Data Warehousing with Star and Snowflake schemas

Why do we use these schemas, and how do they differ?

Star schemas are optimized for reads and are widely used for designing data marts, whereas snowflake schemas are optimized for writes and are widely used for transactional data warehousing. A star schema is a special case of a snowflake schema in which all hierarchical dimensions have been denormalized, or flattened.

Attribute	**Star schema**	**Snowflake schema**
Read speed	Fast	Moderate
Write speed	Moderate	Fast
Storage space	Moderate to high	Low to moderate
Data integrity risk	Low to moderate	Low
Query complexity	Simple to moderate	Moderate to complex
Schema complexity	Simple to moderate	Moderate to complex
Dimension hierarchies	Denormalized single tables	Normalized over multiple tables
Joins per dimension hierarchy	One	One per level
Ideal use	OLAP systems, Data Marts	OLTP systems

Table 1. A comparison of star and snowflake schema attributes.

Normalization reduces redundancy

Both star and snowflake schemas benefit from the application of normalization. “Normalization reduces redundancy” is an idiom that points to a key advantage leveraged by both schemas.

Normalizing a table means to create, for each dimension:

. A surrogate key to replace the natural key, that is, the unique values of the given column, and

. A lookup table to store the surrogate and natural key pairs.

Each surrogate key’s values are repeated exactly as many times within the normalized table as the natural key was before moving the natural key to its new lookup table. Thus, you did nothing to reduce the redundancy of the original table.

However, dimensions typically contain groups of items that appear frequently, such as a “city name” or “product category”. Since you only need one instance from each group to build your lookup table, your lookup table will have many fewer rows than your fact table. If there are child dimensions involved, then the lookup table may still have some redundancy in the child dimension columns. In other words, if you have a hierarchical dimension, such as “Country”, “State”, and “City”, you can repeat the process on each level to further reduce the redundancy.

Notice that further normalizing your hierarchical dimensions has no effect on the size or content of your fact table - star and snowflake schema data models share identical fact tables.

Normalization reduces data size

When you normalize a table, you typically reduce its data size, because in the process you likely replace expensive data types, such as strings, with much smaller integer types. But to preserve the information content, you also need to create a new lookup table that contains the original objects.

The question is, does this new table use less storage than the savings you just gained in the normalized table?

For small data, this question is probably not worth considering, but for big data, or just data that is growing rapidly, the answer is yes, it is inevitable. Indeed, your fact table will grow much more quickly than your dimension tables, so normalizing your fact table, at least to the minimum degree of a star schema is likely warranted. Now the question is about which is better – star or snowflake?

Comparing benefits: snowflake vs. star data warehouses

The snowflake, being completely normalized, offers the least redundancy and the smallest storage footprint. If the data ever changes, this minimal redundancy means the snowflaked data needs to be changed in fewer places than would be required for a star schema. In other words, writes are faster, and changes are easier to implement.

However, due to the additional joins required in querying the data, the snowflake design can have an adverse impact on read speeds. By denormalizing to a star schema, you can boost your query efficiency.

You can also choose a middle path in designing your data warehouse. You could opt for a partially normalized schema. You could deploy a snowflake schema as your basis and create views or even materialized views of denormalized data. You could for example simulate a star schema on top of a snowflake schema. At the cost of some additional complexity, you can select from the best of both worlds to craft an optimal solution to meet your requirements.

Practical differences

Most queries you apply to the dataset, regardless of your schema choice, go through the fact table. Your fact table serves as a portal to your dimension tables.

The main practical difference between star and snowflake schema from the perspective of an analyst has to do with querying the data. You need more joins for a snowflake schema to gain access to the deeper levels of the hierarchical dimensions, which can reduce query performance over a star schema. Thus, data analysts and data scientists tend to prefer the simpler star schema.

Snowflake schemas are generally good for designing data warehouses and in particular, transaction processing systems, while star schemas are better for serving data marts, or data warehouses that have simple fact-dimension relationships. For example, suppose you have point-of-sale records accumulating in an Online Transaction Processing System (OLTP) which are copied as a daily batch ETL process to one or more Online Analytics Processing (OLAP) systems where subsequent analysis of large volumes of historical data is carried out. The OLTP source might use a snowflake schema to optimize performance for frequent writes, while the OLAP system uses a star schema to optimize for frequent reads. The ETL pipeline that moves the data between systems includes a denormalization step which collapses each hierarchy of dimension tables into a unified parent dimension table.

Too much of a good thing?

There is always a tradeoff between storage and compute that should factor into your data warehouse design choices. For example, do your end-users or applications need to have precomputed, stored dimensions such as ‘day of week’, ‘month of year’, or ‘quarter’ of the year? Columns or tables which are rarely required are occupying otherwise usable disk space. It might be better to compute such dimensions within your SQL statements only when they are needed. For example, given a star schema with a date dimension table, you could apply the SQL ‘MONTH’ function as MONTH(dim_date.date_column) on demand instead of joining the precomputed month column from the MONTH table in a snowflake schema.

Scenario

Suppose you are handed a small sample of data from a very large dataset in the form of a table by your client who would like you to take a look at the data and consider potential schemas for a data warehouse based on the sample. Putting aside gathering specific requirements for the moment, you start by exploring the table and find that there are exactly two types of columns in the dataset - facts and dimensions. There are no foreign keys although there is an index. You think of this table as being a completely denormalized, or flattened dataset.

You also notice that amongst the dimensions are columns with relatively expensive data types in terms of storage size, such as strings for names of people and places.

At this stage you already know you could equally well apply either a star or snowflake schema to the dataset, thereby normalizing to the degree you wish. Whether you choose star or snowflake, the total data size of the central fact table will be dramatically reduced. This is because instead of using dimensions directly in the main fact table, you use surrogate keys, which are typically integers; and you move the natural dimensions to their own tables or hierarchy of tables which are referenced by the surrogate keys. Even a 32-bit integer is small compared to say a 10-character string (8 X 10 = 80 bits).

Now it’s a matter of gathering requirements and finding some optimal normalization scheme for your schema.