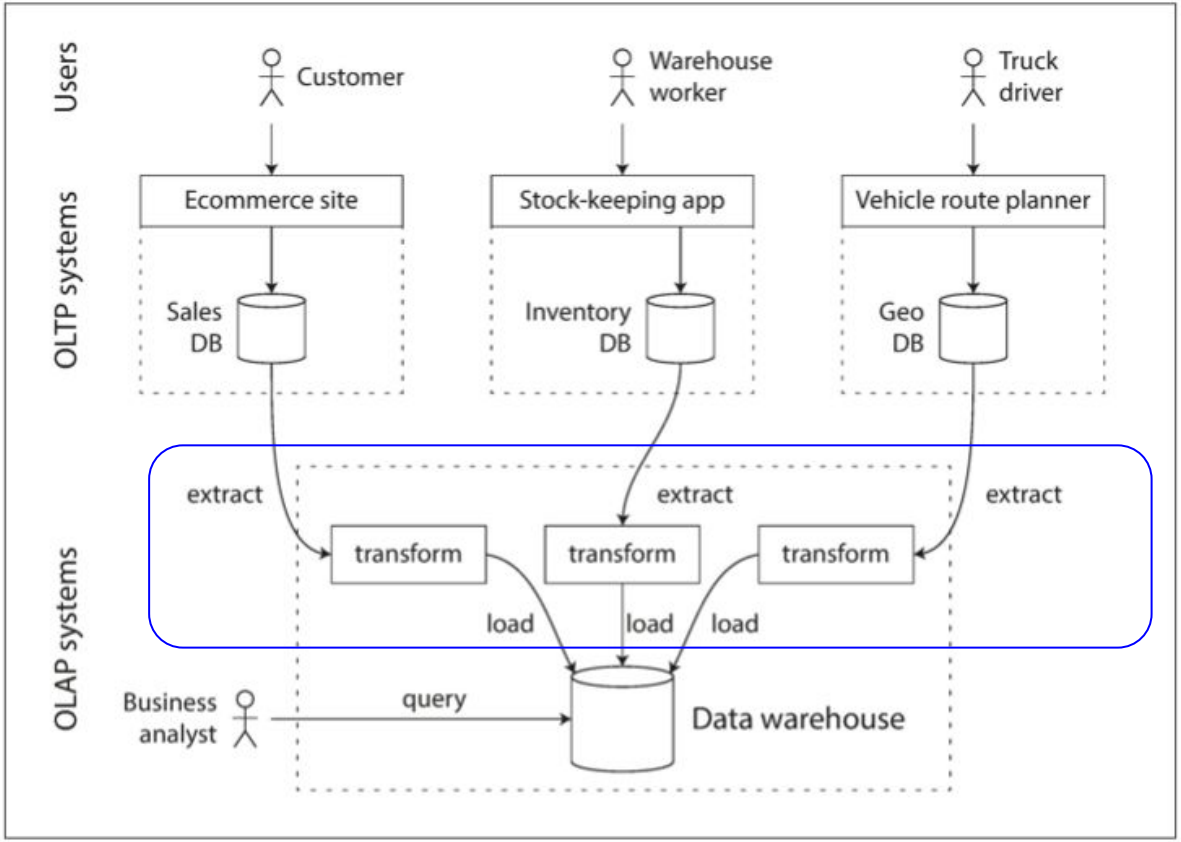


Figure 3-8. Simplified outline of ETL into a data warehouse.

ETL: Extract-Transform-Load

Both data warehouse and a relational OLTP database have a SQL query interface.

OLTP database engine includes:

Disk database:

- LSM-Tree: HBase, Cassandra, LevelDB, MongoDB and etc.
- B-Tree: all major relational databases

In-memory database:

- Memcached, VoltDB, MemSQL, RAMCloud, Redis and Couchbase.

OLAP data warehouses includes:

Teradata, Vertica, SAP HANA, Amazon Redshift, Apache Hive, Spark SQL, Cloudera Impala, Facebook Presto, Apache Tajo and Apache Drill.

Two Families of OLTP Storage engines:

Log-structured storage engines -> LSM-Tree

Page-oriented storage engines -> B-Tree

本质上，这两类实现方式都是为了解决一个如何index的问题。

What is Index?

- An index is an additional structure that is derived from the primary data.
- In database, removal or adding index does not affect the content of database, but only affects the performance of queries.
- Index usually slows down writes, because the index also needs to be updated every time data is written.

Trade off:

Need indexes to speed up read queries, but every index slow down writes.

Hash Indexes

- Append-only log on disk
- Key-value stores can be implemented with a hash map (a.k.a. hash table)
- In memory and every key mapped to a byte offset in the data file on disk.

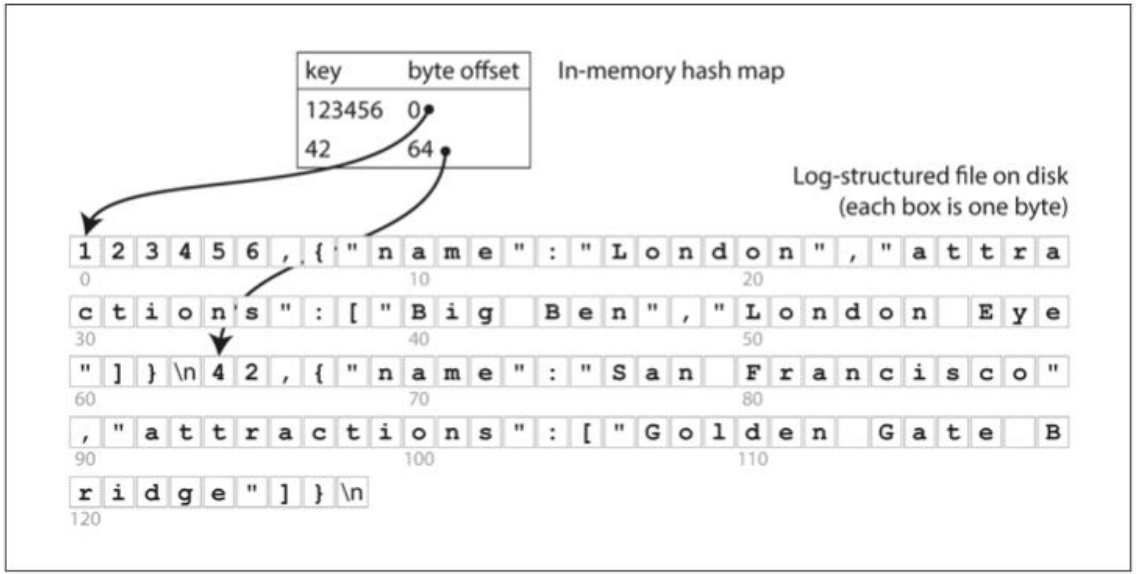


Figure 3-1. Storing a log of key-value pairs in a CSV-like format, indexed with an in-memory hash map.

How to deal with data increase over the time, when the memory can’t hold all keys?

- Segment solution with compaction
- Merge segment at some time
- Deletion is achieved by adding a special marker.

When read, first check most recent segment.

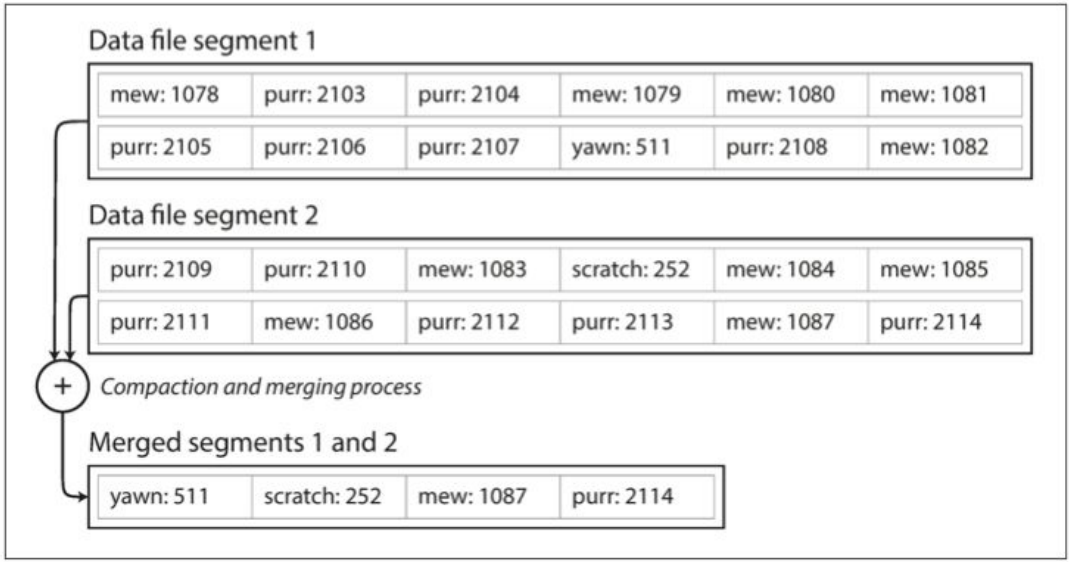


Figure 3-3. Performing compaction and segment merging simultaneously.

SSTables: Sorted String Tables

Sorted key in the segment files.

1. Merging segments is simple like merge sort algorithm.
2. No longer need to keep all index key in memory, but need to know offset.
3. Compress block to save disk space and reduce I/O bandwidth use.

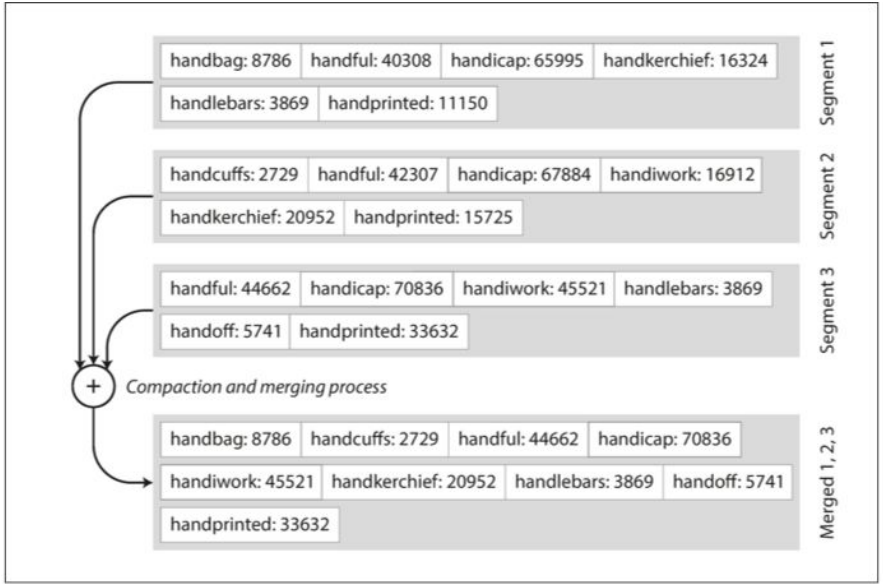


Figure 3-4. Merging several SSTable segments, retaining only the most recent value for each key.

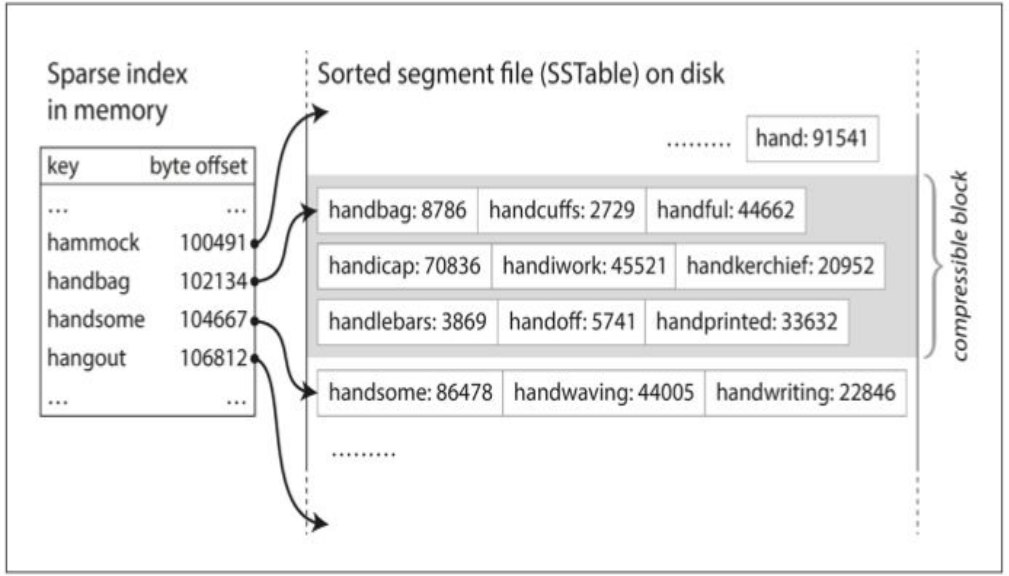
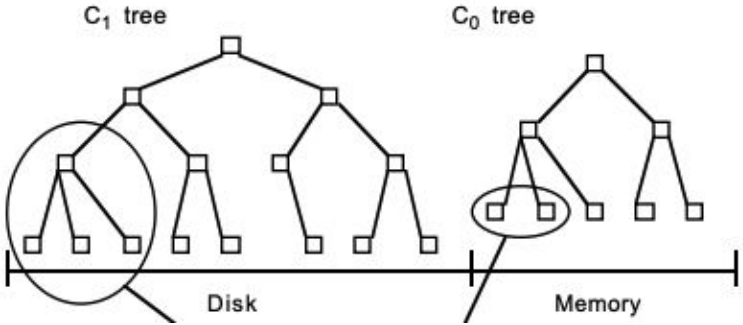


Figure 3-5. An SSTable with an in-memory index.

LSM-Trees

- The Log-Structured Merge-tree (LSM-tree) is a disk-based data structure designed to provide low-cost indexing for a file experiencing a high rate of record inserts (and deletes) over an extended period.
- The LSM-tree uses an algorithm that defers and batches index changes, cascading the changes from a memory-based component through one or more disk components in an efficient manner reminiscent of merge sort.
- https://www.cs.umb.edu/~poneil/lsmtree.pdf



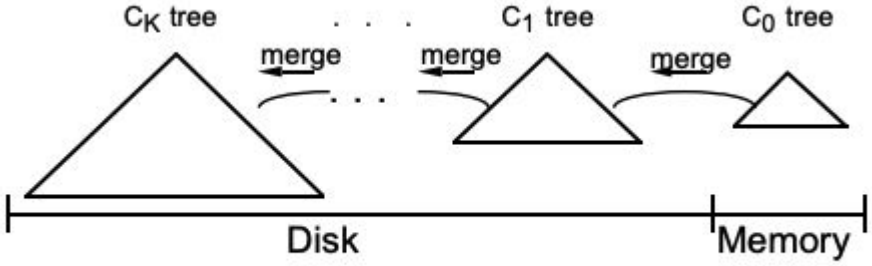


Figure 3.1. An LSM-tree of K+1 components

Performance optimizations mainly focus on the order and timing of how SSTables are compacted and merged.

- LevelDB and RocksDB use leveled compaction
- HBase use size-tiered compaction
- Cassandra supports both

B-Trees: most widely used indexing structure

A self-balancing search tree, multiple child nodes, each node can hold more than one keys.

B-trees break the database down into fixed-size(4 KB) blocks or pages, and read or write one page at a time.

Write operation is to overwrite a page on disk with new data.

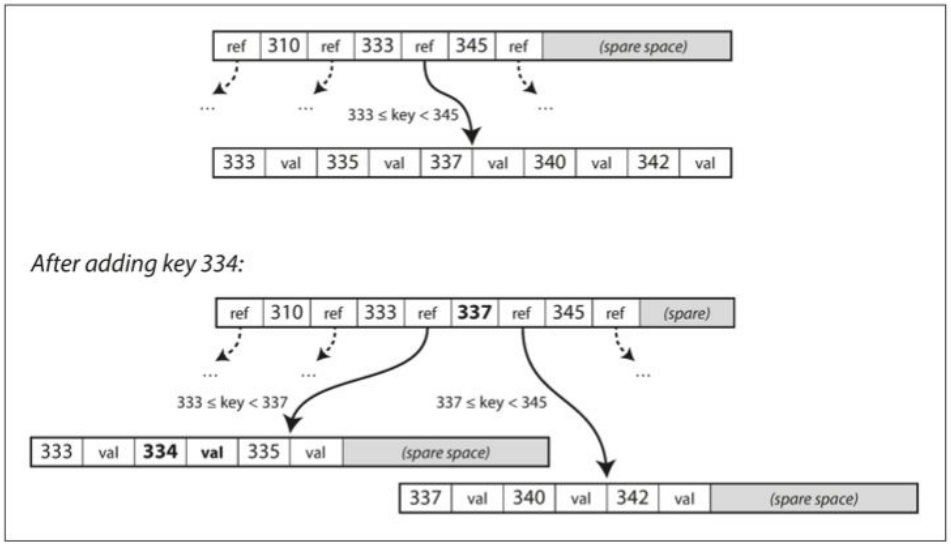


Figure 3-7. Growing a B-tree by splitting a page.

B-Trees optimizations

Making B-trees reliable

- Writing in place and page splits are dangerous operations to be interrupted by a server crash
- Most B-tree implementations use a *write-ahead log* (WAL, or *redo log*) on disk. Modifications are written here prior to updating the B-tree
- Concurrency control is required, typically with lightweight locks *latches*.

Other optimizations

- Abbreviating long keys, so to pack more keys into a page
- Theoretically, pages can be positioned anywhere on disk, but can save scan time by laying out pages in sequential order.
- Additional pointers added to the tree, like leaf page may have reference to its sibling pages.

Comparing B-Trees and LSM-Trees

	B-Trees	LSM-Trees
Mature	More mature	Less mature
Faster On	Read	Write
Write Amplification	More with writing entire page	Less with SSTable compaction
Disk usage	More due to internal fragmentation	Compressed better
Others	Transactional functionality is easier, each key exists in one place	Background compaction can slow down writes

OLAP system schema

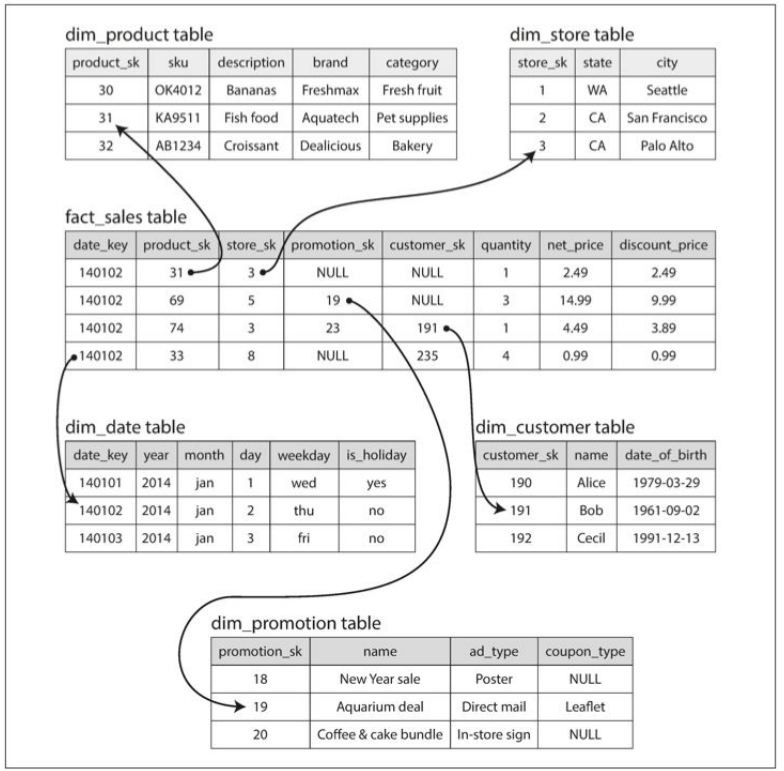
Star schema (dimensional modeling)

- Fact table
- Dimension tables

Snowflake schema: dimensions are further broken down into subdimensions.

In a typical data warehouse, tables are often very wide: fact tables often have several hundred columns.

Column-Oriented Storage



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

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.

Column-Oriented Storage

OLTP database: row-oriented fashion.

Tables are wide in data warehouse, but analytic query only needs a small number of many columns in a fact table.

Column-oriented storage: don't store all values from one row together, but keep rows in the same 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

Column compression

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

Data Cubes and Materialized Views

This is other options to column-oriented storage in data warehouse.

Materialized Views:

A table-like object whose contents are the result of some query from the original tables.
When data changes, the materialized view needs to be updated.

Data cube:

It is a grid of aggregates groped by different dimensions.
Especially useful in OLAP for fast read of aggregate results, because of pre-computation.

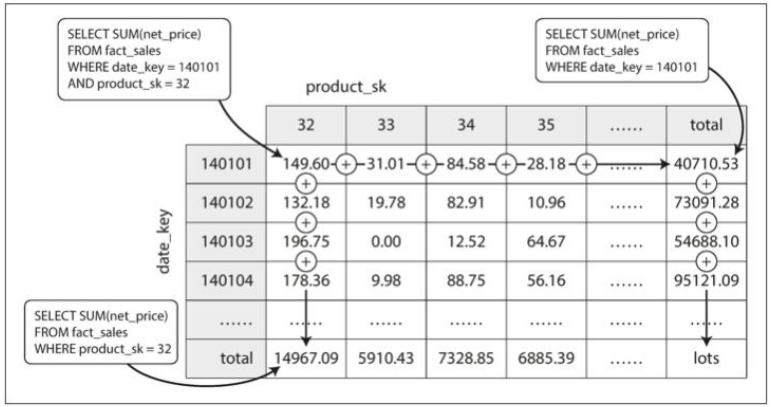


Figure 3-12. Two dimensions of a data cube, aggregating data by summing.

Thank you!