AG Heterogene Informationssysteme

Prof . Dr.-Ing. Stefan Deßloch

Fachbereich Informatik

Technische Universität Kaiserslautern

Recent Developments for Data Models – Solution to Exercise 4

Monday, June 18, 2012 – 15:30 to 17:00 – Room 36-336

1) Multi-dimensional Modeling

Multi-dimensional schemas are typically used for data warehousing. The design goals are query performance, user understandability (through simplicity), and resilience to changes.

Consider the following scenario: An online bookstore company wants to build a data warehouse to better understand its sales. Each book has a unique identifier and a title. Books are further classified into genres. The price on which books are purchased by the company is known as *cost price*. The price on which books are sold to customers is known as *sales price*. The difference between cost price and sales price is known as *gross profit*. The *gross margin* can be calculated by dividing the gross profit by the sales price.

Customers need to register before they can place orders. During registration customers have to enter their name, email address, age, city, state, and country. Each customer is assigned a unique identifier (customer key).

Orders may contain multiple order lines. Each order line refers to one or more copies of the same book. Each order is assigned a unique identifier (order key), order lines are numbered consecutively. For analyzing the sales it is important to understand when an order was placed, i.e. at what year, quarter, month, day of month, week of year, day of week, and whether the day was a holiday.

a. Design a star schema that captures the information to be analyzed described in the scenario above!

A star schema is made up from fact tables and dimension tables. A fact table is the primary table in a dimensional model where the numerical performance measurements of the business are stored. A row in a fact table corresponds to a measurement. A measurement is a row in a fact table. The most useful facts are numeric and additive, such as dollar sales amount. We often describe facts as continuously valued mainly as a guide for the designer to help sort out what is a fact versus a dimension attribute. The dollar sales amount fact is continuously valued because it can take on virtually any value within a broad range.

Dimension tables are integral companions to a fact table. The dimension tables contain the textual descriptors of the business. Each dimension table is defined by its sin-

gle primary key, which serves as the basis for referential integrity with any given fact table to which it is joined.

Dimension attributes serve as the primary source of query constraints, groupings, and report labels. For example, when a user states that he or she wants to see dollar sales by week and by brand, week and brand must be available as dimension attributes.

Sometimes when we are designing a database it is unclear whether a numeric data field extracted from a production data source is a fact or dimension attribute. We often can make the decision by asking whether the field is a measurement that takes on lots of values and participates in calculations (making it a fact) or is a discretely valued description that is more or less constant and participates in constraints (making it a dimensional attribute).

(Ralph Kimball, Margy Ross: The Data Warehouse Toolkit, Wiley, 2002)

In the above scenario the numeric attributes are customer age, order line quantity, cost price, sales price, gross profit, gross margin, year, quarter, month, day of month, and week of month. Customer age, year, quarter, month, day of month, and week of month are more likely to participate in constraints than in calculations and should thus appear in dimension tables. Percentages and rations, such as gross margin, are non-additive. The numerator and denominator should be stored in the fact table. The ratio can be calculated in a data access tool by calculating the ratio of the sums (not the sums of the ratios!).

All non-fact attributes must appear in the dimension tables. Given the above scenario, it is natural to have a customer, product, and date dimension. Note that cost price and sales price are not part of the product dimension. These attributes are rather measures than descriptive attributes. Furthermore, the prices are likely to change frequently. While it is possible to introduce changes to dimensions it is more complex.

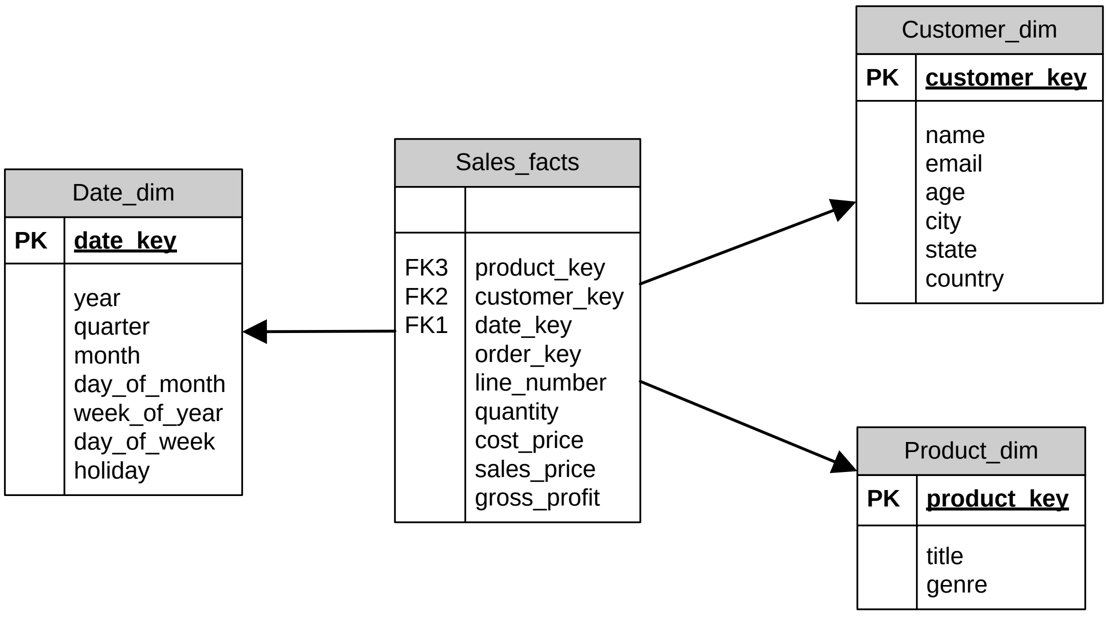

Figure 1: Star schema

b. What are the differences between a star schema and a schema in third-normal-form? What are the differences between a star schema and a snowflake schema? What are the advantages and drawbacks of either approach?

Dimensional modeling is quite different from third-normal-form (3NF) modeling. 3NF modeling is a design technique that seeks to remove data redundancies. Data is divided into many discrete entities, each of which becomes a table in the relational database.

Normalized modeling is immensely helpful to operational processing performance because an update or insert transaction only needs to touch the database in one place. Normalized models, however, are too complicated for data warehouse queries. Relational database management systems can't query a normalized model efficiently; the complexity overwhelms the database optimizer. The use of normalized modeling in the data warehouse presentation area defeats the whole purpose of data warehousing, namely, intuitive and high-performance retrieval of data.

(Ralph Kimball, Margy Ross: The Data Warehouse Toolkit, Wiley, 2002)

Dimension table normalization typically is referred to as snowflaking. Redundant attributes are removed from the flat, denormalized dimension table and placed in normalized secondary dimension tables.

(Ralph Kimball, Margy Ross: The Data Warehouse Toolkit, Wiley, 2002)

Snowflaking prevents update anomalies caused by denormalized models. However, queries may require more joins to be evaluated and query processing may therefore be less efficient.

2) Data Analysis in SQL

In SQL:1999 the GROUP BY clause was extended for improved data analysis. These extensions, invoked with the keywords CUBE, ROLLUP, and GROUPING SETS, provide multi-dimensional summaries for grouped data.

a. Reconsider the star schema designed in 1). Specify SQL queries to retrieve the following information. (Assume that a view named `sales` has been defined that joins the fact table and the dimension tables of the star schema).

   1) The revenue (sum of sales price) of book sales per genre and year.

```
select genre, year, sum(sales_price) as revenue
from sales
group by genre, year;
```

   2) The revenue of book sales per genre and per year with subtotals for each year and the grand total.

```
select genre, year, sum(sales_price) as revenue
from sales
group by ROLLUP (year, genre)
```

3) The revenue per city, state, and country with subtotals for each state and country, subtotals for each country, and the grand total.

```
select city, state, country, sum(sales_price) as revenue
from sales
group by ROLLUP(country, state, city);
```

4) The revenue per city, state, and country with subtotals for each state and country, and subtotals for each country.

```
select city, state, country, sum(sales_price) as revenue
from sales
group by GROUPING SETS((country, state, city),
    (country, state), (country));
```

5) The average order total per calendar month, the average order total per calendar year, the average order total per month and year, and the overall average order total.

```
select year, month, avg(order_total) as avg_order_total
from (
    select year, month, sum(sales_price) as order_total
    from sales
    group by year, month, order_key) as order_total
group by CUBE(year, month);
```

6) The revenue of book sales in 2009 per genre and per state and country (in combination) with subtotals for each genre and the grand total.

```
select genre, state, country, sum(sales_price) as revenue
from sales
where year = 2009
group by ROLLUP(genre, (state, country));
```

7) The revenue of book sales in each country per year, per quarter, per year and quarter, and the grand total.

```
select country, year, quarter, sum(sales_price) as revenue
from sales
group by country, CUBE(year, quarter);
```

8) The gross margin of book sales by genre and year and the grand total.

```
select genre, year,
       sum(gross_profit) / sum(sales_price) as gross_margin
from sales
group by GROUPING SETS((genre, year), ());
```

b. Rewrite the following queries using alternative grouping features.

1) Use ordinary grouping to rephrase the following query.

```
select genre, year, sum(sales_price) as revenue
from sales
group by ROLLUP (year, genre)
```

```
select genre, year, sum(sales_price) as revenue
from sales
group by genre, year
union
select CAST(NULL AS VARCHAR(50)) as genre, year,
       sum(sales_price) as revenue
from sales
group by year
union
select CAST(NULL AS VARCHAR(50)) as genre,
       CAST(NULL AS INTEGER) as year,
       sum(sales_price) as revenue
from sales;
```

2) Use GROUPING SETS to rephrase the following query.

```
select month, day_of_month,
       avg(order_total) as avg_order_total
from (
    select month, day_of_month,
           sum(sales_price) as order_total
    from sales
    group by month, day_of_month, order_key)
group by CUBE(month, day_of_month);
```

```
select month, day_of_month,
       avg(order_total) as avg_order_total
from (
    select month, day_of_month,
           sum(sales_price) as order_total
    from sales
    group by month, day_of_month, order_key)
group by GROUPING SETS((month, day_of_month), (month),
                       (day_of_month), ());
```

3) Use GROUPING SETS to rephrase the following query.

```
select year, quarter, month, sum(gross_profit)
from sales
group by year, ROLLUP(quarter, month);
```

```
select year, quarter, month, sum(gross_profit)
from sales
group by GROUPING SETS((year, quarter, month),
                       (year, quarter), (year));
```

4) Use GROUPING SETS to rephrase the following query.

```
select genre, country, state, avg(quantity)
from sales
group by GROUPING SETS(ROLLUP(genre),
ROLLUP(country, state));
```

```
select genre, country, state, avg(quantity)
from sales
group by GROUPING SETS((genre), (), (country, state),
                       (country), ());
```

3) Window Functions in SQL

The most fundamental enhancement that SQL/OLAP adds to the SQL language is the notion of windows – a user-defined selection of rows within a query that determines the set of rows used to perform certain calculations.

Reconsider the star schema designed in 1). Specify SQL queries to retrieve the following information. (Again, assume that a view named `sales` has been defined that joins the fact table and the dimension tables of the star schema).

a. The total amount of each order in 2009 together with the moving average over the last three orders of the respective customer.

```
select order_key, sum(sales_price) as order_total,
    avg (sum(sales_price)) over (PARTITION BY customer_key
    ORDER BY month, day_of_month ROWS 2 PRECEDING)
    as moving_avg

from sales

where year = 2009

group by month, day_of_month, order_key, customer_key;
```

b. The total amount of each order in 2009 together with the moving average over orders placed in the last three month by the respective customer.

```
select order_key, sum(sales_price) as order_total, month,
    avg (sum(sales_price)) over (PARTITION BY customer_key
    ORDER BY month RANGE 2 PRECEDING)
    as moving_avg_order_total

from sales

where year = 2009

group by month, order_key, customer_key;
```

c. The monthly sales and the cumulative sales for each genre in 2009 ordered by genre and month.

```
select genre, month, sum(sales_price) as genre_total,
    sum (sum(sales_price)) over (PARTITION BY genre
    ORDER BY month ROWS UNBOUNDED PRECEDING)
    as cumulative_total

from sales

where year = 2009

group by genre, month

order by genre, month;
```

d. The actual sales in each state together with the average sales in that state in the previous three month.

```
select state, month, year, sum(sales_price) as state_total,
    avg (sum(sales_price)) OVER (PARTITION BY state
    ORDER BY month ROWS BETWEEN 4 PRECEDING AND 1 PRECEDING)
    as three_month_avg
```

```
from sales
group by state, year, month
order by state, year, month;
```

e. A ranking of the bestselling books in 2009.

```
select title, sum(quantity) as total_quantity,
    RANK() OVER (ORDER BY sum(quantity) DESC) as rank,
    DENSE_RANK() OVER (ORDER BY sum(quantity) DESC)
    as dense_rank,
    ROW_NUMBER() OVER (ORDER BY sum(quantity) DESC)
    as row_number
from sales
where year = 2009
group by title;
```

f. A ranking of the most valuable customers (highest order totals) in the last quarter of 2009 per country and per state.

```
select name, country, state,
    sum(sales_price) as quarter_total,
    RANK() OVER (PARTITION BY country, state
    ORDER BY sum(sales_price) DESC) as state_rank,
    RANK() OVER (PARTITION BY country
    ORDER BY sum(sales_price) DESC) as country_rank
from sales
where year = 2009 and quarter = 4
group by country, state, name;
```

g. The monthly order totals together with the cumulative order totals of customers from "Texas" in 2009.

```
select name, month, sum(sales_price) as month_total,
    sum(sum(sales_price)) OVER (PARTITION BY name
    ORDER BY month ROWS UNBOUNDED PRECEDING)
    as cumulative_total
from sales
where year = 2009 and state = 'Texas'
group by name, month;
```

h. The percentage contribution of cities to the total sales of the respective countries.

```
select distinct city,
    sum(sales_price) over (partition by city) /
    sum(sales_price) over (partition by country) * 100

from sales

order by city;
```