Advanced Data Management Technologies
Project Module 1 – Data Warehouse

Julian Aron Prenner

February 5, 2014

Contents

1 Domain, Requirements and Modelling

The *Covelano Murble Spa.*, a company in Silandro - South Tyrol, has specialized in the manufacturing, extraction and commercialization of marble and is considering the creation of a data warehouse to aid them in business decisions, mainly where to dig in the quarry and where to extend business efforts . Murble from this quarry has been used to built castles in South Tyrol, churches in Rome, mosques in the Near east and lately in the luxury flats of the One57 skyscraper in New York.

Murble blocks are excavated from the quarry at at 2.200m over the sea surface (it is the heighest marble quarry in Europe). These raw blocks are then pre-machined and cut into smaller blocks as to make them safely and easily transportable. At a later time the blocks are brought down by road into the valley to the storehouse next to the factory in Silandro.

Once arrived at the factory, the blocks are roughly sorted by type and vein pattern. While some blocks are left as they are and sold as a whole, most of them are cut into slabs by big machines. Before a great deal of the slabs is cut into tiles however, they are further processed; their surface is treated, depending on their later use. Most slabs that end up as tiles are being sanded and polished. Slabs too, are, if the customer wishes, sold as a whole or cut to a specific size. After the surface has been treated they are examined for cracks, flaws and other imperfections.

Recently, a system has been installed that automatically photographes all the slabs on the conveyor belt immediately after they leave the grinding and polishing machine using a line camera. In the future, the system should also be able to detect, identify and locate flaws and defects of any kind. The data collected in this manner will then be, amongst other data, loaded into the data warehouse in the context of the ETL process.

As one of the last steps, flawed slabs and blocks are processed to gravel and crushed rock and solts intended for processing to tiles are cut into tiles of varying size. Eventually a part of the tiles are treated with resin.

All products are then sorted and moved into the warehouse.

1.1 Business Process Modelling

I proposed the following business processes to be modeled:

Extraction Each block extracted from the querry is recorded, along with important measures such as its dimensions, weight and type. The date of the extraction, the gallery, the height and depth at which the block has been extracted, the date of transportation to the storehouse is stored, as well as the machinery used and the workers involved; the latter two are arguably mock and arificial and have been added mostly to make this example more convoluted and bump the number of dimensions over the required minimum.

Sale Each sale is registered, along with the date of purchase and a detailed customer profile.

Inventory Each product in the warehouse is recorded, together with details such as its dimensions, pattern etc. Additionally, the date of storage, quantity and possible reservations by customers are noted.

1.2 Dimensions and Bus Matrix

Before we can set up a Bus Matrix we need to consider necessary dimensions:

Date Dimension An identical conformed dimension that is shared across all business process fact tables, as can be easily be spotted on the Bus Matrix below. Granularity is discussed at a later point.

Product Dimension The product dimension is by far the biggest of the seven dimensions. It contains all products, that is blocks, slabs and tiles with all their myriad different properties. The product dimension is shared and among different fact tables and mostly identical. It contains the conformed surface dimension.

Surface Dimension This dimension contains information about the quality of the stone's surface. That is its type (e.g. *Black and White* or *Vena d'Oro*) and details about the vein patterns such as color, intensity, angle, thickness and cloudyness.

Customer Dimension This dimension represents a customer profile, that is name, addresses, and other details. It is shared and identical.

Worker Dimension Lists all workers and operators of the company and is used to record who was involved in specific processes. This dimension is only used in the EXTRACTION fact.

Machine Dimension Similarly, this dimension is a list of all the machines the company owns. It allowes to find out which machines were used for a given business process. This dimension is only used in the EXTRACTION fact.

Gallery Dimension A simple and streightforward dimension. A quarry usually (and in this case too) has multiple galleries. It is generally interesting to know the originating gallery for a given product. This dimension is only used in the EXTRACTION fact.

Position Dimension A simple dimensions that is mostly comprised of positional tuples $(depth, height)$: each gallery is divided into floors (called "bancata" by the specialist).Further, a block is excavated at a specific depth (called "avanzamento" by the specialists). We will simply call them *depth* and *height*. This dimension is only used in the EXTRACTION fact.

Dimensions are discussed with more detail in section 2.

Considerations about the Gallery and Position Dimensions The *gallery* and *position* dimensions could be unified to a single new *position* dimension, possibly with the following functional dependencies defined upon it: $position \rightarrow gallery, position \rightarrow depth, position \rightarrow height$. But the company is mostly interested in restricting queries to galleries. Thus, in order to simplify queries they were separated. Of course, it is always nice too, to get a dimension for free when there is a minumum number of dimensions required.

Table 1 shows a possible bus matrix for this data warehouse.

	Date	**Product**	**Surface**	**Customer**	**Worker**	**Machine**	**Gallery**	**Position**
Extraction	✗		✗		✗	✗	✗	✗
Sale	✗	✗	✗	✗				
Inventory	✗	✗	✗	✗				

Table 1: Simple and straightforward,
the Bus Matrix for the planned data warehouse.

1.3 Queries

Table 2 lists possible questions the company would like to have answered.

EXTRACTION	1. What is the average amount of stone (in tons) extracted per month per gallery ? 2. What is the average volume of stone (in m^3) extracted per month per gallery ? 3. What is the average volume of stone (in tons) extracted per month per gallery per worker ? 4. What is the average volume of stone (in tons) extracted per month per gallery per type ? 5. What is the gallery from which the most amount of *Vena'd Oro* (murble of a specific type) has been extracted ? 6. What is the gallery from which the most amount (volume) of flawed stone has been extracted ? 7. What is the longest time a block has been exposed directly to the effects of weathering per month (i.e. the time between the extraction and transportation from the quarry to the storehouse) ?
SALE	1. What are our best customers ? 2. Where are our best customers ? 3. What is the best selling stone type (e.g. *Venosta*) by continent ? 4. What are the trends in time in terms of stone type that customers want ? 5. To which country is most of the stone exported by mass or volume ? 6. In what country is selling stone most profitable, possibly broken down by year ?

INVENTARY	1. How much stone is stored in our warehouses ? 2. How much stone is added on average to the warehouses per month or year, or even day ? 3. How much stone is removed on average (hopefully sold) from the warehouses per month or year ? 4. Break down the inventory by product type and express as percentages, to give e.g. answer to the question: How much percent of the stone in the stock are blocks, tiles etc. ? 5. Refine the previous query and add stone type or vein patterns. That is, to what products, in percentages, has the stone of a certain type been processed ? 6. Is there a recognizable trend that we will be out of stock for a specific product in the long future.

Table 2: A few questions that the CTO of the *Covelano Murble Spa.* would like to have answered.

It is obvious that the queries for the SALE fact can perfeclty be matched with the queries for the EXTRACTION. If the most successful stone type is *Silvergold*, concentrate on the gallery that yielded most stone of that type as it is likely that there is more there. If the company gets a large order for a specific stone type, or even vein pattern that cannot be satisfied from the stock they have at least a clue where to dig for it.

1.4 Relevant Measures, Granularity and Additivity

In the following, the appropriate fact granularity and measures are discussed to at least be able to answer the queries above. Additional measures and attributes will be added in the conceptual design phase.

As far as the EXTRACTION fact is concerned,

- The hierachy built on the *surface* dimension must include at least a *type* attribute.
- The *date* hierachy must include at least the *day*, *month* and *year* attributes.
- The amount is caluclated by aggregating the *weight* measure along any dimension. It is fully additive. It is measured in kilograms.

- The *volume* measure is added when aggregating every dimension and is calculated indirectly from the *height*, *width* and *diameter* measures.
- The *flawed volume* measure is indirectly calculated by summing up the products of the *flawed* measure, which records how much percent of an extracted block had flaws of any kind, and the *volume* measure. It is additive and measured in m^3, while the *flawed* measure itself is non-additive.
- The *height*, *width* and *diameter* measures are by themselves non-additive along any dimension (except maybe for things like "all the blocks we extracted stacked on top of each other extend to the moon and back" etc.). They can, however, be aggregated using $AVG()$, $MIN()$ and $MAX()$ along any dimension.

As far as the SALE fact is concerned,

- The hierachy built on the *customer* dimension must include at least an address attribute, country and continent attributes.
- The hierachy built on the *product* dimension must include at least a *surface type* attribute.
- The date hierachy must include at least the day, month and year attributes.
- The *quantity* measure is added when aggregating every dimension, i.e. is additive.
- The *profit* measure is added when aggregating every dimension, i.e. is additive.

As far as the INVENTORY fact is concerned,

- The *Inventory Periodic Snapshot* model is used.
- The *quantity* measure is added when aggregating along the product and customer dimension. It is non-additive for the *date* dimension.

2 Conceptual Design

The structre of the facts and dimensions has already been briefly alluded to and partly anticipated in the previous chapter. Here, we will succintly but in detail discuss specific aspects of the modelling, mostly with reference to the *dimensional fact model*, herafter referred simply as DFM.

The notation used for the DFM models on the following pages mostly follows the notation outlined in [Riz08] and [GR09], although slightly different notations are used there.

It holds true for dimensions, and facts too, that only an arbitarily selected subset of attributes and measures will be depicted, and that conversely a number of them have been omitted, as the author is thoroughly convinced that there lies no didactic value within complicating this example and the corresponding diagrams.

2.1 The Extraction Fact

The EXTRACTION fact (Figure 1) models the fact already mentioned in the analysis section with its measures and dimensions. The *date* dimension is shared for the roles *transport* (date of transportation) and *excavate* (date of excavation from the quarry), which can be, but not necessarily are, the same date.

Since possibly multiple workers and machines are involved in the extraction process, a many-to-many association exists between those dimensions and the fact, denoted by the double line. These "multiple arcs" generally mean the loss of the summarizability property, which can, however, be restored by weighting [LS97]. Since we assume that each worker contributes the same effort a weight is not necessary, at least as far as workers are concerned. Generally, these kind of situations are to be avoided, for they are highly problematic when it comes to mapping the conceptual schema to a logical or physical design. This is especially true for *ROLAP*. Fair enough, these kind of situations seem to appear not all that rare and I thinks there should be better ways to deal with them; maybe array or set types, that some of the more modern relational DBMS support (e.g. *Postgres*, which supports array types) could remedy this problem. In our case, multiple arcs were added to make this otherwise rather dull example a little more exciting.

I slightly deviate from the guideline in [Riz08]. Instead of marking non-additivity with a dashed line, the possible aggreates are listed in parenthesis behind the measures so as not to clutter and make unreadable the diagram.

The *opening* (i.e. when they first started digging in that gallery) descriptive attribute for the *gallery* and the *last revision* (e.g. security checks) attribute for the *machine* dimension were mostly added to show off notation and to meet the appropriate requirements and are arguable.

Additionaly attributes could be added to any of the dimensions, specifically to the worker and machine dimensions, but they would be circumstantial, needlessly complicate everything and consume time spent better elsewhere.

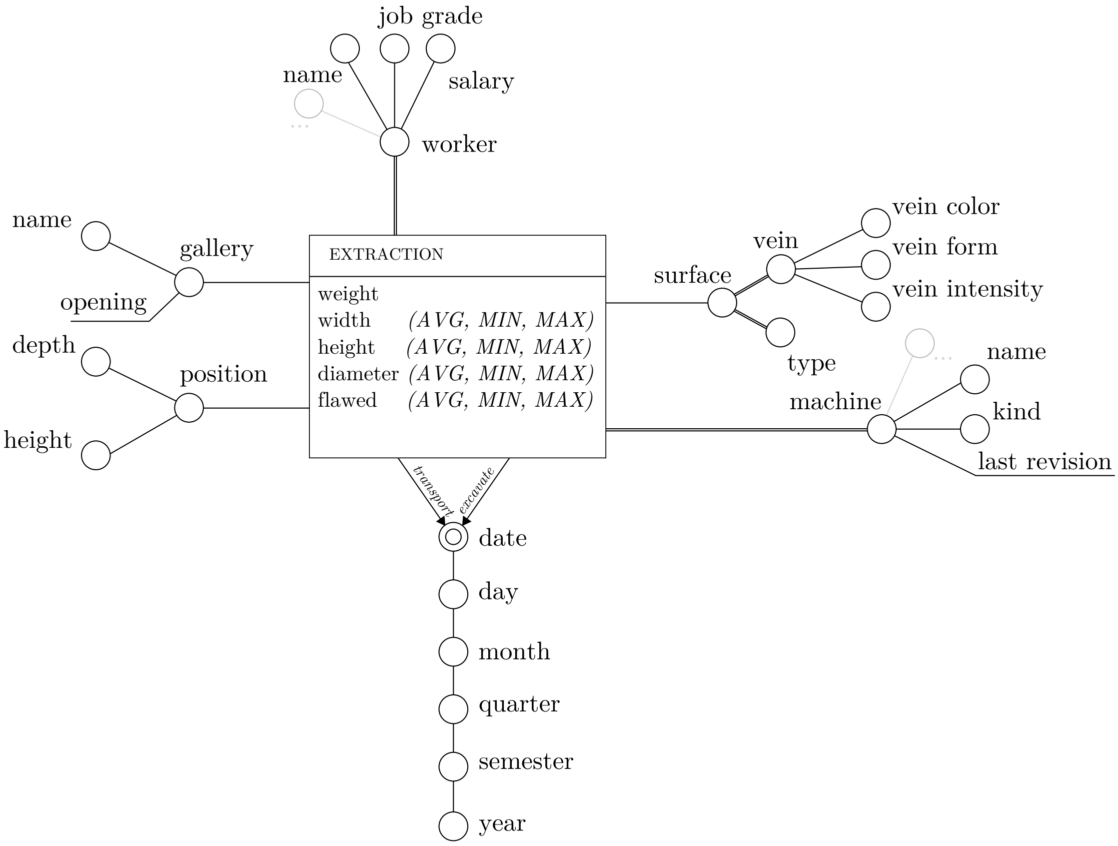

Figure 1: The *DFM* for the EXTRACTION fact

2.2 The Sale Fact

The *customer* dimension is incomplete (or ragged) for the city branch, as different nations have different ways of dividing the country into administrative units. Large countries, such as the U.S. have a very thorough subdivision, while e.g. a microstate in constrast is very flat in this regard. There is convergence between the *sales district* on one and the *city*, *country*, *country* attributes on the other side, for they both determine the *continent* attribute. Only the most important attributes for the customer dimensions are depicted, in real world additional attributes might be added.

The surface dimension has two multiple arcs for *vein* and *type*. This is due to the fact that a product (mostly blocks, but occasionally also slabs) can have patterns of two, three or even more types. Likewise, it might have veins of different colors, intensity etc.

The *type* attribute does only weakly functionally determine the vein atrribute.

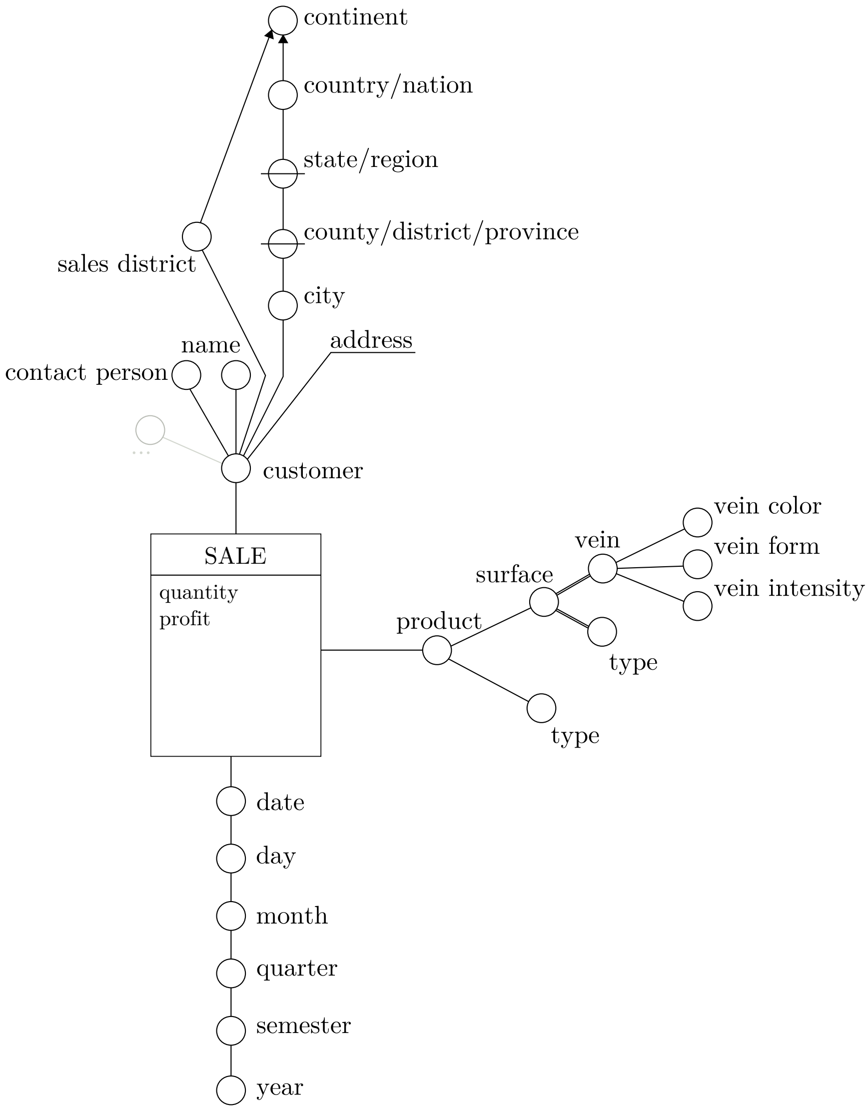

Figure 2: The *DFM* for the SALE fact

2.3 The Inventory Fact

There isn't much left to say about the INVENTORY fact, for it strongly resembles the *sale* fact (which is a bit worrying in terms of example chosen).

The fact has two shared date dimensions, that represent the time a product has been stored and reserved, respectively.

The quantity measure is non-additive with respect to the *date* dimension as far as the *store* role is concerned.

The *date* dimension in the *reserve* role is optional (i.e. an optional arc) since not all products are reserved.

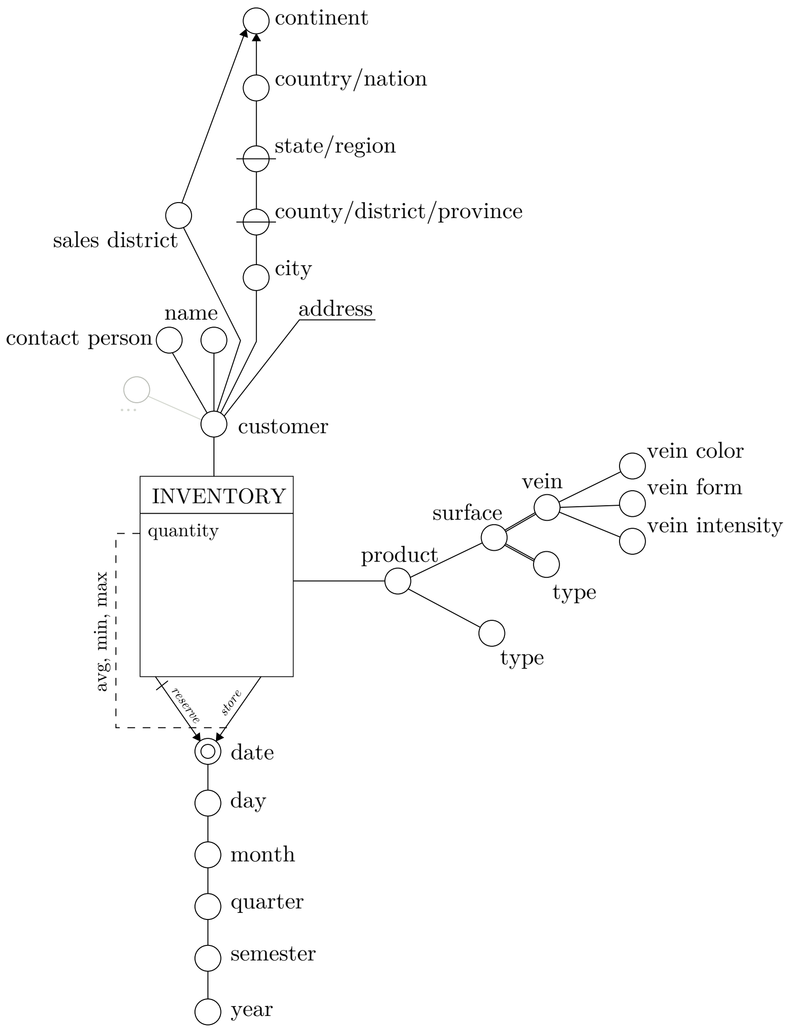

Figure 3: The *DFM* for the INVENTORY fact

3 Logical Design

We "resist normalization urges" [KR02] and limit ourselves to star schemas (and so escape the pains of drawing snowflake schemas). In the following, the star schemas for our simple examples are depicted. The abbreviations (FK) and (PK) after the measures and attributes stand for *foreign key* and *primary key*, respectively.

3.1 Mapping Multiple Arcs

Except for the dimensions with multiple arcs (*machine*, *worker*, *product*, *surface*, logical integration is streighforward. As for those others, I decided to use bridge tables. Initially I did consider "inlining" by using arrays or *hstore* (*Postgres*) in the later physical design step. However, it is very hard to maintain summarizability using e.g hstore because support for range queries on hstore values for specific keys is fairly limited. And so it happened that bridge tables were used for all multiple arcs relations. To keep summarizability, the bridge tables for the surface type and veins need a weight field. The machine and worker bridge tables do not; for it is assumed that weights are evenly distributed amongst workers and machines.

3.2 Snowflake against will

Although a star schema was chosen, the schemas depicted below look suspiciously like a snowflake schema. This is due to the bride tables, that were used to map multiple arcs. This is, of course, not without consequence. Consider a query that is only interested in the vein color attribute, it cannot be carried out without a join, possibly inhibiting performance. In theory, multiple arcs could be "unfold". This would considerably increase the size of the dimension tables, which would however, considering that most of the data is kept in the fact tables anyway, be negligible. Unfortunately, there is, as far as the author is aware, no support for this in any DBMS.

3.3 The time dimension anomaly

It seems to be usual to model the time dimension linearly in the conceptual design; that means we have for instance a functional dependency $day \rightarrow month$, which is of course utter nonsense unless one would e.g. store the days as Julian Day Numbers. Then still, however, the indirect functional dependency $day \rightarrow year$ would be invalid. This could be solved by storing days since epoch (e.g. days since January 1970), but the day field should be easily readable. Now, after having it done the wrong way, I would break up the usual dependencies and make the bottom *date* attribute fully determine all the other attributes in the dimension. Currently, however, the *day* attribute has been logically modelled to use Julian Day Numbers.

3.4 The Extraction Fact

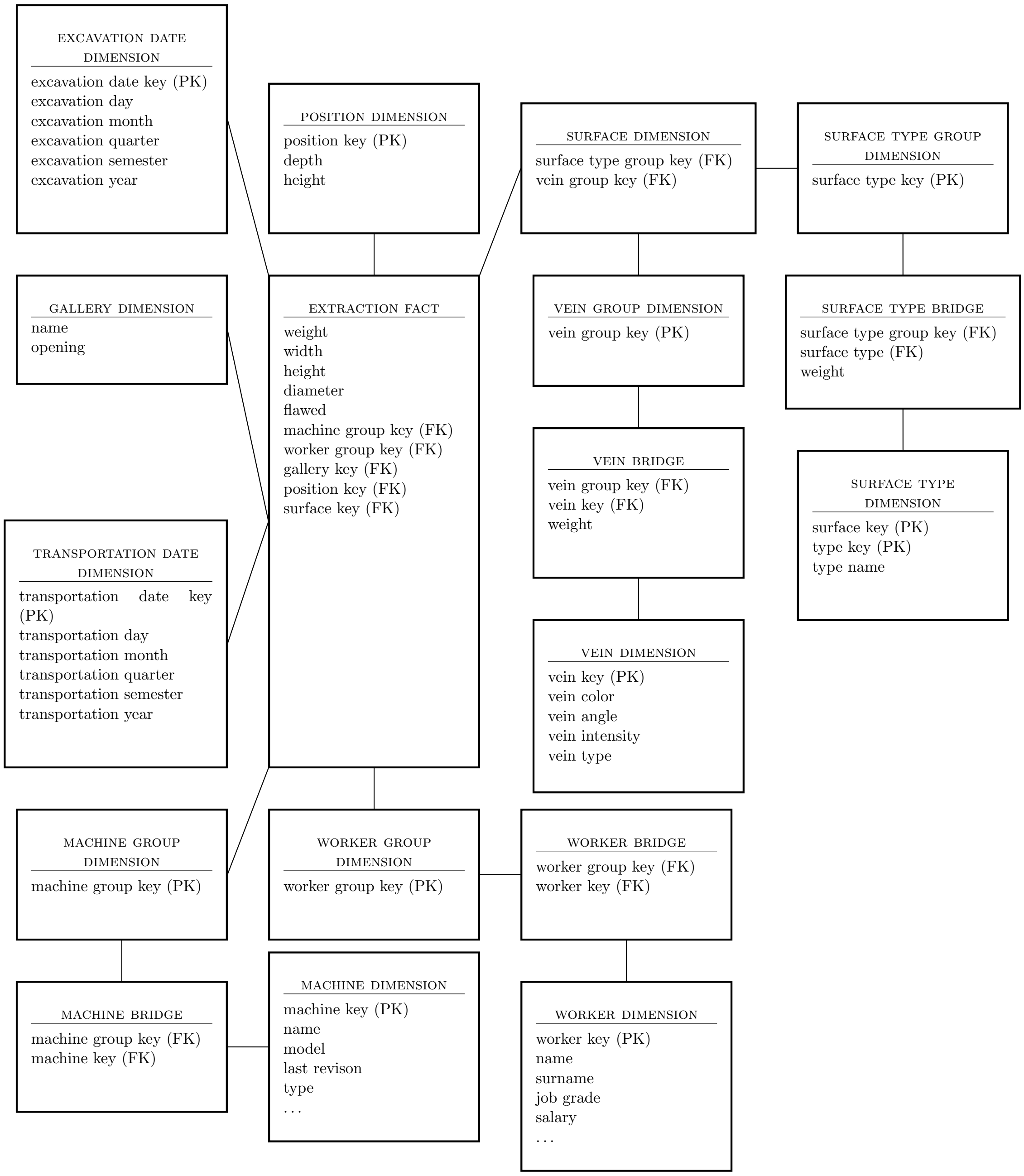

Figure 4: The logical schema for the EXTRACTION fact along with its dimensions.

3.5 The Sale Fact

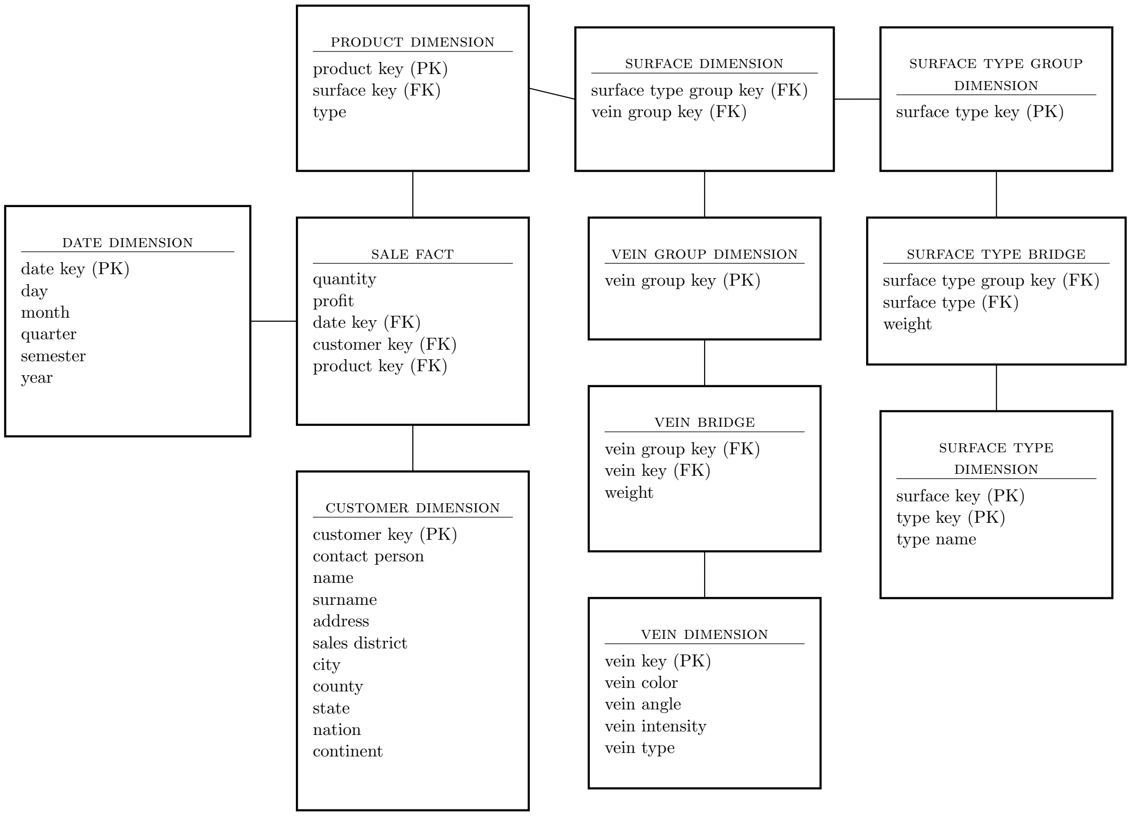

Figure 5: The logical schema for the simpler SALES fact.

3.6 The Inventory Fact

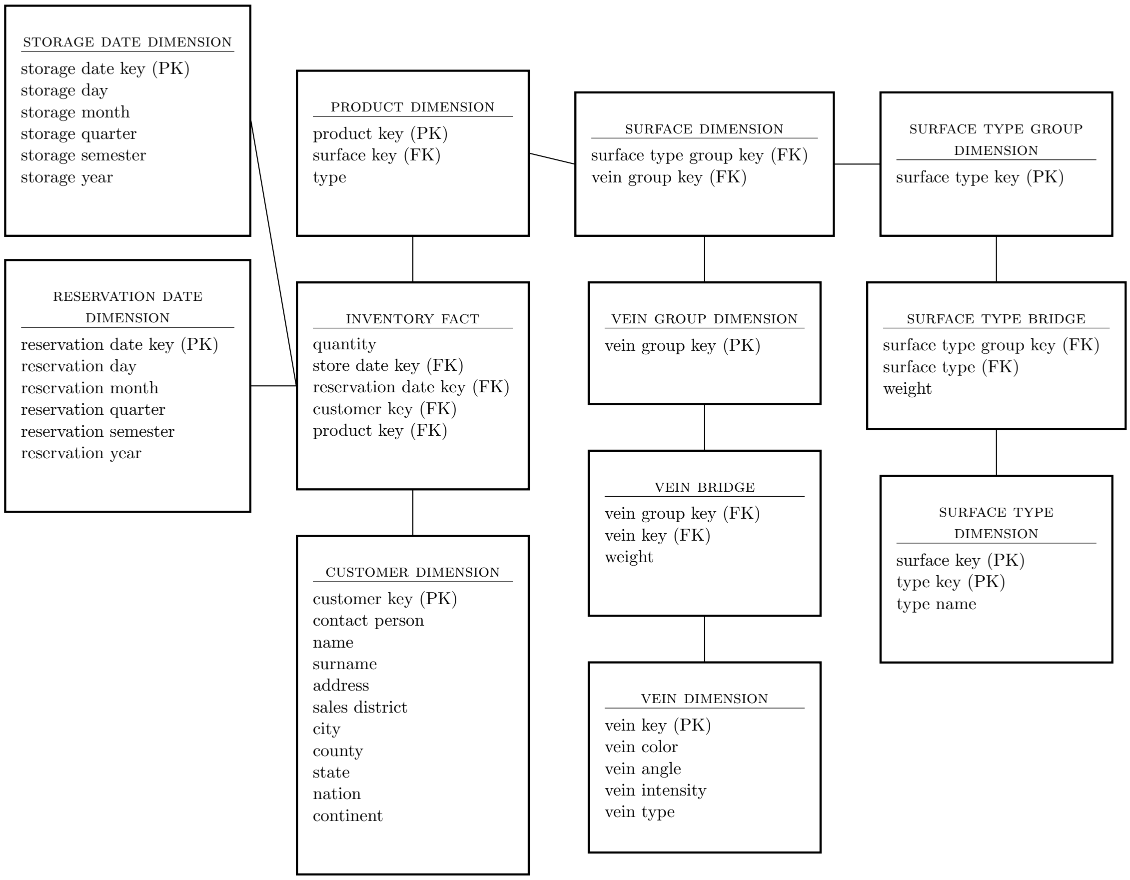

Figure 6: The logical schema for the INVENTORY fact, all too similar to the *sale* fact.

3.7 Simple Queries

We give queries to answer two simple business questions:

3.7.1 Extraction Fact

What is the total amount of stone extracted (volume and mass) in the year 2008 for each gallery ?

```
SELECT gallery.name,
       date.year,
       SUM(height * width * diameter) AS volume,
```

```
       SUM(weight) AS mass
FROM extraction, date, gallery
WHERE extraction.excavation_date_id = date.id
AND   extraction.gallery_id         = gallery.id
AND   date.year = 2008
GROUP BY date.year, gallery.name
```

id	**date**	**year**	**semester**	**quartal**	**month**	**day**
0	2007-11-26	2007	2	4	NOV	330
1	2008-03-08	2008	1	2	MAR	068
2	2008-04-26	2008	1	2	APR	117
3	2008-05-07	2008	1	2	MAY	128
4	2009-01-25	2009	1	1	JAN	025
5	2009-10-01	2009	2	4	OCT	274
6	2010-01-04	2010	1	1	JAN	004
7	2010-05-03	2010	1	2	MAY	123
8	2010-08-10	2010	2	3	AUG	222
9	2010-09-24	2010	2	4	SEP	267
10	2011-11-01	2011	2	4	NOV	305

Table 3: A sample *date* dimension table – all data is fictional

id	**name**	**opening**
0	G1	2007-07-22
1	G2	2008-03-31
2	G3	2008-04-14
3	G4	2008-07-08
4	G5	2009-03-04
5	G6	2009-04-06
6	G7	2011-08-02
7	G8	2011-06-02
8	G9	2011-07-03
9	G10	2012-04-01
10	G11	2012-05-10

Table 4: A sample *gallery* dimension table – opening dates are fictional

excavation_date_id	transportation_date_id	gallery_id	position_id	machine_group_id	worker_group_id	surface_id	width	height	diameter	weight
9	7	9	16	0	2	14	2.89	1.25	1.40	13.97
7	9	7	9	3	1	24	2.68	1.26	1.17	11.00
5	8	9	5	2	3	9	2.93	1.25	1.22	12.41
3	3	9	14	2	4	3	2.66	1.49	1.39	15.39
1	3	5	9	3	4	28	2.79	1.54	1.34	15.94
5	4	2	16	2	4	15	2.66	1.49	1.42	15.63
6	3	1	5	1	4	21	2.71	1.34	1.16	11.70
9	4	7	4	0	4	5	2.68	1.27	1.32	12.47
8	7	8	3	2	3	2	2.75	1.45	1.20	13.27
6	7	7	14	3	1	13	2.85	1.40	1.34	14.91

Table 5: A sample *extraction* fact table – all data is fictional, although realistic. Width, height and diameter is given in meters, weight in tons.

Year	Name	Volume	Mass
2008	G10	5.598	15.40
	G6	5.798	15.94

Table 6: The result set for the query above.

3.7.2 Sale Fact

What is the profit made on the north-american market for the last three years ?

id	first_name	last_name	address	sales_district	city	county	state	country	continent
0	Laila	Fend	57 St Georges Hill	ANGLO	Crawcrook and Greenside Ward	Tyne and Wear	NULL	UK	EUROPE
1	Corrinne	Jaret	2150 Morley St	ANGLO	Dee Ward	Dumfries and Galloway	NULL	UK	EUROPE
2	Margurite	Loperfido	218 Greenbank Drive	ANGLO	Devizes	Wiltshire	NULL	UK	EUROPE
3	Pok	Molaison	211 Hobart St	ANGLO	Newquay	Cornwall	NULL	UK	EUROPE
4	Ahmad	Alsaqri	21 Pickwick St	ANGLO	Sutton cum Duckmanton	Derbyshire	NULL	UK	EUROPE
5	Murray	Fode	59 W Century Rd	ANGLO	Pointe-Claire	NULL	QC	CA	NORTH AMERICA
6	Lavelle	Lillywhite	5 S Taylor Ave	ANGLO	La Malbaie	NULL	QC	CA	NORTH AMERICA
7	Truman	Mondale	1657 N Green St	ANGLO	Peterborough	NULL	ON	CA	NORTH AMERICA
8	Lea	Steinhaus	80 Maplewood Dr 34	ANGLO	Bradford	NULL	ON	CA	NORTH AMERICA
9	Olga	Adessa	8507 Upland St	ANGLO	Burlington	NULL	ON	CA	NORTH AMERICA
10	Stephaine	Barfield	47154 Whipple Ave Nw	ANGLO	Gardena	Los Angeles	CA	US	NORTH AMERICA
11	Fannie	Lungren	17 Us Highway 111	ANGLO	Round Rock	Williamson	TX	US	NORTH AMERICA
12	Kasandra	Semidey	369 Latham St 500	ANGLO	Saint Louis	Saint Louis City	MO	US	NORTH AMERICA
13	Carey	Dopico	87393 E Highland Rd	ANGLO	Indianapolis	Marion	IN	US	NORTH AMERICA
14	Yuki	Whobrey	1 State Route 27	ANGLO	Taylor	Wayne	MI	US	NORTH AMERICA
15	Isreal	Calizo	2 Landmeier Rd	ANGLO	Wombeyan Caves	NULL	NS	AU	AUSTRALIA
16	Coletta	Thro	64865 Main St	ANGLO	North Fremantle	NULL	WA	AU	AUSTRALIA
17	Carman	Robasciotti	4 Spinning Wheel Ln	ANGLO	Granya	NULL	VI	AU	AUSTRALIA
18	Terina	Wildeboer	462 Morris Ave	ANGLO	Seddon	NULL	VI	AU	AUSTRALIA
19	Leatha	Block	6926 Orange Ave	ANGLO	Two Rocks	NULL	WA	AU	AUSTRALIA

Table 7: A sample *customer* dimension table with english-speaking customers – all data is fictional

date_id	product_id	customer_id	quantity	profit
3	4	13	5	83227
1	1	5	371	79598
6	0	5	350	71346
4	4	6	264	66579
8	0	7	15	20067
9	1	3	110	60668
8	6	3	251	83514
0	6	4	99	83287
9	0	11	11	104433
8	4	5	68	56410

Table 8: A sample *sale* fact table – all data is fictional.

```
SELECT customer.continent, date.year, SUM(profit)
FROM   sale, date, customer
WHERE  sale.customer_id = customer.id
AND    sale.date_id = date.id
AND    date.year IN (2009, 2010)
AND    customer.continent = 'NORTH_AMERICA'
GROUP BY customer.continent, date.year
```

Continent	Year	Profit
North America	2009	66579
	2010	252256

Table 9: The result set for the query above.

4 Physical Design

4.1 Simple ROLAP Queries

```
SELECT date.year, date.semester, date.quartal,
       SUM(height * width * diameter) AS volume, SUM(weight) AS mass
FROM extraction, date
WHERE extraction.excavation_date_id = date.id
GROUP BY ROLLUP(date.year, date.semester, date.quartal)
```

A note about Postgres Unfortunately, *Postgres* has at this point in time no support for ROLAP queries; neither ROLLUP nor CUBE groupings are supported, which is surprising, considering that even MySQL has support (although very basic) for these kind of queries. What remains is the possibility to "paraphrase" ROLLUP queries by creating the union of multiple subqueries. The above query would look then as follows. Note that UNION ALL was used, as the UNION keyword defaults to UNION DISTICT, which would induce unnecessary processing (e.g. sorting). Obviously, this does not perform as well as a corresponding ROLLUP would. Also, controlling the order in which rows should appear seems to be tricky.

```
SELECT date.year as year,
       date.semester as semester,
       date.quartal as quartal,
       SUM(height * width * diameter) AS volume,
       SUM(weight) AS mass
FROM extraction, date
WHERE extraction.excavation_date_id = date.id
GROUP BY date.year, date.semester, date.quartal
UNION ALL
SELECT date.year,
       date.semester,
       NULL as "quartal",
       SUM(height * width * diameter) AS volume,
       SUM(weight) AS mass
FROM extraction, date
WHERE extraction.excavation_date_id = date.id
GROUP BY date.year, date.semester
UNION ALL
SELECT date.year,
       NULL AS "semester",
       NULL AS "quartal",
       SUM(height * width * diameter) AS volume,
       SUM(weight) AS mass
FROM extraction, date
WHERE extraction.excavation_date_id = date.id
GROUP BY date.year
UNION ALL
SELECT NULL AS year,
```

```
        NULL AS "semester",
        NULL AS "quartal",
        SUM(height * width * diameter) AS volume,
        SUM(weight) AS mass
FROM extraction, date
WHERE extraction.excavation_date_id = date.id
ORDER BY year, semester, quartal;
```

year	**semester**	**quartal**	**volume**	**mass**
2007	1	1	96.188185836	264.505
2007	1	2	225.807796467	620.974
2007	2	3	383.097466294	1053.507
2007	2	4	230.967756251	635.163
2007	1		321.995982303	885.479
2007	2		614.065222545	1688.670
2007			936.061204848	2574.149
2008	1	1	333.987827350	918.463
2008	1	2	275.250892176	756.923
2008	2	3	183.719783027	505.209
2008	2	4	237.817976417	653.965
2008	1		609.238719526	1675.386
2008	2		421.537759444	1159.174
2008			1030.776478970	2834.560
2009	1	1	217.306085913	597.578
2009	1	2	311.630122536	857.038
2009	2	3	242.391642842	666.621
2009	2	4	259.711394078	714.202
2009	1		528.936208449	1454.616
2009	2		502.103036920	1380.823
2009			1031.039245369	2835.439
2010	1	1	235.904361442	648.704
2010	1	2	239.787009053	659.440
2010	2	3	292.159687454	803.368
2010	2	4	163.936097149	450.822
2010	1		475.691370495	1308.144
2010	2		456.095784603	1254.190
2010			931.787155098	2562.334
2011	1	1	105.546086406	290.233
2011	1	2	241.993527613	665.492
2011	2	3	215.725887074	593.263
2011	2	4	280.632694755	771.739
2011	1		347.539614019	955.725
2011	2		496.358581829	1365.002
2011			843.898195848	2320.727
			4773.562280133	13127.209

Table 10: The result set for the query given above. Empty cells denote *NULL* values.

Again, this query cannot be run on Postgres. The use of the GROUPING function makes this query even harder to translate into something Postgres can understand.

```
SELECT CASE
       WHEN GROUPING(date.year)=1 THEN
         'all␣years'
       ELSE
         date.year
       END,
       CASE
       WHEN GROUPING(date.semester)=1 THEN
         'all␣semesters'
       ELSE
         date.semester
       END,
       CASE
       WHEN GROUPING(date.quartal)=1 THEN
         'all␣quartals'
       ELSE
         date.quartal
       END,
       SUM(height * width * diameter) AS volume,
       SUM(weight) AS mass
FROM extraction, date
WHERE extraction.excavation_date_id = date.id
GROUP BY ROLLUP(date.year, date.semester, date.quartal)
```

year	semester	quartal	volume	mass
2007	1	1	96.188185836	264.505
2007	1	2	225.807796467	620.974
2007	2	3	383.097466294	1053.507
2007	2	4	230.967756251	635.163
2007	1	all quartals	321.995982303	885.479
2007	2	all quartals	614.065222545	1688.670
2007	all semesters	all quartals	936.061204848	2574.149
		...		
all years	all semesters	all quartals	4773.562280133	13127.209

Table 11: The result set for the query given above. The *NULL* values have been replaced with a describtive value.

5 Advanced Querying

5.1 Ranking Query

In each sales district, financial resources (e.g for advertising) are distributed evenly across its countries; however, if sales in a country were good it gets an additional bonus. In order not to disadvantage any sales district, the bonuses depend only on how well the other countries in the same district did. The countries in a district are sorted by total profit, then the coutries in the top 30% get a "high" bonus, the following top 30% a "low" bonus and the remaining lower 30% get no bonus at all. The following query helps the management to assign boni to the countries in the sales district for the english-speaking countries. The best and worst profit sale is removed to expunge outliers and increase fairness.

Listing 1: For each country in the 'ANGLO' sales district, list the minimum and maximum profit, and its "bonus category".

```
SELECT customer.country,
       SUM(sale.profit),
       MIN(sale.profit),
       MAX(sale.profit),
       NTILE(3) OVER (
         ORDER BY (SUM(sale.profit) - MIN(sale.profit) - MAX(sale.profit))
       )
FROM sale, customer
WHERE sale.customer_id = customer.id
AND customer.sales_district = 'ANGLO'
GROUP BY customer.country;
```

country	total profit	minimum profit	maximum profit	bonus category
AU	13578425.000	10700.000	108674.000	1
CA	14576231.000	10001.000	109854.000	1
US	15718433.000	10348.000	109118.000	2
UK	15900180.000	10120.000	109702.000	3

Table 12: The result set for the query given above. The United Kingdom will get a "high" bonus, the U.S. a "low" bonus.

5.2 Windowing Query

Consider the query above. To distribute bonuses in a fair fashion, we removed the best and worst profit sale. The management decides that that's not enough and want's me to remove the best and worst 15% of the sales.

Listing 2: Same as above, but outliers are removed more reliably.

```
CREATE TEMPORARY VIEW sale_centile AS
SELECT customer.country,
       sale.profit,
       NTILE(100) OVER (
          PARTITION BY customer.country ORDER BY sale.profit
       ) AS centile
FROM sale, customer
WHERE sale.customer_id = customer.id
AND customer.sales_district = 'ANGLO';

SELECT customer.country,
       SUM(sale.profit),
       MIN(sale.profit),
       MAX(sale.profit),
       NTILE(3) OVER (ORDER BY (
          SUM(sale.profit) -
          (SELECT SUM(profit) FROM sale_centile WHERE centile < 15 OR centile > 85))
       )
FROM sale, customer
WHERE sale.customer_id = customer.id
AND customer.sales_district = 'ANGLO'
GROUP BY customer.country;
```

country	total profit	minimum profit	maximum profit	bonus category
AU	13578425.000	10700.000	108674.000	1
CA	14576231.000	10001.000	109854.000	1
US	15718433.000	10348.000	109118.000	2
UK	15900180.000	10120.000	109702.000	3

Table 13: The result set for the query given above. Nothing changed, because there seem to have been no outliers.

5.3 Period-to-Period Query

The *MARMOMACC* is the biggest international fair for operators in the marble sector. Ever since 2009 the *Covelano Marmi Srl.* is represented with a stand. It is by far the most important advertising event of the year. The management would like to know the impact of this event on the number of sales. Are sales notably increasing ? Is it worth putting much money and effort into this event ? We need to write a query that compares the number of sales made within 100 days before the event with the sales made 100 events after it.

Since each year *MARMOMACC* takes place at slightly different days we first create a temporary table of the begin and end dates of the fair for the last couple of years.

Year	Begin	End
2009	30-09-2009	03-10-2009
2010	29-09-2010	02-10-2010
2011	21-09-2011	24-09-2011
2012	26-09-2012	29-09-2012
2013	25-09-2013	28-09-2013

Listing 3: For each

```
CREATE TEMPORARY VIEW before AS
SELECT date.year as year, COUNT(*) AS count
                  FROM date, sale
                  WHERE sale.date_id = date.id
                  AND date.date BETWEEN (
                                           SELECT begin_date
                                           FROM marmomacc_dates
                                           WHERE year = date.year
                                         ) - 100
                                         AND
                                         (
                                           SELECT begin_date
                                           FROM marmomacc_dates
                                           WHERE year = date.year
                                         )
                  GROUP BY year;


CREATE TEMPORARY VIEW after AS
                  SELECT date.year as year, COUNT(*) AS count
                  FROM date, sale
                  WHERE sale.date_id = date.id
                  AND date.date BETWEEN (
                                           SELECT begin_date
                                           FROM marmomacc_dates
                                           WHERE year = date.year
```

```
                        )
                        AND
                        ((
                          SELECT begin_date
                          FROM marmomacc_dates
                          WHERE year = date.year
                        ) + 100)
            GROUP BY year;
```

```
SELECT DISTINCT date.year, before.count, after.count
FROM sale, date, before, after
WHERE sale.date_id = date.id
AND after.year = date.year
AND before.year = date.year;
```

Year	Sales before	Sales afterwards
2009	53	56
2010	53	36
2011	43	55
2012	50	40
2013	50	10

Table 14: The result set for the query given above. Our sample data does not reflect reality.

5.4 Materialized Views

We choose three queries that might be executed frequently. For the sake of simplicity, we reduce one-to-many to one-to-one relations, more specifically we assume that the extraction fact has a one-to-one sort dimension and an additional *vein angle* degenerate dimension that is stored in the fact table as a measure. Also, additional dimensions have been removed or simplified, see Figure 7

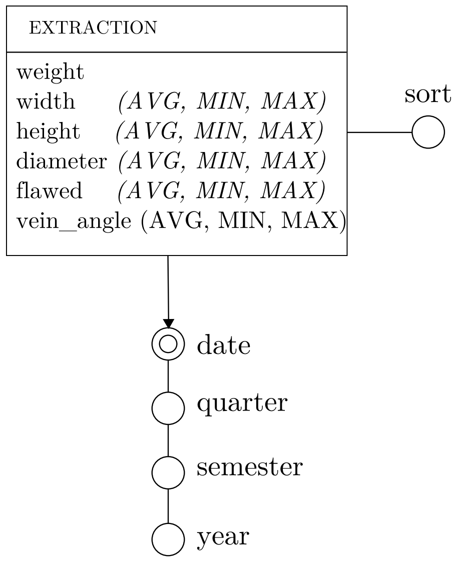

Figure 7: The simplified *DFM* for the EXTRACTION fact

1. What is the amount of stone excavated per gallery and year, semester and quartal. The query is similar to a previous one. This query takes ca. 14ms to execute with 10'000 tuples in the fact table. The grouping is (*year*).

```
SELECT date.year,
       SUM(weight) AS amount
FROM extraction, date
WHERE extraction.excavation_date_id = date.id
GROUP BY date.year
ORDER BY date.year;
```

2. What is the amount of stone excavated per sort and quartal (of a specific year) ? Executing this query is surprisingly fast; about 5 milliseconds for 10'000 tuples. The only grouping is (*quartal*, *sortname*).

```
SELECT date.quartal,
       sort.name as sort,
       SUM(extraction.weight) AS amount
FROM extraction, date, sort
WHERE extraction.excavation_date_id = date.id
AND        extraction.sort_id       = sort.id
AND                          date.year = 2009
GROUP BY date.quartal, sort.name
ORDER BY date.quartal, sort.name;
```

3. It important for the company to know the angles of the vein patterns because two different blocks can only be sold if angles more or less match, so when tiles are cut from the blocks there is a uniform pattern. Thus, the following query might be of interest: What is the amount of stone exacated per semester (of a specific

year) grouped by vein angle ? The vein angle is rounded to the nearest multiple of 10. Query execution time was around 5 milliseconds. The only grouping is (*semester*, *veinangle*).

```
SELECT date.semester,
       ceil(vein_angle / 10) * 10 AS angle,
       SUM(extraction.weight) AS amount
FROM extraction, date
WHERE extraction.excavation_date_id = date.id
AND                 date.year = 2009
GROUP BY date.semester, angle
ORDER BY date.semester, angle;
```

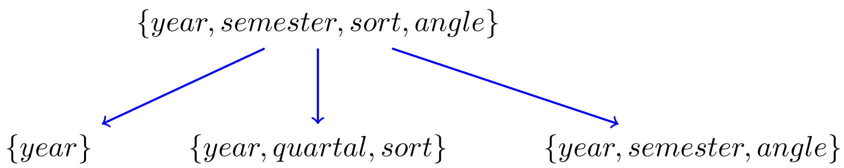

Figure 8: The lattice representing depedencies between sets of attributes that are suitable for materialization to improve the execution time of the queries above. Note that we have *quartal* > *semester*.

Getting Physical Fortunately, Postgres does support materialized views since version 9.3. The following piece of SQL creates the view and executing it takes over 200 milliseconds. The following view can be used by all three queries.

```
CREATE MATERIALIZED VIEW mv1 AS
SELECT date.year,
       date.quartal,
       sort.name,
       extraction.vein_angle,
       SUM(extraction.weight) AS amount
FROM extraction, date, sort
WHERE extraction.excavation_date_id = date.id
AND extraction.sort_id = sort.id
GROUP BY date.year, date.quartal, sort.name, extraction.vein_angle
ORDER BY date.year, date.quartal, sort.name;
```

We can rewrite the previous queries to make use of the materialized view. Unfortunately, Postgres does not support automatic query rewriting, so we do it by hand.

1.
```
SELECT year,
       SUM(amount)
FROM mv1
GROUP BY year
ORDER BY year;
```

2.
```
SELECT quartal,
       name as sort,
       SUM(amount)
FROM mv1
WHERE year = 2009
GROUP BY quartal, sort
ORDER BY quartal, sort;
```

3.
```
SELECT ceil(quartal / 2.0) AS semester,
       ceil(vein_angle / 10) * 10 AS angle,
       SUM(amount)
FROM mv1
WHERE year = 2009
GROUP BY semester, angle
ORDER BY semester, angle;
```

As Figure 8 shows, we might be able to reduce the query execution time further by creating additional materialized views. We will now create all materialized views and later discuss whether it was worth doing so. The following SQL code creates all different views.

```
CREATE MATERIALIZED VIEW mv2 AS
SELECT date.year,
       SUM(extraction.weight) AS amount
FROM extraction, date
WHERE extraction.excavation_date_id = date.id
GROUP BY date.year
ORDER BY date.year;

CREATE MATERIALIZED VIEW mv3 AS
SELECT date.year,
       date.quartal,
       sort.name,
       SUM(extraction.weight) AS amount
FROM extraction, date, sort
WHERE extraction.excavation_date_id = date.id
AND extraction.sort_id = sort.id
GROUP BY date.year, date.quartal, sort.name
ORDER BY date.year, date.quartal, sort.name;

CREATE MATERIALIZED VIEW mv4 AS
SELECT date.year,
       date.semester,
       extraction.vein_angle,
       SUM(extraction.weight) AS amount
FROM extraction, date, sort
WHERE extraction.excavation_date_id = date.id
GROUP BY date.year, date.semester, extraction.vein_angle
ORDER BY date.year, date.semester;
```

The following table shows the exeution time for the various query depending on whether they did make use of the materialized views or not or which view they used.

Query	No mat. view	Generic mat. view	Specific mat. view
Query 1	13	5	0.4
Query 2	5	2	0.7
Query 3	5	4	1.6

Table 15: Comparison of the execution times (in ms.) for the example queries on different materialized views. The generic materialized view is the view that all queries can take advantage of, the specific materilazed view is the view that only this specific query can make use of.

Mat. View	Size
mv1 (Generic)	7621
mv2 (for Query 1)	7
mv3 (for Query 2)	196
mv4 (for Query 3)	2438

Table 16: Comparison of the materialzed view sizes.

Comparing result with theoretic expectations Given the table sizes in Table 16 we can analyze the experimental data. It' clear that the greedy algorithms presented in the lectures would choose *mv2* first, then *mv3*, finally *mv4*. In practive, however, it's worth considering creating *m4* instead of *m3*, given that the improvement in execution time for the 3rd query is almost as good as for the 2nd query; this would depend on how often the respective queries are executed. Generally, execution times are farily short, given the relatively small amount of data. In this specific scenario it might be perfectly fine to not create any materialized views at all.

References

[GR09] Matteo Golfarelli and Stefano Rizzi. *Data Warehouse Design: Modern Principles and Methodologies.* McGraw-Hill, Inc., New York, NY, USA, 1 edition, 2009.

[KR02] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling.* John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 2002.

[LS97] Hans-Joachim Lenz and Arie Shoshani. Summarizability in olap and statistical data bases. In *Proceedings of the Ninth International Conference on Scientific and Statistical Database Management*, SSDBM '97, pages 132–143, Washington, DC, USA, 1997. IEEE Computer Society.

[Riz08] Stefano Rizzi. *Conceptual Modeling Solutions for the Data Warehouse.* 2008.