Main Steps of Data Warehouse Design. Sales Case

In the previous handout we discussed properties of a data warehouse and how they influence approaches to design of this special database. In this handout we continue working with `Sales` case and will concentrate on the main decisions that define the design of a data warehouse.

2.1. Design Steps

It is recommended to approach design of a data warehouse in the following way:

- *Choose the business process*. A data warehouse is built to help users analyze performance of a particular business activity or, we say, business process. In this handout we discuss a data warehouse for the sales activity of a chain of stores.
- *Define the grain*. Decide how detailed data about the business has to be to satisfy analytical needs of users. The level of details of business data is defined as the *grain* of a data warehouse. For example, we may consider the grain of *daily sales of products in stores* or the grain of *monthly sales of products in sales districts*.
- *Build dimensions*. Build dimensions with the help of which users will be able to analyze business data. If in `Sales` case users need to analyze *what* is sold, then we need to consider `Product` dimension and include in this dimension attributes used for analyses, e.g. subcategory and category. If users also want to know about *where* products were sold, then we need to add dimension `Store` with required attributes. There is a strong connection between the grain and dimensions.
- *Construct facts*. It is possible to analyze business performance if there is a way to measure it. The final step of a data warehouse design is about choosing the appropriate measures of the business, which will become facts in `Fact` table.

2.1.1. Business Process

A data warehouse is built for a definite business process. Previously, we discussed a data warehouse for the sales activity of the chain of stores. We started with a traditional OLTP database for `Sales` application and then transformed its design into a dimensional model.

Why a data warehouse has to reflect a business process? The purpose of a data warehouse is to measure and analyze performance of the business. Business processes are started with some input and produce an output when they are finished. By measuring and then analyzing the input and output of the process users can judge about the effectiveness of this particular activity and see what influences it. Usually, a business process is related to several departments or other organizational units of a company, and recording data about a department's activities would not give the complete picture of the business process. Also, often departments have various business activities, each of which is measured differently, and it is difficult, if not impossible, to measure performance of a department. For example, departments of a company can be involved in such activities as purchasing of parts and supplies, manufacturing products, and selling them; while it is difficult to measure the performance of a department, it is easy to separately measure performance of either of the activities for a department or the whole company.

OLTP databases or other primary data sources are usually built for departments or other organizational units of a company and contain data about multiple business processes, as did our initial OLTP database. We mentioned that `Sales` database contained data about ordering and purchasing of products, but for simplicity, we discussed only the part about sales. Often, data about a whole business process is distributed across several databases or other data sources. It is important to understand what data from the sources is relevant to analyses of the business process and consider using it for the data warehouse.

In our case, we are going to build a data warehouse for the sales business process.

2.1.2. Grain

The grain of a data warehouse defines what details about the business process will be available to users. For example, if we keep track of sale of each product to every customer, we will have the most detailed picture of the business, and the grain will be defined as "each sale of products in stores".

In many cases, for making conclusions about business performance users do not need to see all details. Maybe, in this case we could consider the grain "yearly sales of products in stores", which, we say, is larger than the grain "each sale of products in stores".

While we can get more details from a data warehouse with a smaller grain, a larger grain hides some details. For example, grain "monthly sales of products in stores" will not allow users to see sales for a particular day of the month, e.g. 01/01/04; but such request can be handled by a data warehouse with a smaller grain "daily sales of products in stores" (Figure 2-1):

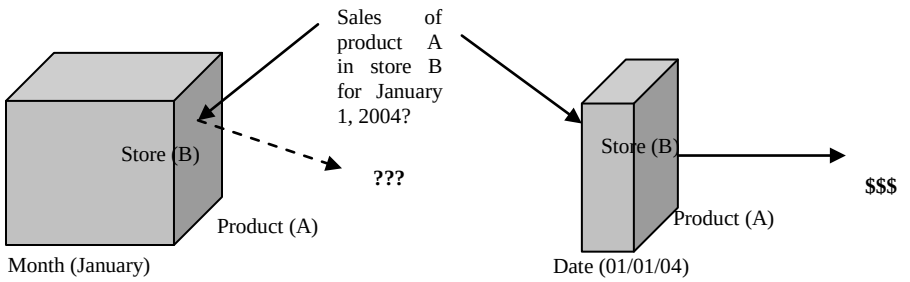

Figure 2-1. Restrictiveness of the grain

We may view a grain as a building block or a brick of a data warehouse; the larger is the grain-brick, the fewer bricks we need to build a warehouse. In database terms it means that we have fewer records and a smaller database.

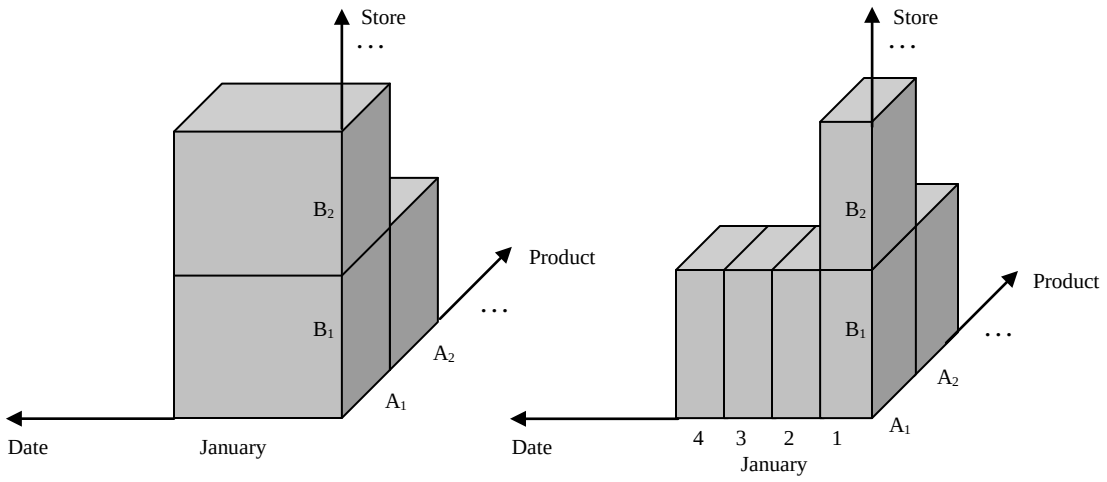

Figure 2-2. Building data warehouses with different grains

Figure 2-2 shows the process of building two different data warehouses for `Sales` case: one with the grain of monthly sales of products in stores, and another--of daily sales of products in stores. The process of populating the corresponding databases with records is schematically shown below:

January	Store B_1	Product A_1	$$$
January	Store B_2	Product A_1	$$$$
January	Store B_1	Product A_2	$$
...			

01/01/04	Store B_1	Product A_1	$
01/02/04	Store B_1	Product A_1	$$
01/03/04	Store B_1	Product A_1	$$
01/04/04	Store B_1	Product A_1	$
01/01/04	Store B_1	Product A_2	$$
01/01/04	Store B_2	Product A_1	$$
...			

A smaller grain does not mean that it is better--if users are not interested in details, then it does not make sense to keep these details exposed to them. A user will never use detailed data directly, but will need to gather and process it in order to get the requested result (see Figure 2-3). For example, if in our case users need only monthly analysis of sales, then with the daily grain of the data warehouse we will loose in both, size of the database and performance.

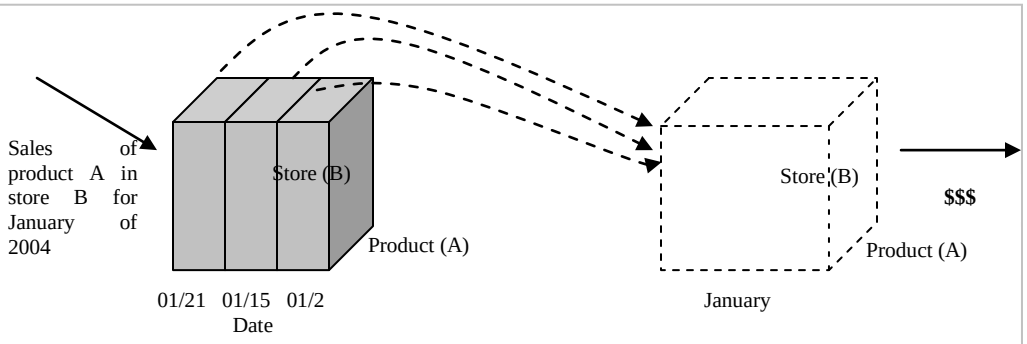

Figure 2-3. Processing of data details

When choosing the grain, we must remember two main rules:

1. *The level of details provided by the grain must satisfy all requests of users.*

2. *The chosen grain must be the largest possible grain for these requests.*

To make a conclusion about the grain of our `Sales` data warehouse, we must know how users need to analyze sales data. Let us discuss two scenarios of users trying to specify their analytical requirements.

In the first scenario users say that they need to see:

- Monthly totals of sales of different products in stores of various sales regions.
- How for products of particular categories sales on weekdays differed from sales on weekends.
- Total quarterly sales in different stores.

In the second scenario users ask about:

- Monthly sales of products of different categories in stores.
- Yearly total sales in stores.

For the first scenario, from the first request we could decide on the grain “monthly sales of products in regions”, but this monthly grain will not satisfy the second request about sales on different days. The third request makes us reconsider the initial grain as well, because it addresses sales in particular stores, while we decided to keep data about sales in regions. Taking into account *all* requests of the first scenario, we will need the grain “daily sales of products in stores.”

For the second scenario, the first request may lead us to the grain “monthly sales of categories of products in stores”, which will suit the second request as well. This grain is larger than the grain for the first scenario--it conceals details of sales of separate products on different days. It cannot be used for requests of the first scenario, but it is good for requests of the second scenario.

2.1.3. Dimensions

The grain of a data warehouse defines all or some of dimensions across which users plan to analyze business performance. In some cases, dimensions are explicitly mentioned in the definition of the grain, e.g. for the first scenario of our case the grain "daily sales of products in stores" explicitly addresses details of sales of products in stores, and it means that we need dimensions `Product` and `Store`.

It is important to understand that the grain imposes requirements on inclusion of particular dimensions in the model of a data warehouse. For example, to maintain the grain "daily sales of products in stores" we need dimensions `Product` and `Store`. If we choose only one dimension, e.g. `Product`, then we get less detailed data about the business performance. Two tables below schematically show this: the table on the left reflects the business measured across two dimensions `Product` and `Store`, while the table on the right stands for the business measured across only one dimension `Product`:

Date	Store	Product	Quantity
01/01/04	Store B_1	Product A_1	10
01/01/04	Store B_2	Product A_1	5
01/02/04	Store B_1	Product A_1	12
01/02/04	Store B_2	Product A_1	24
...			

Date	Product	Quantity
01/01/04	Product A_1	15
01/02/04	Product A_1	36
...		

Both grains that we defined for the above two scenarios need a dimension for analysis of *what* was sold, but the level of details this dimension should provide is different. In the first case, we need information about each product, while in the second--about categories of products (the details of products are hidden). Examples of such dimensions are shown in Figure 2-4.

Not only the grain of a data warehouse defines *what* dimensions must be included in the model, but also the *grains* (level of details) of dimensions. While both mentioned scenarios include `Product` dimension, in the first case its grain is "product," and in the second--"category of product."

Grain could be seen as a multidimensional cube (see Figure 2-5), and we measure the business performance on intersections of cube's dimensions, e.g. "sales of product A in store B" for dimensions `Product` and `Store`, or "sales of product A" for one dimension `Product`.

SKU prodName subCatCode subCatName catCode catName

SKU	prodName	subCatCode	subCat Name	catCode	catName
1	Coca-Cola	001	Soda	111	Soft Drink
2	Pepsi	001	Soda	111	Soft Drink
3	Tropicana Juice	002	Juice	111	Soft Drink
. . .					

a)

catCode catCode catName

catCode	catName
111	Soft Drink
112	Candy
. . .	

b)

Figure 2-4. Dimension `Product` with grain: `a)` product; b) category.

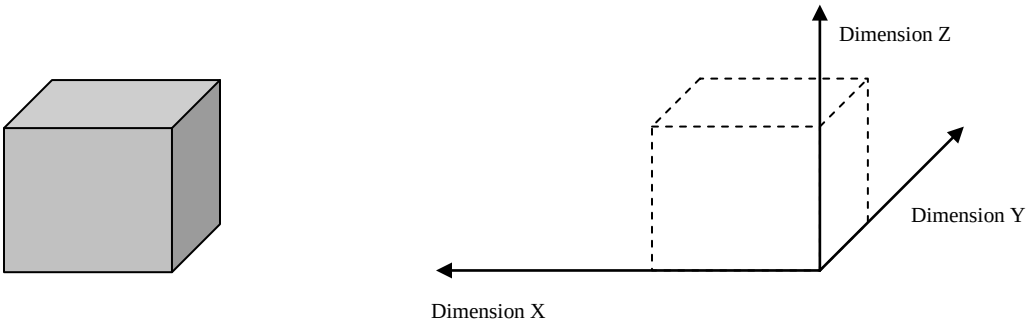

Figure 2-5. Grain and dimensions

Later we will see that the grain not always explicitly defines dimensions and that not all dimensions of a data warehouse are defined by the grain.

2.1.4. Facts

If dimensions allow users to see business data from different angles, or we say, analyze data across different dimensions, `Fact` table contains business data itself. What should be included in business data (what should be facts) is also defined by users' needs and goals of analysis.

The measures (facts) must satisfy several requirements:

- *Be relevant to the business process*. In our `Sales` case we used as facts quantity of sold product and actual sale price; we could also use some other measures, e.g. original sale price. However, inventory levels of products in stores or manufacturer's cost are not relevant to sales process.
- *Correspond to the chosen grain of a data warehouse*. For the grain "daily sales of products in stores," we will use the fact that shows daily total quantity sold for each sold product in every store. However, neither the fact with monthly total quantity of sold product in a store, nor the fact with quantity of sold product for each sale agrees with this grain. The former of these two facts is good for a larger grain "monthly sales of products in stores," while the latter corresponds to a smaller grain "each sale of products in stores."
- *Be accurate enough to satisfy users' needs*. Such measurement of quantity of sold product as "a few" or "many" will not allow users to understand the business well. Not only will these measurements be not accurate enough, but also they will not allow to perform aggregate requests (e.g. "What was quantity of sold product for the previous year?").
- *Be numeric* to allow finding totals or averages, which are included in typical requests of OLAP applications.

Facts are related to dimensions--they contain as detailed data about the business as is defined by the grain. If we have the grain "daily sales of products in stores", then each fact is about total sales of a particular product in some store at a given date.

In our case, users suggest that a good measurement of sales is quantity of sold product. They also need to know about the dollar amount of each sale, as well as whether the actual sale price was lower than the original price (due to various promotions). OLTP database for `Sales` application contains the attributes we need. We may take them as they are in the OLTP database: `quantity, price,` and `salePrice`. We also could consider some other attributes, e.g. manufacturer's price from another part of OLTP database, which we didn't include in the discussion.

2.2. Moving to a Better Database

The "dimensional" or "star" architecture of an OLAP database serves analytical needs of users: it is simple, most of requests have simple and regular structure, and it promises a better performance.

2.2.1. Advantages of Dimensional Model - Once Again

Dimensional model provides easy and direct access to data about the business performance-- a user specifies analytical needs on a dimension, and because `Fact` table references each dimension, the user gets from dimensional rows to corresponding rows of `Fact` table. For example, Figure 2-6 shows that if a user wants to know about sales of drinks, dimension `Product` enables specifying this request (`Product.catName = 'Drink'`) and connects the user to needed facts about the business. The "flattened" non-normalized structure of a dimension makes it easy for users to formulate their request by

selecting one or more values of the dimension's attribute, e.g. `catName = 'Drink' OR catName = 'Diary'`.

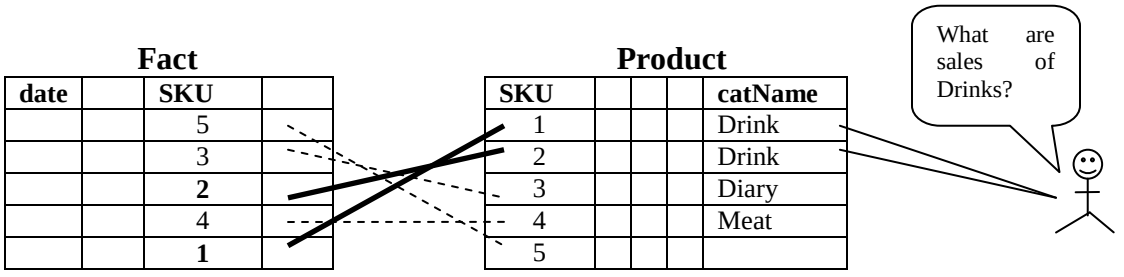

Figure 2-6. Ease of using a dimensional database.

Dimensions allow easy implementation of new analytical needs. If, for example, users decide that they want to analyze sales for diet and not diet products, we can add a corresponding attribute to dimension `Product`, and this will be the only change to the data warehouse. The updated dimension `Product` provides the required possibility (see Figure 2-7).

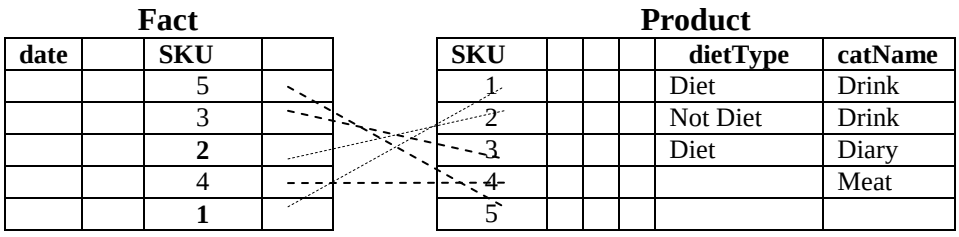

Figure 2-7. Ease of changing a dimensional database.

2.2.2. `Date` Dimension

Data warehouses are "historical" databases—they record business performance over time, and users are often interested in the "historical" view of the business, i.e. performance for particular periods or moments of time. Examples of such requests could be:

- What were sales of drinks in the previous year?
- What were sales during the Christmas season of the previous year?
- How this year sales on weekends differ from sales on weekdays?

On the dimensional database from handout 1, the first of the requests could be implemented as:

```
SELECT SUM(quantity)
FROM Sales s, Product p
WHERE <join conditions> AND
```

```
p.catName = 'Drink' AND
year function(s.date) = '2003';
```

Please recall that we agreed to use short and general notation `<join conditions>` for showing join conditions between `Fact` table and the dimensions involved in the query. In this query, we applied special date function that returns the year of the date field. Such function is included in most of the commercial DBMS.

There are several problems with this request. First of all, access to business data is more complicated than, for example, when we wanted to analyze sales of drinks and were able to simply specify this on the dimension `Product (Product.catName = 'Drink')`. Users have to know about the functions that return different features of date; in this case, it is the year. Second, for some of requests commercial DBMS do not provide the corresponding function, e.g. for the Christmas season, not to mention that different business define this season differently. And one more thing--those who are familiar with performance issues of databases would know that applying functions to attributes by which data is searched could have a serious impact on performance.

We saw how dimensions simplify access to data. The solution to all mentioned problems is to create `Date` or `Time` dimension, on which users could specify their historical requests and which could be easily adjusted to a particular business. The grain of this dimension (the same as of other dimensions) is defined by the grain of a data warehouse. All possible features of dates that can be used in analytical requests have to be included in this dimension as attributes, e.g. month, year, season, weekend, weekend, etc. Figure 2-8 shows `Date` dimension with daily grain.

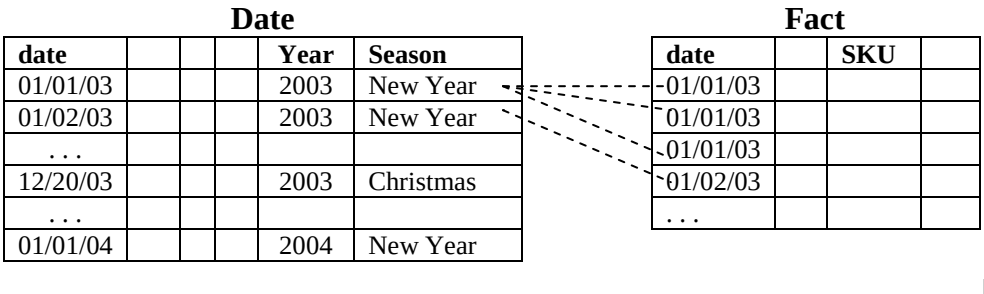

Figure 2-8. `Date` dimension.

2.2.3. Surrogate Dimensional Keys

In the dimensional database, `Fact` table references all dimensions and therefore includes primary keys of all dimensions as foreign keys. In `Fact` table of `Sales` case these are: `date` (`Date` dimension), `SKU` (`Product` dimension), and `storeNumb` (`Store` dimension).

`Fact` table is the largest table of a data warehouse, and it is important to design it in the best possible way to minimize its size and ensure good performance of read queries. Assuming that we have the appropriate grain (and can't minimize the number of rows by moving to a larger grain), we may try to minimize the length of each row. The few facts we have: quantity of sold product, original sale price, and actual sale price are important to users, and we can't eliminate either of them. Nor can we eliminate the dimensional keys. But we may consider minimizing the size of dimensional keys, for example, instead of `date` type for date fields (which is usually 8 bytes long) and alphanumeric type for SKU (which can be pretty long), we may choose surrogate keys represented by integers, which are much shorter (often 4 bytes long). On the millions rows of `Fact` this will give a considerable gain. The final design of our data warehouse with `Date` dimension and surrogate dimensional keys is shown in Figure 2-9.

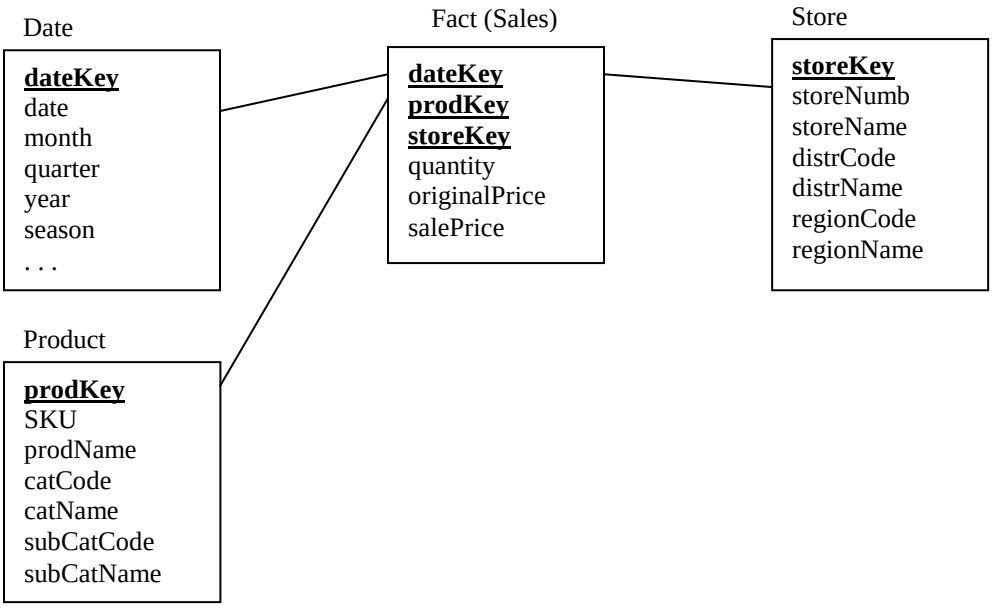

Figure 2-9. Modified dimensional model of `Sales` case.

Pursuing the goal of minimizing the size of `Fact` table, we additionally benefited from the surrogate dimensional keys. As you remember, data warehouse takes data from primary data sources. In case the same data is received from multiple data sources, it can have conflicting primary keys or in some sources primary keys can change over the time (the lifetime of data in a data warehouse is usually longer than in an OLTP database). By using the surrogate keys, we protect ourselves from having a conflict either within a data warehouse or while loading it with data. Sustainability is considered to be one of the main challenges in a successful utilization of a data warehouse, and making a data warehouse resilient to changes in the data sources is very important.

There is one more gain from the short dimensional keys--a better performance of queries with join of `Fact` table to dimensions, which are the most common queries on a data warehouse.

Usually, the surrogate keys are tightly administered and assigned within a data warehouse.

We often preserve the original primary keys in dimensions, because in some cases they bear meaningful data (`date`) or can help in maintaining a data warehouse.

The primary key of this `Fact` table is composed of all dimensional keys. It is not always the case.

2.3. Drilling a Data Warehouse

The process of getting data from a data warehouse is called drilling. Users can *drill down* and see the more detailed data, or they can *drill up* for the more general and less detailed view of the business.

Here is the scenario of drilling down:

1. Users may request total sales for the current year (in all stores):

```
SELECT SUM(quantity)
FROM Fact f, Date d
WHERE < join conditions> AND
     d.year = '2004';
```

2. After that users may request total sales for January of the current year only:

```
SELECT SUM(quantity)
FROM Fact f, Date d
WHERE < join conditions> AND
     d.month = '200401';
```

3. Then, users may decide to see sales of a particular category of products for the same period of time:

```
SELECT SUM(quantity)
FROM Fact f, Date d, Product p
WHERE < join conditions > AND
     d.month = '200401' AND
     p.catName = 'Drink';
```

4. Or, they could request details of sales for different products of this category:

```
SELECT SUM(quantity), p.prodName
FROM Fact f, Date d, Product p
WHERE < join conditions > AND
     d.month = '200401' AND
     p.catName = 'Drink'
GROUP by p.prodName;
```

5. They can continue to drill down and get details of sales of products of this category in a particular sales region:

```
SELECT SUM(quantity), p.prodName
FROM Fact f, Date d, Product p, Store s
WHERE < join conditions > AND
     d.month = '200401' AND
     p.catName = 'Drink' AND
     s.regionName = 'North-East'
GROUP by p.prodName;
```

Drilling down can be performed by:

- Constraining an attribute which represents a smaller grain of the dimension (as in query 2).
- Adding to a query a constraint on another dimension (queries 3 and 5).
- Requesting subtotals (query 4).

Drilling down it is limited by the grain of a data warehouse:

```
SELECT SUM(quantity)
FROM Fact f, Date d, Product p, Store s
WHERE <join conditions> AND
     d.date = '01/01/04' AND
     p.prodName = 'Coca-Cola' AND
     s.storeName = 'Best Buy'.
```

Executing requests of this scenario in reversed order will demonstrate drilling up.

2.4. Summary

The basic steps of a data warehouse design are: choosing the business process, deciding about the grain, defining dimensions, and building `Fact` table.

A data warehouse contains data about performance of a particular business process. It is important to understand what constitutes the business process and what data from data sources will be needed to implement it.

We can keep data about the chosen business process on different levels of details; our decision about how detailed data should be is defined by the grain. The smaller is the grain, the more detailed view of the business users can have. The larger grain limits the possibility to analyze details of business performance. The grain of a data warehouse is defined so that users could get data they need without being distracted by the details they are not interested in.

Needed details of business performance are supported by dimensions. The grain of a data warehouse defines some of dimensions and their grains. Data warehouses include `Date` as a separate dimension. When users request data for specific time periods or moments,

e.g. a particular year, Christmas time, or Super Bowl day, we have all benefits of access to business data through dimensions--simplicity, effectiveness, and flexibility.

Short integer field is added to each dimension to serve as a primary key. This allows to minimize the size of `Fact` table and to resolve possible conflicts between keys from different data sources.

The process of retrieving data from a data warehouse is called drilling. The users drill down to more detailed data, or they drill up for a larger view of the business.

Review Questions

1. Why does a data warehouse keep data about a business process?
2. How is the needed level of details of data about business performance defined?
3. What is the grain of a data warehouse?
4. Which grain is better: a larger or a smaller one?
5. How the grain and dimensions of a data warehouse are related?
6. What is the role of dimensions in a data warehouse?
7. How design of dimensions helps in achieving the main goals of a data warehouse?
8. Why we choose to have `Date` dimension?
9. What are the benefits of surrogate keys of dimensions?
10. What is the structure of `Fact` table?
11. How you decide on facts?
12. What are "good" facts?
13. How users drill a data warehouse?

Practical Assignments

1. Build a model of `Sales` data warehouse for the grain "monthly sales of categories of products in stores":
    a. Define dimensions and their attributes.
    b. Construct facts, which will measure the performance in terms of quantities of sold products and dollar amounts.
    c. Compare facts of this model to facts of model with the grain "daily sales of products in stores".
2. Build a model of `Sales` data warehouse for the grain "daily sales of products":
    a. Define dimensions and their attributes.
    b. Construct facts, which will measure the performance in terms of quantities of sold products and dollar amounts.
    c. Compare facts of this model to facts of model with the grain "daily sales of products in stores".
3. On the models of assignments 1 and 2 show different scenarios of drilling up and down.
4. Explain which of the following queries make sense on the grain "daily sales of products in stores":

```
a. SELECT SUM(salePrice)
   FROM Fact f, Date d
```

```
      WHERE <join conditions> AND
        d.month = '200401'.
   b. SELECT AVG(salePrice)
      FROM Fact f, Date d
      WHERE <join conditions> AND
        d.month = '200401'.
   c. SELECT AVG(salePrice)
      FROM Fact f, Date d, Product p
      WHERE <join conditions> AND
        d.month = '200401' AND
        p.name = 'Coca-Cola'.
   d. SELECT SUM(quantity * salePrice)
      FROM Fact f, Date d, Product p
      WHERE <join conditions> AND
        d.month = '200401'.
```

5. The chain of stores regularly promotes products with the help of coupons, discounts, ads, etc.; at any moment, a product in a store can be on one type of promotion only. Users of `Sales` data warehouse need to have additional information about whether sold products were on any kind of promotion and what promotion it was. Explain how you will redesign `Sales` data warehouse to provide users with possibility to analyze sales of promoted products.
6. There can be two scenarios of sales of promoted products. In the first, if a product is on a promotion in a store, then this promotion takes place for every sale of the product. In the second scenario, some promoted products can be sold at regular sale prices (e.g., customer may not have a coupon).
    a. Build dimensional model of `Sales` data warehouse for each scenario.
    b. For each dimensional model, explain whether the grain is different from the grain "daily sales of products in stores."
7. Explain which of the following requests is possible on the models of assignment 6 and if the request is possible, write for it SQL query:
    a. Total dollar amount of sold products of category `X` that were on promotion `Y` on date `Z`.
    b. List of products of category `X` that were on any promotion in stores of region `Y` on date `Z`.
    c. List of sold products of category `X` that were on any promotion in stores of region `Y` on date `Z`.
8. Build `Promotion` dimension for situation when a product can be on several different promotions at the same time.
9. Build a "family" data warehouse for analysis of spending by the members of your family.
    a. Define how you would like to analyze financial performance of your family.
    b. Suggest a dimensional model, which would satisfy your requirements.

References:

1. R. Kimball "The Data Warehouse Toolkit", John Wiley & Sons, Inc., 1998.

2. C. Imhoff, N. Galemmo, J. Geiger “Mastering Data Warehouse Design”, John Wiley & Sons, Inc., 2003.
3. C. Adamson, M. Venerable “Data Warehouse Design Solutions”, John Wiley & Sons, Inc., 1998.
4. W. Inmon “Building the Data Warehouse”, John Wiley & Sons, Inc., 2002.