Analytics Databases

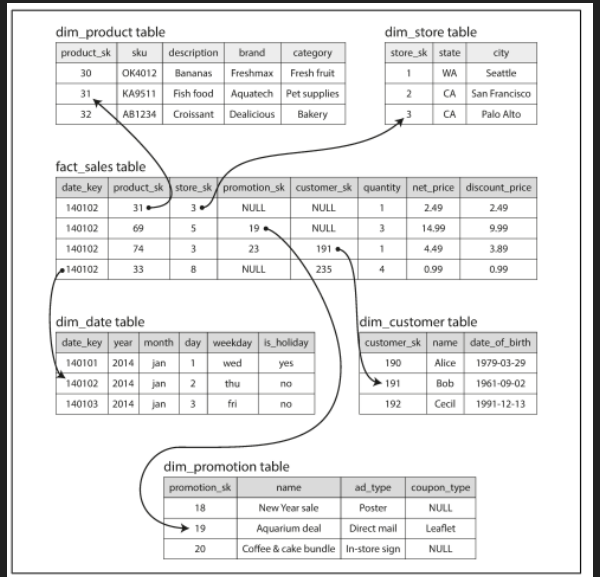
Analytics Databases Background

Businesses will often want to run large internal queries across their data, that do full table scans. However, doing such a thing can take a huge performance hit on their database that deals with client interaction. Hence, they will typically have a second database, for analytics processing, where data is copied some period of time after the fact.

This is done using an ETL (Extract, Transform, Load) process which is typically scheduled as a batch job (we will discuss these later).

Stars and Snowflakes

In the transaction database, there may be many tables with all different types of relationships between them. However, in an analytics database, there is typically one central table known as the “fact” table. The fact table has many foreign keys which reference other tables known as dimension tables - this is called the “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

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.

If the dimensions tables reference sub-dimensions tables, this is known as a snowflake schema.

Column Oriented Storage Background

Most transactions based databases use row oriented storage: they store the entire contents of a row together on disk to improve with locality.

However, in analytical queries, it is rare that we need the entire row, but rather are aggregating the value of one column over a certain table. Hence, it makes more sense to store the columns of the table together, known as column oriented storage.

Note that each column must be stored in the same order.

Compression

If each column has a lot of duplicate values, we can compress it to save space.

Imagine the following column representing coding proficiency out of 10:

8, 8, 8, 4, 5, 2, 2, 2, 2, 1

Compression

If each column has a lot of duplicate values, we can compress it to save space.

Imagine the following column representing coding proficiency out of 10:

8, 8, 8, 4, 5, 2, 2, 2, 2, 1

Bitmap encodings:

For val 8: 1 1 1 0 0 0 0 0 0 0

For val 4: 0 0 0 1 0 0 0 0 0 0

For val 5: 0 0 0 0 1 0 0 0 0 0

For val 2: 0 0 0 0 0 1 1 1 1 0

For val 1: 0 0 0 0 0 0 0 0 0 1

Compression

If each column has a lot of duplicate values, we can compress it to save space.

Imagine the following column representing coding proficiency out of 10:

8, 8, 8, 4, 5, 2, 2, 2, 2, 1

Bitmap encodings:	Run-Length encodings:
For val 8: 1 1 1 0 0 0 0 0 0 0	For val 8: 0, 3, 7
For val 4: 0 0 0 1 0 0 0 0 0 0	For val 4: 3, 1, 6
For val 5: 0 0 0 0 1 0 0 0 0 0	For val 5: 4, 1, 6
For val 2: 0 0 0 0 0 1 1 1 1 0	For val 2: 5, 4, 1
For val 1: 0 0 0 0 0 0 0 0 0 1	For val 1: 9, 1

Compression Continued

Perform bitwise operations on the encodings to find rows where multiple fields match certain values (column A = 10 AND column B = 20, could also do column A = 10 OR column A = 15)

Allows more data to fit in CPU cache

If you want columns sorted in a different way, can have a replica of the analytics database with a different sort order:

- Acts as an index for efficient querying if you have a common query pattern
- Allows more column compression

Writing to Column Oriented Storage

Inefficient writes because would have to modify every column file

Instead all writes go to a sorted tree in memory (LSM tree), which is eventually written in bulk to all of the column files once it gets too big

- Reads must check both the tree and the column files and merge them

Materialized Views

Idea: Database precomputes common queries so they do not constantly have to be rerun

Pros:

- Do not have to rerun certain expensive common aggregations

Cons:

- Writes take longer since materialized views must be updated
- Less flexibility than querying raw data

Data Cube

Idea: Special type of materialized view that precomputes a multi dimensional table (e.g. sales of every product_id on every day in the database)

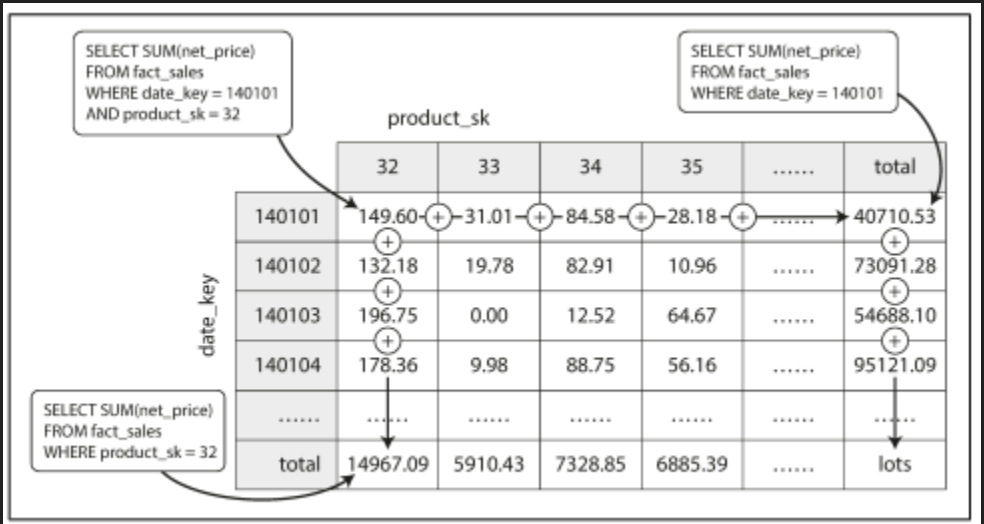


Figure 3-12. Two dimensions of a data cube, aggregating data by summing.

Analytics Databases Summary

Very useful to decouple analytics databases from transactional ones as analytics queries can take a very long time.

Additionally, using an analytics database allows us to store our data in a column oriented manner, which once compressed, can be better utilized by a CPU's cache and iterated on in tight loops (no function calls). When writing to column oriented storage, we can use an LSM tree as a buffer.

Finally, materialized views are a potentially very useful caching mechanism by analytics databases in order to avoid recomputation of popular aggregations, at the cost of slower writes.