A short primer of multidimensional modeling

Hans Friedrich Witschel

Contents

1 Purpose

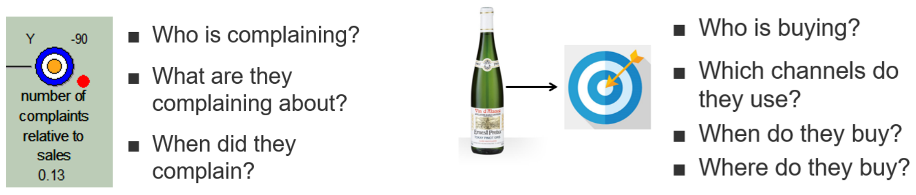
A multidimensional model helps to answer questions. That is, it supports *analytical* processing – which, as opposed to *transactional* processing – does not need to create, update or delete data records, but is based on *reading* the data only.

In general, the purpose of a data model is to describe all relevant information that should be stored in an information system. In principle, there are usually many possible ways of doing that, but there are some criteria for "good" data models:

- **Precision:** The model should reflect reality accurately
- **Granularity:** The model should be as detailed as necessary for serving its purpose
- **Complexity:** the model should be as easy to understand and as simple as possible. This helps to implement the model, but even more to maintain and update it later when it might become necessary.
- **Efficiency:** when translated into e.g. a database schema, the model should guarantee efficiency in terms of:
  - Query throughput, i.e. efficiency of read operations
  - Transaction throughput, i.e. efficiency of insertion, update and deletion operations
  - Disk space consumption
- **Consistency:** when implemented, the model should allow to easily keep data consistent, even in the face of frequent and possibly concurrent data insertions, updates or deletions. The most important prerequisite for achieving this, is usually non-redundancy of the model.

For analytical needs, transaction throughput and consistency do not need to be considered, since most processing is read-only and updates are never concurrent. Disk space consumption is nowadays also rarely an important constraint and should be readily sacrificed for query throughput if necessary – business users should not wait longer than necessary for their answers.

Thus, we need a way of modeling that will help to describe reality accurately, result in models that are easy to understand (possibly also for business stakeholders) and, above all, facilitates high query throughput for analytical queries. We will see later (see Section 6) that implementations of multidimensional models tend to result in redundancy of data storage – which speeds up queries and is non-critical because consistency is not a problem.



(a) (b)

Figure 1: Questions derived from a KPI

2 Supported analyses

How do we construct a multidimensional model? We best approach this question by analysing the types of questions that the model will help to answer.

2.1 Typical analytical questions

Broadly speaking, analytical questions may come from two sides:

1. Business performance management: business stakeholders want to monitor the achievement of strategic company goals and understand where deviations come from. For instance, when a company has formulated the strategic goal "increase customer satisfaction", it might derive an indicator (KPI) as shown in Figure 1 (a) and, when the KPI is off-target, the company might ask the questions that are shown next to the KPI.

2. Operational decisions: the execution of business processes involves decisions. For instance, when preparing a marketing campaign, one has to make decisions about whom to target, how to address the target audience and when to best launch the campaign. Figure 1 (b) shows the example of marketing a certain wine and the questions addressed to the outcome of previous campaigns of similar wines.

When we analyse the questions from Figure 1, we quickly find a pattern: they can all be translated into the following form:

show the <number of X> by <type of entity Y>

For instance, the question "Who is complaining" can be answered by showing the number of complaints (X) per customer or customer segment (Y) and then sorting the entries so that the customers with most complaints are shown first. Table 1 shows the questions from Figure 1 and several others and how they can be translated into the X-and-Y form described above. Obviously, in our pattern, X usually refers to a numeric variable and Y to one or more categorical variables, where the Y-part is meant to give a meaning to the numbers.

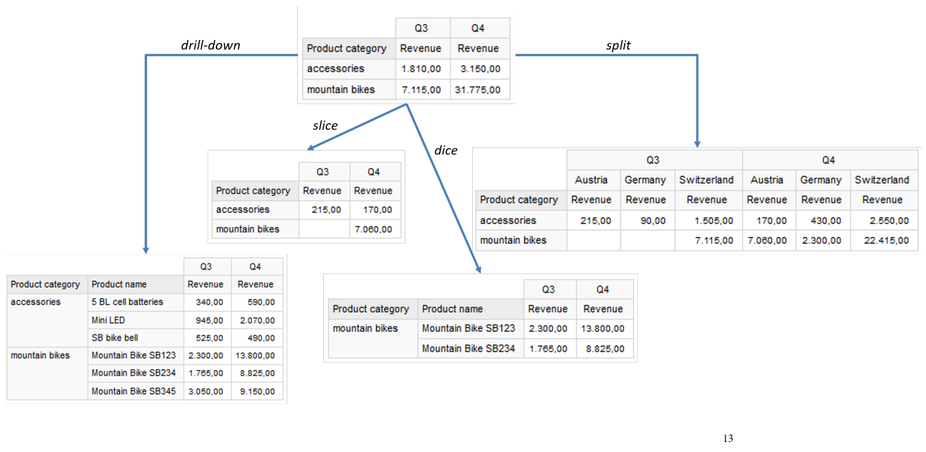
Question	X	Y
Who is complaining?	#complaints	customer (segment)
What are they complaining about?	#complaints	product / service
When did they complain?	#complaints	month
Who is buying	#sales	customer segment
Which channels do they use?	#sales	channel
When do they buy	#sales	month
Where do they buy?	#sales	store
Who are our most profitable customers?	profit	customer
In which claims area do we have the highest cost	cost	claims area
Which sellers' strategies work best to yield high sales?	revenue	seller's strategy

Table 1: Typical analytical questions and how they can be decomposed into a numerical variable (X), which is counted or summed up along the values of one or more categorical variables (Y)

2.2 Online-analytical processing (OLAP)

In this section, we introduce the operations of so-called *On-line Analytical Processing (OLAP)*, a technique for answering analytical queries such as the ones introduced in Section 2.1 above.

The general principle of OLAP is based on (pivot) tables: during an OLAP session, data is always displayed in a table with a matrix form where the inner cells contain numbers and the row and column headers contain the values of categorical variables to describe these numbers (see e.g. the tables displayed in Figure 2).



	Q3	Q4
Product category	Revenue	Revenue
accessories	1.810,00	3.150,00
mountain bikes	7.115,00	31.775,00

	Q3	Q4
Product category	Revenue	Revenue
accessories	215,00	170,00
mountain bikes		7.060,00

	Q3			Q4		
	Austria	Germany	Switzerland	Austria	Germany	Switzerland
Product category	Revenue	Revenue	Revenue	Revenue	Revenue	Revenue
accessories	215,00	90,00	1.505,00	170,00	430,00	2.550,00
mountain bikes			7.115,00	7.060,00	2.300,00	22.415,00

		Q3	Q4
Product category	Product name	Revenue	Revenue
accessories	5 BL cell batteries	340,00	590,00
	Mini LED	945,00	2.070,00
	SB bike bell	525,00	490,00
mountain bikes	Mountain Bike SB123	2.300,00	13.800,00
	Mountain Bike SB234	1.765,00	8.825,00
	Mountain Bike SB345	3.050,00	9.150,00

		Q3	Q4
Product category	Product name	Revenue	Revenue
mountain bikes	Mountain Bike SB123	2.300,00	13.800,00
	Mountain Bike SB234	1.765,00	8.825,00

Figure 2: Effects of OLAP operations

OLAP is interactive since it lets users easily change what is diplayed in rows, columns and inner cells of the table. This happens through a few simple operations that can be described as follows:

- **Drill-down / roll-up:** attribute values of categorical values are often structured hierarchically. With a drill-down, one can make the values of a lower hierarchy level visible (with a roll-up, one collapses them). Figure 2 shows an example: when starting from the table on the top, one may choose to display the products inside the two product categories (accessories and mountain bikes), i.e. one goes from the top view to the bottom view. The roll-up creates the top view from the bottom view, i.e. it hides the individual products. This requires the tool to aggregate the numbers in the cells: the table at the top shows the number of sales for all accessories and all mountain bikes.
- **Slice:** when the user chooses a slicing operation, this essentially means to filter the values in the cells by showing only those that are associated with one certain value of a certain categorical variable. In the example, the bottom view only shows the numbers for Austria (i.e. the "country" variable has been used as a filter). The special thing about slicing is that the resulting table does *not show* the values of the variable that is used for filtering.
- **Dice:** constraining one or more categorical variables to a (smaller) range of values is called dicing. As opposed to slicing, the resulting table shows these values. In Figure 2, dicing is done on the product variable, constraining its values to two chosen bikes.
- **Split / merge:** although computer displays are two-dimensional, OLAP can show values for more than two categorical variables. A split operation adds values of another categorical variable to the table and splits up the values in the cells accordingly. A merge operation takes away a variable and aggregates the numbers (similar to a roll-up) across all values of the variable. Figure 2 shows a split that puts the country variable on columns: since the top view has already a categorical variable used as column headers, the tool creates *all possible combinations* of quarters (which are already on columns) and countries.

A final OLAP operation that deserves being mentioned is the **drill-through**: this allows a user to see all the original transaction records from which a value in a cell results. In the example, the user could select a cell (e.g., at the top of Figure 2, the upper left cell containing the revenue from accessories in Q3) and see all individual sales transactions from which the number results.

3 Key concepts

3.1 Measurements, facts and dimensions

The analytical questions introduced in Section 2.1 all ask for numerical data which is grouped, filtered and aggregated along the values of categorical variables (see again Table 1). We have seen an example of how the data can be interactively explored through OLAP operations in Section 2.2.

Where does the data come from that is used to answer such questions? Such data usually results from series of *measurements* which are made to reflect the outcome or other quantitative characteristic of instances of a certain business process.

Measurements may be triggered by events (e.g. an order is placed, a purchase is made, a user clicks on a link etc.) or can be generated by making periodic snapshots of a certain system. We call each measurement a **fact**. With each measurement, we can associate several **measures**, i.e. things that get measured simultaneously.

> **Example:** Consider the business process "sales in a supermarket". We might want to make a measurement each time a customer checks out with his/her shopping cart. In that case, the check-out event is a *fact*. With each check-out event, we can measure e.g. the number of items bought and the total amount paid by the customer. These two numbers are then called *measures*. In this example, facts are given by events – and for these events, data is being captured anyway in the cash register.

Depending on the business process, there are often several possibilities to define when a measurement should be made – i.e. how often or triggered by which kind of event within process instances. The definition of when/how often a measurement should be made is called the **grain** of a multidimensional model.

> **Example:** In the supermarket, we have at least the following three possible grains:
>
> - closing of cash register in one store on one day
> - the shopping cart of a customer
> - a single line item from a customers shopping cart
>
> Obviously, the further we go down in this list, the more frequent are the measurements, i.e. the greater the number of generated facts.

When we store the measures associated with a fact, we need to include more information that describes the event or situation from which the measurement resulted. This is done via categorical attributes, called **dimensions**. Usually, the definition of a grain implies a certain set of dimensions. Additional dimensions can be introduced to satisfy analytical needs – but they must be "**true to the grain**", i.e. it must be possible to associate one single value of each dimension to a given fact.

> **Example:** The event of closing a supermarket's cash register in a store can be completely described by specifiying a date and a store.

The two categorical variables "date" and "store" are then called *dimensions*. When we choose a different grain, we can associate more dimensions: e.g. a shopping cart of a customer can be described by the same dimensions (date and store), as well as the additional dimension "customer". Note that the dimension customer is *not* true to the coarser grain because there are usually many customers that we would need to associate with the closing of a cash register – and truth to grain requires dimensions to have one unique value for each fact. Finally, line items from a shopping cart can be additionally associated with a product dimension. In the line item case, we can also introduce an additional dimension that is not necessary to specify the measurement completely, namely e.g. the promotion under which the product was sold. This dimension is *true to the grain* if we can guarantee that there is always only one promotion for a given product.

Later, the dimensions will be the "by-words" (see last column in Table 1) in analytical questions. When we imagine such questions, they usually imply aggregation of measures: when we ask e.g. for the number of sales per store, then we usually want these to be aggregated over a certain range of dates. That is, questions that do not mention all dimensions imply that values should be aggregated over all values of these dimensions. In our example, the aggregation happened by summing up, in other situations averaging or counting may be applied – these different ways of aggregating are called **aggregation functions**.

Values of dimensions can also often be organised into so-called **dimension hierarchies** (see the drill-down and roll-up OLAP operations in Section 2.2). This may happen when there is a "is-a" relationship between values (e.g. between products and product groups: a Ferrari *is a* car) or when there is a "part-of" relationship (e.g. between a store and a city or a city and a country). Hierarchies must be defined in a way that guarantees a 1:n relationship between level elements. Analytical questions are often asked on a higher level of dimension hierarchies – which again implies aggregation of values.

Example: In the supermarket example, sales are always aggregated by summing them up. When we ask for e.g. the sales from last year in Switzerland, we address the higher levels of the two *dimension hierarchies* date (date $\rightarrow$ month $\rightarrow$ quarter $\rightarrow$ year) and store (store $\rightarrow$ city $\rightarrow$ region $\rightarrow$ country). Here, the 1:n relationship is always given: a store is in only ony city (but a city can have multiple stores), a city is only in one region etc. Since the question asks for the hierarchy levels year and country, it is implied that the measures (e.g. revenue) must be summed over all dates in the year and all stores in the country.

3.2 Examples

In this section, we explore some typical examples of multidimensional models that come from various parts of the value chain and from various industries.

Table 2 shows examples from the various parts of the value chain, specifying each time the part of the value chain, the grain, the measures and the dimensions (not including hierarchies).

Part of value chain	**Facts**	**Fact measures**	**Dimensions**
Inbound logistics	Orders	quantity, amount	product/material, supplier, date, contract terms, transcation type
Outbound logistics	(line items of) quotes, orders, shipments, invoices	quantity, amount	date, sales rep, shipped from, shipper, (product)
Sales	Sales	quantity, revenue, cost, profit	store/channel, product, date, customer, promotion
Service	Complaints, warranty claims	count	customer, date, channel, product

Table 2: Typical examples of facts and dimensions for various parts of the value chain

Table 3 shows typical examples of facts, measures and dimensions for various industries.

Industry	**Facts**	**Fact measures**	**Dimensions**
Retail	Sales	quantity, revenue, cost, profit	store/channel, product, date, customer, promotion
Banking	account snapshots	balance, interests, fees	customer, account, product, branch, date
Banking	Transactions	count, value, fee	accounts, customers, regions, transaction type
Insurance	policy sales	premium fee	policy, policy holder, covered risk, date
Insurance	claims	count, cost	policy, policy holder, covered risk, date
Online	webshop visits	count, duration, number of clicks, revenue	customer, referring website, landing page, date

Table 3: Typical examples of facts and dimensions for various industries

For even more examples, see [2].

4 Construction

According to Kimball [2], the construction of a multidimensional model should be done along the following four steps:

1. Select the business process.
2. Declare the grain.
3. Identify the dimensions.
4. Identify the facts.

If we assume that a business process is given (by the kind of decisions that should be supported), we need to start with declaring the grain. The most important advice is to start this and the following steps by analysing the *information needs of the business stakeholders* and *not* the transactional data that is available!

Thus, when we need to declare the **grain**, we first collect a list of questions that the business has towards the data, i.e. questions such as the ones in Table 1. By analysing them (as explained in Section 2.1), we find which dimensions are involved in the questions. This allows us to select a suitable grain.

> **Example:** in the case of the supermarket POS data, if business stakeholders tell us that they want to analyse revenue by product category, we know that we need to choose the finest grain (i.e. single line items) because it is the only one with which a product dimension can be associated.

Note: even if business stakeholders do not mention dimensions associated with a fine grain, it may be advisable to choose the finest (so called *atomic*) grain, i.e. the lowest level at which data is or can be captured – this offers the highest degree of flexibility in terms of accomodating future information needs. It also incurs higher storage cost (more facts need to be stored), but when considering today's cost for storage, this is often not a serious argument. However, in some environments, it can make sense to estimate roughly the number of facts that will accumulate during a year in order to make a judgment on feasibility.

> **Example:** for a supermarket chain, we may work on some assumptions to estimate the number of facts that will accrue: let's assume that the chain only operates in one country and has 100 stores. On a typical day, we may assume that each store is visited by 1,000 people. If an average shopping basket contains 10 line items, using the atomic grain will produce 300 million facts per year (assuming 300 opening days in a year). In contrast, if a fact is defined by the closing of a cash register on one day in one store, we only get 100 facts per day, i.e. 30,000 facts per year. Storage and efficient querying of 300 million records places a different demand on underlying database systems when comapred with 30,000 records. But one

should not underestimate the additional vaule that can be gained from being able to analyse sales by product! When comparing that value to the additional cost, one will likely decide to invest more.

Having fixed the grain, we can choose dimensions and measures – we need to know the grain because dimensions and measures need to be chosen such that they are true to that grain. Remember that this means that it is always possible to associate exactly one value of a given dimension/measure with each fact.

As far as **dimensions** are concerned, there are usually some that one uses to define the grain – these are the dimensions implied by the grain. As described above in Section 3.1, it is possible to add further dimensions – as long as they are true to the grain – as required by the analytical needs of the business.

Obviously, we need to check here the realities of the underlying source data and which information it contains.

Another step in choosing dimensions is to check whether dimension values can be hierarchically organised and thus to derive dimension hierarchies. Hierarchies are often obvious, but one should take care that adjacent levels are in a true 1:n relationship.

Example: In the case of the atomic grain in the supermarket example, the dimensions date, store, customer and product are implied by the grain. Recall that we said that a further dimension "promotion" can be introduced if a product is always sold under only one promotion. We also need to check whether promotion information is available in the source data. The same applies to the customer dimension: if customers do not use loyalty cards while shopping, we may not be able to recognise them!

We can organise most of the mentioned dimensions into hierarchies: dates can be grouped by month, quarter and year, stores by city, region and country, customers by segments and products by product categories. These relationships are all 1:n – for instance, each store belongs to exactly one city. In contrast, introducing a "week" hierarchy level in between the date and month levels is problematic since a week may belong to two different months.

Finally, we choose the **measures** to be associated with each fact. Such measures can be calculated, but – besides being true to the grain – they should usually be summable (or "additive"), i.e. summing them up should be possible and make sense from a business perspective. We also need to define the aggregation function to be associated with each measure, the most typical one being the sum.

Example: In the supermarket case, using the atomic grain, the following measures are reasonable:

- quantity

- revenue (quantity times unit price)
- cost of purchase and storage of the goods
- gross profit (revenue minus cost)

Gross profit is a calculated measure – but it is possible to sum it up and the sum makes sense from a business perspective. The following measures are more problematic:

- unit price of the line item: this is true to the grain, but the sum does not make sense from a business perspective: if we sold 10 bananas for $1 each and 5 apples for $0.80 each to a customer, what does the sum of unit prices ($1.80) tell us about this sales transaction?
- gross margin (gross profit divided by revenue): ratios are always problematic because the sum of the ratios is not equal to the ratio of the sums. If we sell a product for $10, with a gross profit of $1 and another product for $15 with a gross profit of $3, the margin is 0.1 in the first case and 0.2 in the second. Overall, we had a revenue of $25 and a gross profit of $4, resulting in a gross margin of 0.16 – which is obviously different from the sum of the two individual gross margins (0.3). However, most BI tools can be configured such that they compute a ratio from the sums of numerator and denominator of the ratio instead of summing individual ratios.

5 Modeling languages

Having gone through the four steps of Kimball's process, we have identified facts, measures, dimensions and their hierarchies. In order to document these, it is good practice to use a graphical modeling language – this makes it easier to discuss about the model and to implement it later. Note that the notations described here are *independent of physical storage systems*, i.e. they provide a way to describe a multidimensional model conceptually. This is in contrast with e.g. a *star schema* (see Section 6) which describes the implementation of a multidimensional model in a relational database.

This primer describes the modeling languages Multidimensional Entity/Relationship (ME/R) and multidimensional UML (mUML). Another wide-spread language for multidimensional modeling, not described here, is ADAPT [1].

5.1 ME/R

ME/R is used to create very simple models – where attributes of dimensions cannot be modeled. That is, ME/R trades expressiveness for simplicity. There are three modeling elements, shown in Figure 3 (a):

- A cube symbol: the cube denotes what the facts are. The name of facts is written above a horizontal line within the cube. Below that line, one puts the name of the measures. The example in Figure 3 (b) indicates that the facts are called sales and defines the two measures quantity and revenue.

- Hierarchy levels: These are denoted by boxes with a hierarchy symbol inside. In ME/R one uses a straight line (actually a fourth element of the notation) to join the lowest level of a dimension hierarchy to the facts symbol – in the example, the store level.

- An arrow for hierarchical relations: arrows are used to connect adjacent hierarchy levels (as store and region in the example in Figure 3 (b)).

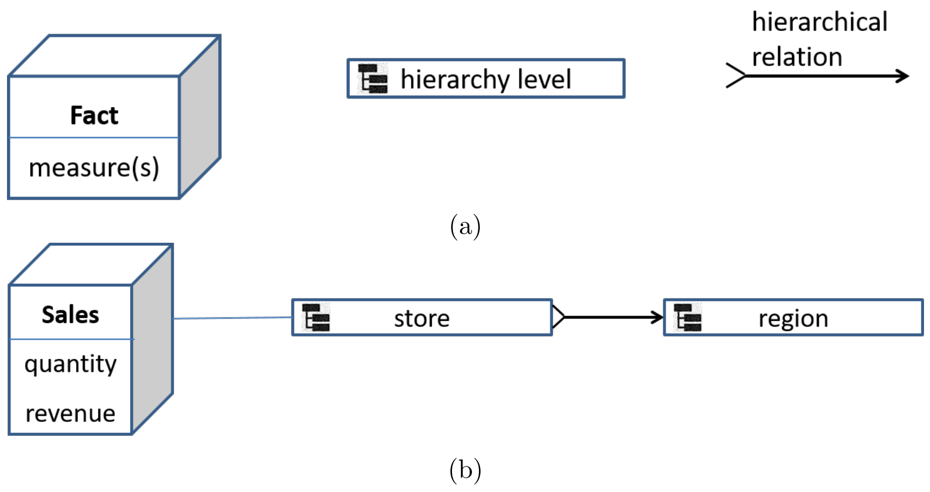


Figure 3: ME/R: (a) elements of the language and (b) an example model

5.2 mUML

mUML is more expressive than ME/R, but also results in significantly more complex models. It extends UML via stereotypes which give a special meaning to certain UML constructs:

- Special classes: <<fact-class>> and <<dimension-class>> for modeling facts and dimensions

- Special relations: <<dimension>> to connect a dimension to a fact element and <<roll-up>> to connect adjacent hierarchy levels

Figure 4 shows how the example model from Figure 3 (b) can be modeled with mUML. The dots "..." inside the dimension classes indicate that mUML allows to specify attributes of dimensions (e.g. the address of a store). This makes it more expressive than ME/R.

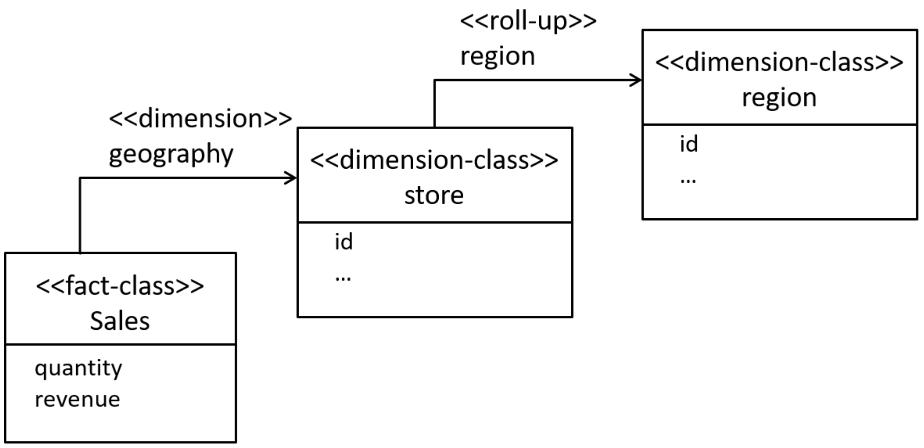


Figure 4: An mUML example model

6 Physical storage

As discussed in Section 1, the purpose of multidimensional models is to describe the information that should be stored in a data warehouse to support analytical queries. As a main requirement, the implementation of the model should allow for efficient querying of the data.

In this primer, we will cover the two traditional ways of storage, namely using either relational databases or special "cube" databases based on the concept of multidimensional arrays.

6.1 Relational storage

When using a relational database to store a multidimensional model, there is a straightforward way to translate a model such as the one in Figure 4 (b) into a database schema. It is called a **snowflake schema**. In a snowflake schema, there is one table that contains the facts. Via foreign keys the fact table links to further tables which contain the lowest level of each dimension hierarchy. In a snowflake schema, each hierarchy level has its own table – each level links to the next level via foreign keys.

From now on, we will use a Swiss Bikes sales scenario as a running example: we assume a multidimensional model where each line item of a sales transaction in a Swiss Bikes store is a fact, we record for each fact the measures "units_sold" and "revenue" and we have three dimension hierarchies, namely store→province→country, product→product group and date→month→quarter→year.

Figure 5

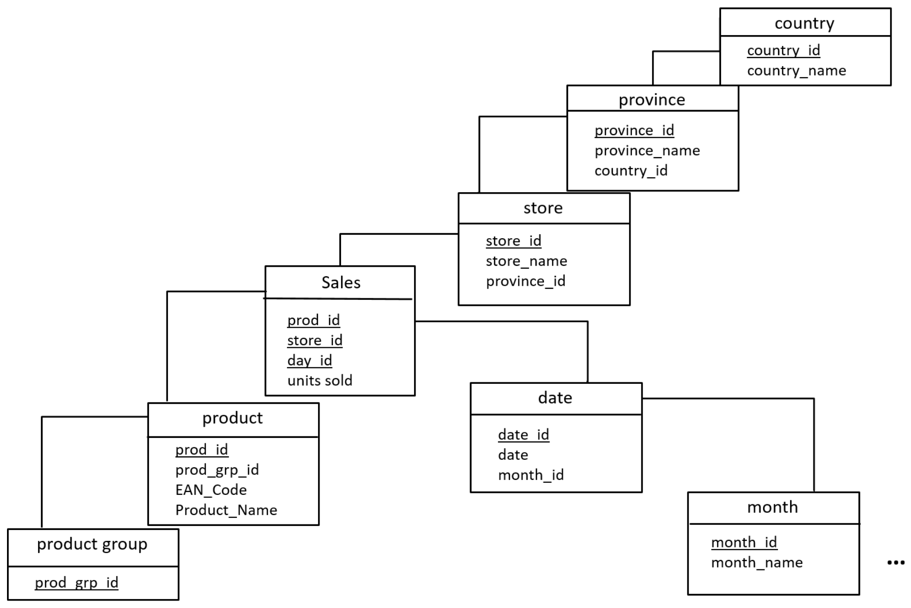


Figure 5: A snowflake schema for the Swiss Bikes sales scenario

The obvious problem with a snowflake schema is that it does not meet the requirement of efficient querying: when we want to e.g. display the sum of revenue for Switzerland, we would need the following SQL query:

```
SELECT sum(sa.units_sold)
FROM
        sales sa INNER JOIN store s on sa.store_id=s.store_id
        INNER JOIN province p on s.province_id=p.province_id
        INNER JOIN country c on p.country_id=c.country_id
WHERE c.country_name =Switzerland
```

That is, we need to do three joins (see the red highlighting in the query) – something which can be very time-consuming.

A snowflake schema follows a *normalised* approach to representing the information in the model. That is, redundancy is avoided. Redundancy – as discussed in Section 1 – is a problem in systems where many concurrent updates might be applied to the data, possibly leading to inconsistencies. Since we are considering a purely analytical use case here, consistency is not a problem.

A so-called **star schema** introduces redundancy in order to speed up queries by *de-normalising* the tables: instead of having a separate table for each hierarchy level, a star schema has only one table per dimension hierarchy, where the hierarchy levels are attributes in these tables. Figure 6 shows the star schema for the Swiss Bikes scenario.

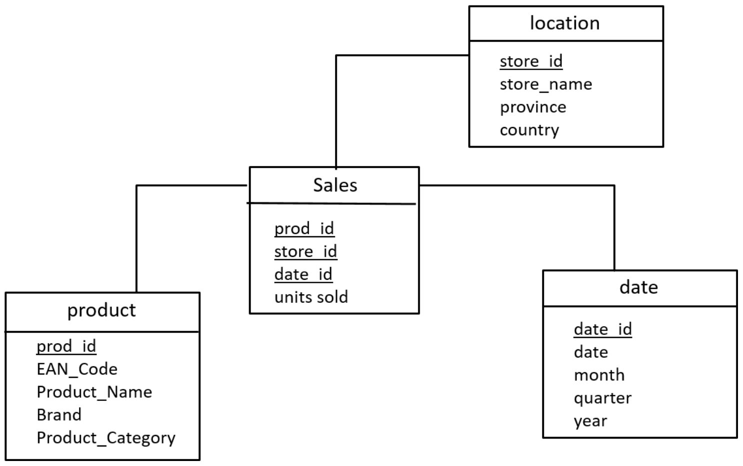
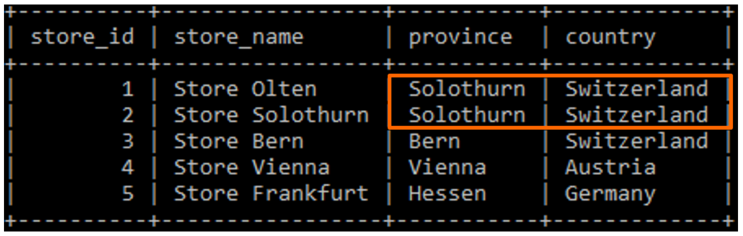


Figure 6: A star schema for the Swiss Bikes sales scenario

Since province and country are now attributes of a store, the fact that a certain province is part of a certain country will be stored redundantly as shown in Figure 7.



```
+----------+-----------------+-----------+-------------+
| store_id | store_name      | province  | country     |
+----------+-----------------+-----------+-------------+
|        1 | Store Olten     | Solothurn | Switzerland |
|        2 | Store Solothurn | Solothurn | Switzerland |
|        3 | Store Bern      | Bern      | Switzerland |
|        4 | Store Vienna    | Vienna    | Austria     |
|        5 | Store Frankfurt | Hessen    | Germany     |
+----------+-----------------+-----------+-------------+
```

Figure 7: An extract from the Swiss Bikes store dimension table, showing an example of redundancy in a star schema

However, we can easily see how this denormalisation speeds up queries by considering again the SQL query needed to retrieve the sales for Switzerland:

```
SELECT sum(units_sold)
FROM sales sa INNER JOIN store s on sa.store_id=s.store_id
WHERE s.country="Switzerland";
```

Obviously, this will be faster since it only needs one join operation instead of three. The increased storage requirements are not a severe problem in practice because for reasonably small hierarchies, the size of the denormalised dimension tables (which potentially enumerate all paths through the hierarchy tree) is still much smaller than the size of the fact table: there are usually millions of facts, but a comparatively much smaller number of possible paths through a hierarchy tree.

6.2 Physically multidimensional storage

Some vendors of BI systems have developed proprietary databases that are optimised for querying multidimensional data. To this end, data is stored in so-called **cubes** which are essentially multidimensional arrays. Figure 8 shows an example of a cube with three dimensions (country, product and date). Note that in the (more usual) case of a cube with more than three dimensions, it is not possible to visualise it in that way – i.e. it can be misleading to think of a cube in such a three-dimensional way.

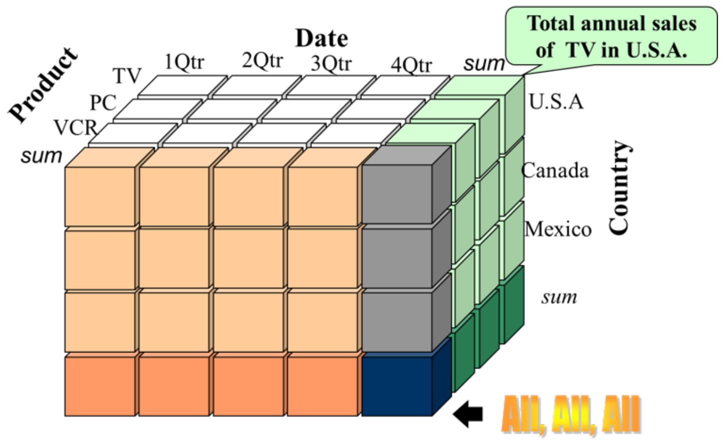


Figure 8: A simple cube with dimensions country, product and date for a sales scenario

In a cube, the measures are stored in the cells. The cells are addressed via integers which are mapped to the values of the dimensions (including their hierarchy levels). Figure 9 shows an example of such a mapping for the Swiss Bikes sales scenario, enumerating all the values of the three dimensions store, product and date.

For a query "retrieve the total revenue and units sold in Switzerland", the system would first look up the values of the dimensions – here, "Switzerland" is the value of the store dimension and we use "ALL" as a value for the product and date dimensions since, according to the query, values should be aggregated over all products and dates. In a next step, the values are mapped to integers – in the example, [Switzerland, ALL, ALL] becomes [10,9,14] based on the mappings defined in Figure 9. With these indices, the values of the measures units sold and revenue can be fetched from the multidimensional array.

In this example, the cube consists of $13 \cdot 9 \cdot 14 = 1,638$ cells.

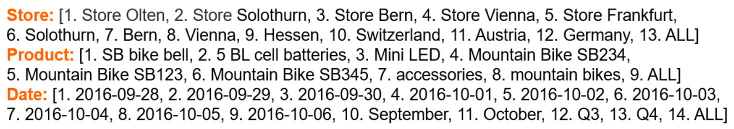

Store: [1. Store Olten, 2. Store Solothurn, 3. Store Bern, 4. Store Vienna, 5. Store Frankfurt, 6. Solothurn, 7. Bern, 8. Vienna, 9. Hessen, 10. Switzerland, 11. Austria, 12. Germany, 13. ALL]
Product: [1. SB bike bell, 2. 5 BL cell batteries, 3. Mini LED, 4. Mountain Bike SB234, 5. Mountain Bike SB123, 6. Mountain Bike SB345, 7. accessories, 8. mountain bikes, 9. ALL]
Date: [1. 2016-09-28, 2. 2016-09-29, 3. 2016-09-30, 4. 2016-10-01, 5. 2016-10-02, 6. 2016-10-03, 7. 2016-10-04, 8. 2016-10-05, 9. 2016-10-06, 10. September, 11. October, 12. Q3, 13. Q4, 14. ALL]

Figure 9: An enumeration of values of the dimensions in the Swiss Bakes sales scenario, together, each value having an integer assigned to it that will be used as index.

7 Implementation of analytical querying

7.1 Important types of aggregations

Before we describe how analytical queries can be answered in the two different storage paradigms, we recall that typical analytical queries often involve aggregation of values. Aggregations can be categorised into two cases:

- Aggregating across all values of other dimensions: this happens when a query only mentions one dimension and omits the others. If we ask, for example, for the total revenue from sales in the Swiss Bikes Store in Olten, we implicitly assume that the revenue will be summed up over all dates and all products contained in the other two dimensions.

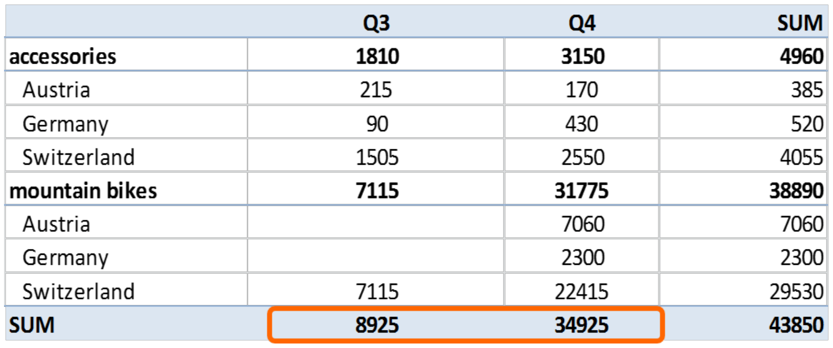
- Aggregating across values of lower hierarchy levels: when we ask e.g. for sales revenue in Switzerland, we imply that revenue should be summed up over all

In OLAP, we may also choose one dimension (e.g. store) and drag and drop it onto columns or rows to display the sales revenue per store, but we may omit the other dimensions or we may remove a dimension from the view (an OLAP *merge* operation). Similarly, when we move to a higher hierarchy level (OLAP roll-up), we imply that all values of that level will be displayed and values will be summed across the values of the lower levels.

Selecting a subset A of the available dimensions and summing measure values over all remaining dimensions, results in a so-called **A-cuboid**. In Figure 8 the cuboids have different colours: the (country)-cuboid is grey, the (country,product)-cuboid light green, the (product)-cuboid dark green etc.

For end users, cuboids become visible in pivoted tables, in the form of (partial) sums: Figure 10 shows a pivoted representation of a three-dimensional cube with the dimensions date (values Q3 and Q4), product category (values accessories and mountain bikes) and country (values Austria, Germany and Switzerland). The orange rectangle in the figure highlights the (date)-cuboid: it contains the total revenue of sales in Q3 and Q4, each summed over all three countries and the two product categories.

A multidimensional model with d dimensions has 2^d cuboids associated with it since that is the number of all possible subsets A of dimensions. Thus, the



	Q3	Q4	SUM
accessories	**1810**	**3150**	**4960**
Austria	215	170	385
Germany	90	430	520
Switzerland	1505	2550	4055
mountain bikes	**7115**	**31775**	**38890**
Austria		7060	7060
Germany		2300	2300
Switzerland	7115	22415	29530
SUM	**8925**	**34925**	**43850**

Figure 10: A pivoted representation of a three-dimensional cube, in which the (date)-cuboid is highlighted

number of cuboids grows exponentially with the number dimensions – hence pre-computation of the cuboid aggregates can be too expensive for models with a large number of dimensions.

7.2 Implementation of ROLAP

When we realise an OLAP tool on top of a relational storage of a multidimensional model, we speak of **Relational OLAP (ROLAP)**. For the discussion of this kind of implementation, we assume that the data is stored in a relational database with a star schema, as shown in Figure 6.

7.2.1 Basic OLAP operations

In order to demonstrate the implementation of basic OLAP operations, we consider the following SQL query:

```
SELECT
  P.product_category,
  D.quarter
  SUM(S.revenue)
FROM
  Product P, Date D, Sales S
WHERE
  P.product_id=S.product_id AND
  D.date_id=S.date_id
GROUP BY
  P.product_category, D.quarter
```

This is a typical query to generate an OLAP view – in fact, it will generate the data needed to produce the view at the top of Figure 2.

The underlying query pattern is called a **star join**. The star join joins the fact table with the necessary dimension tables (in the example, product and date) and applies the SUM function together with a GROUP BY clause that uses the chosen dimensions (and thus sums over the values of all other dimensions).

Based on this pattern, we can implement the OLAP operations from Figure 2 as follows:

- A *drill-down* operation, e.g. to the level of individual products, can be realised by adding "P.product" to both the SELECT and GROUP BY clauses.
- A *slice* operation, e.g. constraining the values to sales in Austria as in Figure 2, can be done by creating another join between the sales table and the slicer table (in this case the store table st) and then adding

  ```
  AND st.country="Austria"
  ```

  to the WHERE clause.
- *Dicing* is done by specifying a range for some of the dimensions, e.g. by adding

  ```
  AND p.product_name in
  ("Mountain Bike SB123","Mountain Bike SB234")
  ```

  to the WHERE clause of the star join to produce the dice result in Figure 2.
- Finally, for a *split* operation – e.g. adding the store dimension as in Figure 2 – one would need to add the desired attribute of the store dimension (e.g. the country) in the SELECT and GROUP BY clauses and add another join in the WHERE clause, joining the store dimension table to the sales table as for the other dimensions using the store_id. Conversely, a merge operation can be realised by removing e.g. the date dimension from all clauses.

7.2.2 Pre-computing aggregates: cube and roll-up

The SQL standard (since SQL 3) includes the possibility to define a GROUP BY clause "with CUBE" and "with ROLLUP". This supports the precomputation of aggregates that are required for the OLAP operations split/merge and drill-down/roll-up.

For instance, one may specify a "GROUP BY quarter, product, country WITH CUBE" (the exact syntax may vary in different implementations) which will create all possible 8 cuboids as shown in Figure 8.

Similarly, using a "GROUP BY country, province, store WITH ROLLUP" will create cuboids for the mentioned dimension hierarchy levels. For roll-up, one only needs to consider the cuboids that sum the values across lower hierarchy levels – in the example: (country), (country, province) and (country, province, store). A (store) cuboid – which would contain sales figures for each store, but summed over all provinces and countries – does not make sense because e.g. the "sales at store Bern, summed over all countries" is meaningless.

Theoretically, the result of the CUBE and ROLLUP operators can also be reached with standard GROUP BY clauses, by selecting each cuboid via a star join (see above) and uniting the results. If you are interested to learn how this works internally, please watch the according video on Moodle.

7.3 Implementation of MOLAP: MDX

Answering analytical queries in a physically multidimensional database is very simple: it comes down to addressing the right cells in the cube (multidimensional array) and fetching their values.

Microsoft has developed a query language called "MultiDimensional eXpressions" (MDX) which allows to specify queries in human-readable syntax that address cells. In MDX, one can e.g. write [Store].[Switzerland] to address the value "Switzerland" in the Store dimension. To actually fetch the data, the system looks up the index of that value in the list of dimension values (see Figure 9), which will yield "10" in this example. It then uses this index to address cells in the cube.

For more information about how to use MDX to implement OLAP operations, please consult the corresponding video on Moodle.

8 Hybrid storage and querying (HOLAP)

8.1 ROLAP vs. MOLAP

Above, we have introduced two alternative ways of storing multidimensional models. Both ways are represented in tools of various vendors and thus, the question arises, when to choose which paradigm. Before we enter into this discussion, let us recap the most important properties of each paradigm:

- **ROLAP** stores multidimensional models in relational databases; data retrieval is based on star joins. In most cases, ROLAP implementations do not precompute aggregates, i.e. aggregation is done on the fly at runtime when requested by users.
- **MOLAP** uses multidimensional arrays to store models, where dimensions are used to address the cells, which contain the values of fact measures. This storage requires that all possible aggregates (cuboids) for all combinations of dimensions and their hierarchy levels are precomputed and stored physically in the array. This means that querying at runtime simply fetches values that are already computed in advance.

Considering these properties, we can derive some pros and cons of the two paradigms, as presented in table 4.

ROLAP	**MOLAP**
Pros	*Cons*
more scalable	hard to scale to a large number of dimensions
fast load times	longer load times (precomputation of aggregates)
incremental updates possible	updates require full cube re-computation
maintenance requires only SQL knowledge	maintenance requires knowledge of proprietary tools and languages
flexible models: can represent partly non-multidimensional data	can represent only multidimensional data
Cons	*Pros*
slow queries (aggregation at runtime, many joins)	very fast queries (due to natural indexing)
pre-aggregation hard to implement (need to avoid "double aggregation")	pre-aggregation comes naturally

Table 4: Pros and cons of relational and multidimensional OLAP

The scalability issue (first row in table) derives from the fact that a multi-dimensional model with n dimensions results in the necessity to pre-compute 2^n cuboids – this exponential growth in computing time and required storage space can have a negative impact on MOLAP since everything needs to precomputed there. In addition (second row of table), MOLAP cannot update a cube incrementally since aggregates are usually invalidated when the underlying data changes, i.e. all cuboids need to be computed from scratch.

> **Example:** Let us assume a cube as the one shown in Figure 8 and let us assume that we are currently in Q4. Let us further assume that the cube should be updated at the end of each week with the new sales transactions that have been recorded during that week. Obviously, the data collected during the week requires updates in those white cells in the figure that correspond to Q4, whereas all other white cells can be left unchanged. However, the changes in white Q4 cells also impact the colored cells (the cuboids): for instance, the values in the (country)-cuboid (grey cells in the figure) has to be updated: the newly recorded sales will change the sum of sales in each country. The same applies to the other cuboids – hence, a full re-computation of the entire cube is usually the easiest update strategy.

It should be mentioned here that, although ROLAP does not suffer from these problems, its scalability is also limited: ROLAP can become prohibitively slow for fact tables with a very high number of rows.

In summary, MOLAP is the best option for multimensional models with a limited number of dimensions for which purely multidimensional analyses with very low latency querying is required.

ROLAP is better suited for cases where one needs to scale to a large number of dimensions, and/or where flexibility is required, where budget for training is low and where longer response times are acceptable.

8.2 HOLAP approaches to partitioning

Sometimes, it is hard to make a decision between ROLAP and MOLAP because one needs both: flexibility and scalability on the one hand, and low latency queries on the other.

In terms of latency, however, one can often live with a compromise: there are some queries that are asked very often and hence require fast answers; for rare queries, a larger latency might be acceptable.

These considerations lead to **Hybrid OLAP (HOLAP)**, an approach that stores often-queried data in multidimensional arrays (MOLAP) and all other data in a relational database (ROLAP).

To achieve this, two types of data partitioning are popular:

- **Vertical partitioning:** stores aggregations in the multidimensional array for fast access, and fine-grained/detailed data in the relational database. This approach is based on the insight that users usually query aggregated data and drill-throughs or drill-downs to the lowest level of dimension hierarchies are rare. This approach also has the advantage that one can choose only higher levels of dimension hierarchies for building the MOLAP cube and thus ensure scalability by reducing the number of cuboids.

- **Horizontal partitioning:** stores often-queried slices or dices of data in the multidimensional arrays and the rest in a relational database. A very popular dimension for dicing data in horizontal partitioning is the time dimension: recent data is put into the array and older data into the relational database. This often works well because old data is not queried that often. However, with this approach scalability can remain an issue because it does not reduce the number of dimensions/cuboids to be dealt with by the MOLAP cube.

References

[1] Dan Bulos and Sarah Forsman. Getting Started with ADAPT. Technical report, Symmetry Corporation, 2006.

[2] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit.* John Wiley & Sons, Inc.