Barcelona GSE Data Science Center

Data Warehouses

Nandan Rao

September, 2019

Normalized Shmormalized

It's worth reviewing *why* we wanted normalization. We saw that normalization:

1. Saved space
2. Made updates faster
3. (This made ACID easier)

Normalized Shmormalized

Space is cheap.

What if we aren’t updating our data in the same way we were in an operational database?

Normalized Shmormalized

In analytics, we want to *read* the data and read *lots* of data.

It should be clear that this is an entirely different paradigm than that of the operational database we saw last week.

Operational vs Analytics (OLTP vs OLAP)

Operational: Many small reads and many updates from thousands of users who are interested only in “their” rows. Data integrity, consistency, is important because the rest of the software relies on it.

Analytics: Few users, rarely writing (batch updates?), making *giant* read queries that might span *all* the rows, usually making aggregations. Importance is speed of reads and flexibility to write any needed query.

An Analytic Query

Let’s consider our ecommerce database of classic models.
Here’s a simple question we might want to ask:

What were our profits per month?

What was the most profitable office?

Let’s design the simplest table to answer these questions.

Star Schema

The star schema will consist of two main parts:

1. Measure (center of the star)
2. Dimensions (the points of the star)

Measures / Fact Tables

These are the things we want to “measure,” and generally correspond one-to-one to a business process, often a “transaction” of some sort.

We think of these being big N! Many rows of the measure and we want to aggregate them.

Because we aggregate them, we usually think of these being continuous variables, preferably that are additive.

Grain! (Event sourcing!)

Dimension Tables

How do we aggregate the facts? We aggregate them by *dimensions*.

Dimension tables usually consist of the discrete variables by which we want to aggregate our measures to ask different questions.

Star Schema

(example and exercise)

Snowflaking

- Normalization - we could normalize the dimension tables.
- Generally an anti-pattern!
- Depends on the use case.

Big Data?

What happens when our data gets really big?

This further drives the design differences between operational and analytics databases.

Sharding

How do we deal with more data in a traditional, operational, SQL database?

We “partition” the data into little mini tables, and put each mini table on a different computer.

This is called sharding.

(draw it out)

Sharding

Sharding in operational databases is optimized, as would be expected, for concurrency. Many users want a small amount of data, so you try and put all the data that would belong to one user in one place (partition/shard).

If we shard by something like “primary key”, this makes the architecture very simple.

(example)

Parallelism

In operational databases, multiple cores in a CPU are usually used to handle concurrency – we are dealing with thousands of small requests at a time. Thus, devoting multiple cores of a CPU to a single request makes no sense, better to just devote different cores to different requests!

If you are dealing with one or two large requests at a time, you probably want all your cores working on that one request!

Columnar storage

If you remember the example on the first day, you should remember that *physical location* of the data matters a lot.

This was even more true with sharded/distributed systems, as we saw, but it’s also true on the disk itself.

Organizing data by rows makes sense when you often read/update/write single rows. But in analytics, we often perform large aggregations across entire columns! We care less about individuals, we care about aggregates.

We can write our data down by column, instead!

Redshift

AWS Redshift is a popular data warehouse tool that embodies these changes we talked about.

Redshift is not open source :(. *But* it is a very popular, hosted solution.

Redshift is basically: Postgres + Columnar + Parallel.

ETL

So now we have our data in a beautiful data warehouse. It has a schema, it’s hyper optimized, it’s columnar and distributed.

But how do you get your data from where it lives to where you will query it (data warehouse)?

This process is referred to as ETL: Extract, Transform, and Load.

ETL processes are basically slow. The idea, then, is that once "loaded" the data is some faster to query than where it was originally.

Where does your data come from?

Where do you extract from?

Traditionally, this was simply your operational database. Contemporarily, we have data all over the place:

- Many different operational systems (POS system, customer-facing website, employee systems, shipping and partner systems, etc.)
- Logs, logs, logs!
- Marketing data.
- Sales data (calls, contacts, attempts, responses, etc.)
- Accounting data.

Many sources, many heterogeneity

Data comes from many places, in many formats. How do we make queries or ask questions that span the entire business?

This is basically what ETL processes seek to solve.

Stepping Back

Like everything in this field, there is a lot of terminology used differently by a lot of people. The term “Data Warehouse” will be used in different ways by different people.

However, if we think of them in the “dimensional modelling” paradigm, data warehouses very much strike a balance between being optimized for certain queries and being flexible, but they are not *extremely* flexible by any means.

One lake, many houses

One solution will be to create a separate data warehouse for each need (team). This allows each team to work independently and create the warehouse needed, optimized for their types of queries.

In this case, however, it can make sense to separate the “E” from the “TL” (slight abuse of terminology), by creating a “data lake.”

A data lake is a place where all data in the entire organization goes to live, in raw format. It’s not optimized for querying, it’s optimized for dumping! Each warehouse, then, can pull the data they need and load it into their own warehouse.

Data Lake

Again, this term will be abused and used in different ways.

For us, we will think of a data lake as a place where data is stored “raw”, often in flat files (think csv or json files), in some sort of networked storage (more on that next week!)