Data Warehousing Guide - Defining Dimension and Fact Tables - 3/4

qimia.io/en/blog/Data-Warehousing-Guide-Defining-Dimensions-and-Fact-Tables

Introduction

Welcome back to our series on Data Warehousing**.** In the last part Following the Data Warehousing Process, we defined the Fact Tables and the Dimension Tables that we are going to work on. (Please go to the previous article before you read this.) We also filled the first row of our fictional Bus Matrix.

This time, we will build the DDL statements for our Star Schema, taking into consideration that data might change over time. We will start by defining the Dimension Tables and then go on to define the Fact Tables.

1. Slowly Changing Dimensions

Although dimensions are typically more stable and static over time than facts, they are also changing which is why we need to think about how to handle those changes. We will talk about the three most common approaches to handle changes in dimensions. Be aware that multiple approaches can also be used in conjunction with one another.

It is also possible to use multiple types separately on a table since the type is usually determined on a column level. Nevertheless, columns from one dimension naturally tend to fall into the same type.

1.1. Type 0

Type 0 means that we keep the original value. This is meant for attributes that we explicitly do not want to change. For a product, this could be something like the date when it was first sold ('original_date').

For our case, this holds true for the values of the date dimension. We do not expect any changes of that.

1.2. Type 1

Type 1 means that we keep the new value and simply discard the old value. This is often wrongfully used as the default configuration when building data warehouses. It is very simple to implement, but throwing away the original values means losing information. Instead, we should strive to use Type 2 as the default approach.

1.3. Type 2

Type 2 means that we keep the old value and insert a new entry with the new value into the dimension. Using Type 2 necessitates the implementation of surrogate keys since the key that we used in the OLTP database will be duplicated when we insert the new entry.

As a result, the key from our OLTP database will usually be the durable key. When we implement a Type 2 approach for our dimension, we should always add at least three columns:

- created at
- valid until
- valid (flag)

There could even be more additional columns like "reason of update" but the above-mentioned ones are most important. Using these columns will facilitate the identification of the appropriate surrogate key to be inserted into the Fact Table when we write our ETL pipeline.

2. Table Definition Statements

2.1. Dimension Tables

We have four major dimensions:

1. date
2. employee
3. store
4. product

The date dimension is very simple. We only define the columns as described in our star schema. We do not have to worry about any changes, since the dates and all the related columns we chose are static and will not change over time.

The date dimension can theoretically be precomputed for the next X years. Even computing for the next 100 years will only result in about 36.500 rows, which is still quite negligible. Computing the entries of the date dimension based on the date entries in the other tables is an option, but it has to be integrated into the daily running pipeline and can blow up quickly if you use the date dimension across multiple businesses processes.

```
CREATE TABLE IF NOT EXISTS qimia_olap.dateDim (
    dateKey INT PRIMARY KEY,
    date DATE,
    dayOfWeekName VARCHAR (10),
    dayOfWeek INT,
    dayOfMonth INT,
    dayOfYear INT,
    calendarWeek INT,
    calendarMonthName VARCHAR (10),
    calendarMonth INT,
    calendarYear INT,
    lastDayInMonth VARCHAR (20)
);
```

The employee dimension is the first dimension where changes are possible. If an employee marries, the last name might change. If an employee moves to a different city, the address information will change. To facilitate keeping the history of the employee, we use a surrogate key in conjunction with the ID that we get from the source system. In this case, we will assume that the employee ID that we get from the source system does not change and can be used as a durable key. However, the employee dimension can be an exception in that regard. Former employees might be reemployed and get a new ID, meaning that you will need to find another solution if you want to analyze the employee over multiple periods of employment at the same time.

```
CREATE TABLE IF NOT EXISTS qimia_olap.employeeDim (
    employeeKey BIGINT PRIMARY KEY,
    employeeID INT,
    employeeFirstName VARCHAR (255),
    employeeLastName VARCHAR (255),
    employeeSalary INT,
    employeeZipCode INT,
    employeeCity VARCHAR (255),
    employeeCountry VARCHAR (50),
    employeeEmployedSince INT,
    employeeManagerLastName VARCHAR (22),
    employeeCreatedAt INT DEFAULT to_char(CURRENT_DATE, 'YYYYMMDD')::integer,
    employeeValidUntil INT DEFAULT 29991231,
    employeeValid VARCHAR (20) DEFAULT 'Valid',
    FOREIGN KEY (employeeEmployedSince) REFERENCES qimia_olap.dateDim(dateKey),
    FOREIGN KEY (employeeCreatedAt) REFERENCES qimia_olap.dateDim(dateKey),
    FOREIGN KEY (employeeValidUntil) REFERENCES qimia_olap.dateDim(dateKey)
);
```

Similar to the employee dimension, the store dimension will also capture history by using a surrogate key. Furthermore, we joined the currencies to the store dimension. Since the currency of payment is in our case determined by the country of the store, it does not make sense to keep the table separated.

```
CREATE TABLE IF NOT EXISTS qimia_olap.storeDim (
    storeKey BIGINT PRIMARY KEY,
    storeID INT,
    storeName VARCHAR (255),
    storeZipCode INT,
    storeCity VARCHAR (255),
    storeCountry VARCHAR (50),
    sellingSquareFootage INT,
    totalSquareFootage INT,
    storeOpenedSince INT,
    storeLastRedesigned INT,
    storeCurrency VARCHAR(255),
    storeCurrencyExchangeRate float,
    storeCreatedAt INT DEFAULT to_char(CURRENT_DATE, 'YYYYMMDD')::integer,
    storeValidUntil INT DEFAULT 29991231,
    storeValid VARCHAR (20) DEFAULT 'Valid',
    FOREIGN KEY (storeCreatedAt) REFERENCES qimia_olap.dateDim(dateKey),
    FOREIGN KEY (storeValidUntil) REFERENCES qimia_olap.dateDim(dateKey),
    FOREIGN KEY (storeOpenedSince) REFERENCES qimia_olap.dateDim(dateKey),
    FOREIGN KEY (storeLastRedesigned) REFERENCES qimia_olap.dateDim(dateKey)
);
```

The product dimension also contains a surrogate key to keep history. Similar to the store dimension, we decided to join the supplier table to the product table. For us, this was an easy decision since our products only have one supplier in the source schema. If that was not the case, joining these two tables would build the cartesian product, which might seem scary at first. However, this is in most cases still the right choice, since the Fact Table is anyways way larger than our dimensions.

```
CREATE TABLE IF NOT EXISTS qimia_olap.productDim (
    productKey BIGINT PRIMARY KEY,
    productID VARCHAR (50),
    productBrand VARCHAR (255),
    productType VARCHAR (255),
    productCategory VARCHAR (255),
    productFit VARCHAR (255),
    productSize VARCHAR (255),
    productSex VARCHAR (10),
    productDescription VARCHAR (255),
    productPurchasePrice Float,
    productSellingPrice Float,
    productPackageSize Int,
    supplierName VARCHAR (22),
    supplierPreferred BOOLEAN,
    supplierZipCode INT,
    supplierCity VARCHAR (30),
    supplierCountry VARCHAR (30),
    productCreatedAt INT DEFAULT to_char(CURRENT_DATE, 'YYYYMMDD')::integer,
    productValidUntil INT DEFAULT 29991231,
    productValid VARCHAR(20) DEFAULT 'Valid',
    FOREIGN KEY (productCreatedAt) REFERENCES qimia_olap.dateDim(dateKey),
    FOREIGN KEY (productValidUntil) REFERENCES qimia_olap.dateDim(dateKey)
);
```

The mindful reader has recognized that each of the changing dimension tables (employeeDim, storeDim, productDim) contain FOREIGN KEY definitions referencing the dateDim. Therefore, the dateDim forms a so-called Role Playing Dimension, meaning it is referenced in different contexts. It is not only referenced by the fact table, where it provides further information to the day of that specific business process, but it is also used for providing information on dimension attributes such as the hiring date of the employee. Therefore, the dateDime plays several roles. However, it is best practice to include a view for each role a Role Playing Dimension plays. We will explain two of them exemplarily in the next section.

2.3. Views

The views in the data warehouse are not a necessary element such as the fact or dimension tables. However, when a Role Playing Dimension is introduced, one should create views to clarify the context in which the dimension is referenced. In our schema we introduced eleven views:

- for referencing facts:
    - v_salesFactDate
    - v_salesDailyFactDate
- for referencing store attributes:
    - v_openedSinceStoreDimDate
    - v_lastRedesignedStoreDimDate
    - v_validUntilStoreDimDate
    - v_createdAtStoreDimDate
- for referencing product attributes:
    - v_validUntilProductDimDate
    - v_createdAtProductDimDate
- for referencing employee attributes:
    - v_employedSinceEmployeeDimDate
    - v_validUntilEmployeeDimDate
    - v_createdAtEmployeeDimDate

However, they are all defined in the same way, by just selecting all columns from the dateDim and rename the columns according to their context.

```
CREATE OR REPLACE VIEW qimia_olap.v_openedSinceStoreDimDate
AS
SELECT dateKey           as openedSinceStoreDim_dateKey
     , date              as openedSinceStoreDim_date
     , dayOfWeekName     as openedSinceStoreDim_dayOfWeekName
     , dayOfWeek         as openedSinceStoreDim_dayOfWeek
     , dayOfMonth        as openedSinceStoreDim_dayOfMonth
     , dayOfYear         as openedSinceStoreDim_dayOfYear
     , calendarWeek      as openedSinceStoreDim_calendarWeek
     , calendarMonthName as openedSinceStoreDim_calendarMonthName
     , calendarMonth     as openedSinceStoreDim_calendarMonth
     , calendarYear      as openedSinceStoreDim_calendarYear
     , lastDayInMonth    as openedSinceStoreDim_lastDayInMonth
from qimia_olap.dateDim;
```

2.4. Fact Tables

Our Fact Table consists of the foreign keys to the previously mentioned dimensions as well as the metrics that we gather. We are joining the product-related information to the facts.

```
CREATE TABLE IF NOT EXISTS qimia_olap.salesFact (
       salesID INT,
       productKey INT,
       dateKey INT,
       employeeKey INT,
       storeKey INT,
       supplierKey INT,
       quantity INT,
       regularUnitPrice Float,
       discountUnitPrice Float,
       totalRegularPrice Float,
       totalDiscountPrice Float,
       totalRevenue Float,
       revenueMargin Float,
       PRIMARY KEY (salesID,productKey),
       FOREIGN KEY (productKey) REFERENCES qimia_olap.productDim(productKey),
       FOREIGN KEY (dateKey) REFERENCES qimia_olap.dateDim(dateKey),
       FOREIGN KEY (employeeKey) REFERENCES qimia_olap.employeeDim(employeeKey),
       FOREIGN KEY (storeKey) REFERENCES qimia_olap.storeDim(storeKey),
       FOREIGN KEY (supplierKey) REFERENCES qimia_olap.supplierDim(supplierKey)
   );
```

Having meaningful metrics is obviously key to building an effective Data Warehouse, and revenue is certainly one of them. In the real world, the purchasing price will not simply be written inside of the product table and instead be determined by a quite complex procurement process. To facilitate analysis along the value chain, such as computing the revenue as the difference between purchasing and selling price, using the product dimension as a conformed dimension is an absolute necessity! It will enable you to combine information from different fact tables by joining via the conformed dimension. Building different solutions for different departments might seem convenient at first since

there will be way less organizational friction. But if you want to get the real benefit from building the data warehouse, there is no way around having discussions and finding a compromise to integrate the processes with one another.

We also decided to build a second Fact Table, which aggregates the sales on a daily basis per store and product. There is little difference when it comes to the schema, other than the missing employee dimension and the 'transactions' attribute describing the number of sales that the product was involved in. The daily aggregated facts can speed up queries that do not require the atomic granularity of the primary Fact Table.

```
CREATE TABLE IF NOT EXISTS qimia_olap.salesDailyFact (
    dateKey INT,
    storeKey BIGINT,
    productKey BIGINT,
    transactions INT,
    quantity INT,
    regularUnitPrice Float,
    discountUnitPrice Float,
    purchaseUnitPrice Float,
    totalRegularPrice INT,
    totalDiscountPrice INT,
    totalPurchasePrice INT,
    totalRevenue INT,
    revenueMargin Float,
    PRIMARY KEY (dateKey,storeKey,productKey),
    FOREIGN KEY (productKey) REFERENCES qimia_olap.productDim(productKey),
    FOREIGN KEY (dateKey) REFERENCES qimia_olap.dateDim(dateKey),
    FOREIGN KEY (storeKey) REFERENCES qimia_olap.storeDim(storeKey)
);
```

Wrapping It Up

This time, we had a look at the definition of our Fact- and Dimension Tables, with a special focus on the slowly changing dimensions. We have now set up the source and target definition for the transfer of our data. In the next article, we will be occupied with the construction of the data pipelines that transfer the data into our Data Warehouse. See you in part 4, Data Warehousing Guide - The ETL Processing!

For the next part of the tutorial click here