DESIGN PATTERNS

Star Schema vs. Snowflake Schema

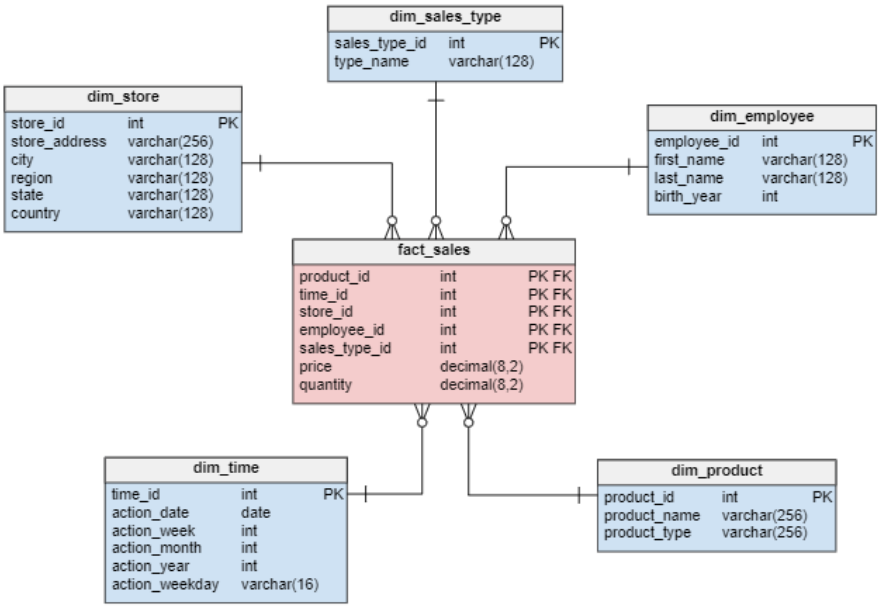
In the previous two articles, we considered the two most common data warehouse models: the star schema and the snowflake schema. Today, we'll examine the differences between these two schemas and we'll explain when it's better to use one or the other.

The star schema and the snowflake schema are ways to organize data marts or entire data warehouses using relational databases. Both of them use dimension tables to describe data aggregated in a fact table.

Everyone sells something, be it knowledge, a product, or a service. Storing this information, either in an operational system or in a reporting system, is also a need. So we can expect to find some type of sales model inside the data warehouse of nearly every company.

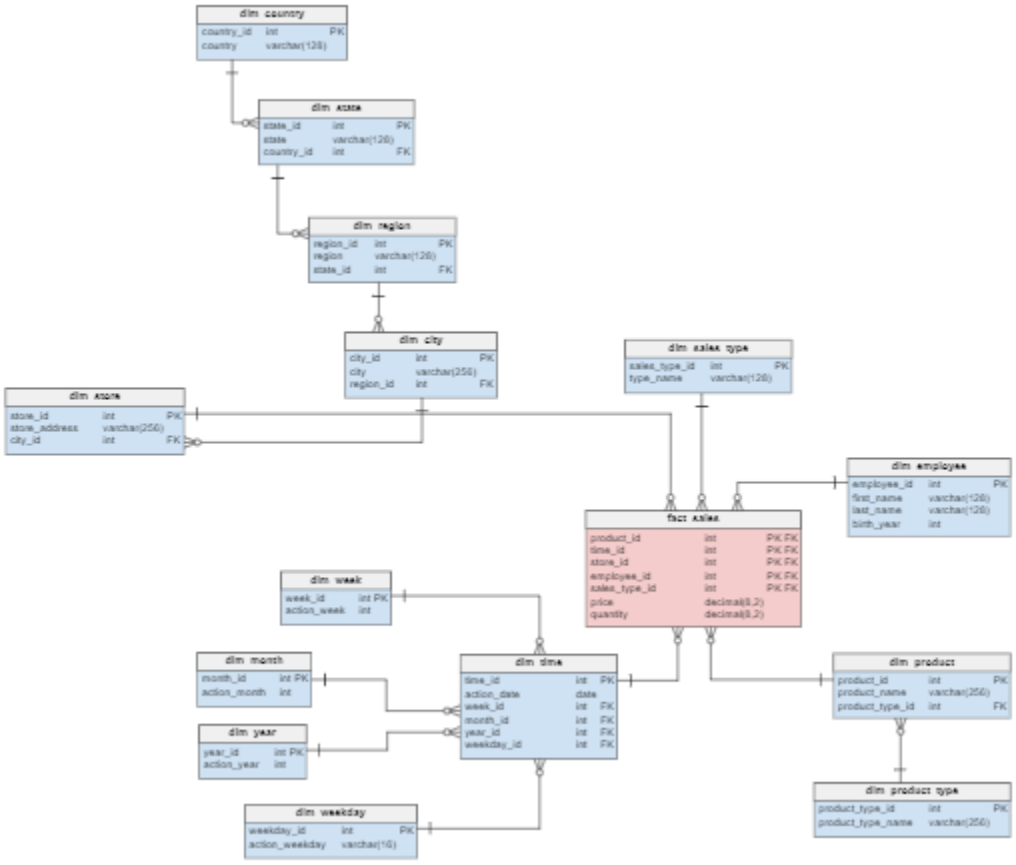
Let's take one more look at the sales model in both the star and snowflake schemas.

The Star Schema



The most obvious characteristic of the star schema is that dimension tables are not normalized. In the model above, the pink `fact_sales` table stores aggregated data created from our operational database(s). The light blue tables are dimension tables. We decided to use these five dimensions because we need to create reports using them as parameters. The granulation inside each dimension is also determined by our reporting needs.
From this model, we can easily see why this schema is called the ‘star schema’: it looks like a star, with the dimension tables surrounding the central fact table.

The Snowflake Schema



This snowflake schema stores exactly the same data as the star schema. The fact table has the same dimensions as it does in the star schema example. The most important difference is that the dimension tables in the snowflake schema are normalized. Interestingly, the process of normalizing dimension tables is called snowflaking.

Once again, visually the snowflake schema reminds us of its namesake, with several layers of dimension tables creating an irregular snowflake-like shape.

The First Difference: Normalization

As mentioned, normalization is a key difference between star and snowflake schemas. Regarding this, there are a couple of things to know:

- Snowflake schemas will use less space to store dimension tables. This is because as a rule any normalized database produces far fewer redundant records .
- Denormalized data models increase the chances of data integrity problems. These issues will complicate future modifications and maintenance as well.
- To experienced data modelers, the snowflake schema seems more logically organized than the star schema. (This is my personal opinion, not a hard fact. :))

Let's move on to the second major difference between these two schemas.

The Second Difference: Query Complexity

In our first two articles, we demonstrated a query that could be used on the sales model to get the quantity of all phone-type products sold in Berlin stores in 2016.

The star schema query looks like this:

```
SELECT
  dim_store.store_address,
  SUM(fact_sales.quantity) AS quantity_sold

FROM
  fact_sales
  INNER JOIN dim_product ON fact_sales.product_id = dim_product.product_id
  INNER JOIN dim_time ON fact_sales.time_id = dim_time.time_id
  INNER JOIN dim_store ON fact_sales.store_id = dim_store.store_id

WHERE
  dim_time.action_year = 2016
  AND dim_store.city = 'Berlin'
  AND dim_product.product_type = 'phone'

GROUP BY
  dim_store.store_id,
  dim_store.store_address
```

To get the same result from the snowflake schema, we have to use this query:

```
SELECT
  dim_store.store_address,
  SUM(fact_sales.quantity) AS quantity_sold

FROM
  fact_sales
  INNER JOIN dim_product ON fact_sales.product_id = dim_product.product_id
  INNER JOIN dim_product_type ON dim_product.product_type_id = dim_product_type.product_type
  INNER JOIN dim_time ON fact_sales.time_id = dim_time.time_id
  INNER JOIN dim_year ON dim_time.year_id = dim_year.year_id
  INNER JOIN dim_store ON fact_sales.store_id = dim_store.store_id
  INNER JOIN dim_city ON dim_store.city_id = dim_city.city_id

WHERE
  dim_year.action_year = 2016
  AND dim_city.city = 'Berlin'
  AND dim_product_type.product_type_name = 'phone'

GROUP BY
  dim_store.store_id,
  dim_store.store_address
```

Obviously, the snowflake schema query is more complex. Because the dimension tables are normalized, we need to dig deeper to get the name of the product type and the city. We have to add another JOIN for every new level inside the same dimension.

In the star schema, we only join the fact table with those dimension tables we need. At most, we'll have only one JOIN per dimension table. And if we're not using a dimension table, we don't even need to bother with it. In the snowflake schema query, we don't know how deep we'll have to go to get the right dimension level, so that complicates the process of writing queries.

Joining two tables takes time because the DMBS takes longer to process the request. The `dim_store` and `dim_city` tables are placed in close proximity in our model, but they may not be located anywhere near each other on the disk. There is a better possibility that data will be physically closer on the disk if it lives inside the same table.

Basically, a query ran against a snowflake schema data mart will execute more slowly. But in most cases this won't present a problem: it doesn't matter much if we get the result in one millisecond or one second.

Speeding Things Up

To speed up reporting, we can:

- Aggregate data to the level we need in reports. This will compress the data significantly. We'll need to create procedures that will transform our live data to fit into the reporting schema structure (the ETL process).
- Build a central storage area for all the company's aggregated data, not just the sales data.
- Only give users the data they need for analysis and reports.

Snowflake vs. Star Schemas: Which Should You Use?

Now that we've looked at theory and query speeds, let's get right into the heart of the matter: how do you know which schema to use on any given project?
Consider using the snowflake schema:

- In data warehouses. As the warehouse is Data Central for the company, we could save lot of space this way.
- When dimension tables require a significant amount of storage space. In most cases, the fact tables will be the ones that take most of the space. They'll probably also grow much faster than dimension tables. But there are certain situations where that doesn't apply. For instance, the dimension tables could contain a lot of redundant-but-needed attributes. In our example, we used the *city* attribute to describe the city where the store is located. What if we wanted a much more detailed description of the city, including the population, postal code, demographic data, etc.? Describing other subdimensions – for example, *store*, *region*, *state* and *country* – with more attributes would turn the `dim_store` dimension table into one large table with a lot of redundancy.
- If you use tools that require a snowflake schema in the background. (Fortunately, most modern tools support both schemas and even the galaxy schema.)

Consider using the star schema:

- In data marts. Data marts are subsets of data taken out of the central data warehouse. They are usually created for different departments and don't even contain all the history data. In this setting, saving storage space is not a priority.

On the other hand, the star schema does simplify analysis. This is not just about query efficiency but also about simplifying future actions for business users. They may understand databases and know how to write queries, but why complicate things and include more joins if we can avoid it? A business user could have a template query that joins the fact table with all the dimension tables. Then they only need to add the appropriate selections and groupings. (This approach is close to Excel's pivot tables.)

- If you use tools that require a star schema in the background. (Again, this usually isn't an issue.)

Both the star schema and the snowflake schema are relational models used to organize data warehouses and/or data marts. No matter how similar they are, they demonstrate two different approaches and have their own benefits and disadvantages. Personally, I would go with the snowflake schema when implementing a data warehouse (to save storage space) and with the star schema for data marts (to make life easier for business users).