Intro to Dimensional Modelling

# Intro to Data Warehouse

## What is a Data Warehouse?

A data warehouse is a very different type of database from those we have seen so far. The purpose of a warehouse is to support data analytics, often via reporting tools such as SQL Server Reporting Services (SSRS). A data warehouse will usually need to support fewer concurrent users than an OLTP, but the queries served are usually much more complex and resource intensive. Rather than looking for a few records at a time, queries against a data warehouse will often access and return large sets of data.

Data warehouses receive their data from other systems, so it is not the source of record for the data it stores. Any changes to the data should be made at the source system, not in the warehouse. This means that the warehouse can be optimized for read operations without worrying too much about write operations.

Data warehouses are very large, combining data from multiple source systems then storing current and historical data for all of them.

## Data Marts

A data warehouse stores data from multiple domains, or subject areas. It is sometimes referred to as an Enterprise Data Warehouse (EDW), storing all analytical data from across the business.

In this class we will be focusing on building a simple data mart. For our purposes, we can think of a data mart as a smaller subset of a data warehouse that is focused on one specific subject (e.g. HR or sales).

We will be focusing on the bottom-up approach to data warehousing called **dimensional modelling**. In this approach, we start by building data marts to suit specific functions, then integrate the data marts as more are created to form the data warehouse.

## OLTP vs DW

We are already familiar with some of the properties of an OLTP from our lessons on Entity-Relationship (ER) design. Let’s compare and contrast typical properties of an OLTP with a data warehouse:

|  |  |  |
| --- | --- | --- |
| Property | OLTP | Data Warehouse |
| Source of Data | End users, through an application. | Other systems |
| Primary Data Source | Yes | No |
| Backup/Recovery | Extremely important. | Usually not important. |
| Number of Users | Very high | Lower |
| User Access | Indirect, through application | Direct |
| Size of Queries | Fast, simple, few rows. | Slow, large, complex. |
| Types of Queries | Read and write | Read only |
| Historical Data | Little, if any | Yes |

## Design Considerations

The unique properties of a data warehouse impact the design choices we make.

### Optimized for Reads

End-users do not have write access. As we’ll learn later, only the systems we design to load and maintain our warehouse will typically have write access. The queries performed by end-users are large, complex read-only operations. A good data warehouse design will optimize for reads, not writes.

### Low Priority on Data Integrity

The data warehouse draws its data from other systems, so it is not the primary source of the data it uses. In addition, end users do not have write access, so there is little concern that end users will make a change that will harm data integrity.

### Optimized for Large Queries

Queries will usually access large sets of data, both in terms of the number of rows and the number of columns. These queries look at large sets in aggregate, rather than the specifics of individuals rows. A good data warehouse design will prioritize patterns over details.

### Easy to Understand

End users usually have direct access to the database. This puts more emphasis on creating a structure that is easy to understand and use. It also makes it important to provide information about the data, which is known as metadata.

### Tracks History

Time is one of the most important elements in a data warehouse. Users often want to track trends over time. A good data warehouse design must track the changes to data over time and make that information easy to access.

# Dimensional Model

## Star Schema Basics

To address the priorities of our data mart, as identified above, we will be building a **star-schema dimensional model**.

The model is comprised of **facts** and **dimensions**. In broad terms, facts are the information we want to know about, and dimensions are the context in which that information exists. We will learn more about these tables in upcoming lessons.

This model gets its name from the general shape of the schema when it is diagrammed. Each fact table references many dimension tables. Dimension tables do not reference other tables. As a result, when we fully map out all the related tables for a particular fact, our diagram might look something like this.



By contrast, in a **snowflake-schema dimensional model**, dimensions do reference other dimensions, resulting in a snowflake-shaped diagram that may look similar to this.



## Star Schema Characteristics

Let’s have a look at some of the characteristics of a star schema model. As we do, we’ll contrast against the ER model we’re already familiar with and see how this model serves the unique priorities of a data warehouse.

### Fewer, Wider Tables

A star schema will have far fewer tables than a comparable ER design, and those tables will have many more columns. Normalization protects data integrity and reduces write contention, creating many small tables in the process. Since data integrity and end user writes are not a priority in the data mart, we can create a much less normalized design.

The result is a schema that is easier for end-users to understand and use. It also improves performance by keeping related data together, reducing the number of joins required.

### Simpler Relationships

A star schema has very simple relationships – facts reference dimensions. Dimensions do not reference facts, and facts will not reference other facts. Unlike a snowflake schema, in a star schema, dimensions do not even reference other dimensions. This simple relationship structure is enabled by the highly de-normalized table structure of a dimensional model.

The simple relationships make the structure easier to understand, easier to query, and improves performance by reducing the number of joins required.

### Redundant Columns

In an ER design, normalization seeks to reduce or eliminate all redundant copies of our data. This helps reduce the data integrity risk posed by potentially having two conflicting copies of the data. A data mart is not the primary source of data, so if two conflicting copies of the data exist, the source system can often be referred to in order to correct the issue.

Keeping redundant copies of the data helps improve performance by reducing the number of joins required to satisfy a query. It also makes the model easier for end users by keeping more complete details in each table, rather than normalizing those details out into another table.

# Homework

## Understand

No homework