Design and Usage Example of a Data Warehouse in the Context of the *DigiEcoQuarry* Project

César Pérez
Sigma Technologies, Madrid
cperez@sigma-ai.com

Javier Gavilanes
Sigma Technologies, Madrid
jgavilanes@sigma-ai.com

Pierre Plaza
Sigma Technologies, Madrid
pplaza@sigma-ai.com

August 2022

Abstract

It is not always obvious the role that a data warehouses plays in many projects. It often seems that it solves the same problem as an operational database. In the context of the *DigiEcoQuarry* project, is not obvious how to implement a data warehouse and the role it plays to answer the business questions used to run a quarry. In this document, we provide an example use-case that aims to illustrate how a data warehouse fits in the project and how it is designed. As a result, we show how a simple data warehouse is designed and how to write an SQL query to answer a hypothetical business question. This document justifies why the data warehouse is needed in the *DigiEcoQuarry* project and illustrates how it is intended to be used in the context of the project.

Keywords: *DigiEcoQuarry, Data Warehouse, Data Warehouse Design Methodology*.

1. Background

The main purpose of a data warehouse is to provide answers to business questions based on data extracted from a data storage as explained in (Kimball & Ross, 2013). Data is normally extracted using SQL queries that are closely related to the business questions that need to be answered.

In contrast to a traditional database, information is stored in a way that matches the point of view of the information used to measure the business, instead of the point of view of the operations that run the business. Therefore, the data warehouse requires the information to be organized in a way that eases building queries to answer the business questions.

The design method described in (Kimball & Ross, 2013) relies on the business questions that need to be answered in order to make decisions that affect the business. In this document we focus on the following example question in order to design the data warehouse structure and to build the query that answer the question:

What is the average volume of each pile of aggregate by day in 2022's Q1?

The following sections discuss the references that support our methodology, the design of the tables of the data warehouse, and the SQL queries built to answer the questions.

2. Related work

The point-of-view and the design philosophy used in this document follows the one proposed in (Kimball & Ross, 2013). As far as we know this is a well stablished reference due to the large number of times this work has been cited on Google Scholar[1].

3. Design of the data warehouse

In this section we design the part of the data warehouse that provides the information used to answer the proposed questions. For this design, we are following the steps that are defined with plenty of examples in (Kimball & Ross, 2013):

1. **What is the business process we aim to model?**
   To be able to answer questions about the business we first need to know what is the process we are interested in. For example, in the sales context we may be interested in modelling the customer experience so, this way, the questions we are formulating will be related to that topic.

2. **What is the granularity[2] of the data we want to model?**
   Choosing the right granularity for the data we are considering in business process modelling determines which level of detail we can provide when answering business questions. For example, we can store daily information about the orders placed by a customer but that will make impossible to get information aggregated by hour; we have lost hourly information after the monthly aggregation!

3. **What are the dimensions of the questions to be answered?**
   Answering business questions usually requires to perform some level of aggregation. For example, one may want to know the total sales in a period aggregated by day or by month. Dimensions in a data warehouse are used to support these aggregations. In the provided example, the dimension is time and it supports performing queries by day and by month.

4. **What are the facts that result from the business process?**
   If the dimensions of the data warehouse support the aggregations of the queries, facts provide the actual information that answers the formulated questions. For example, in the previous sales context, one may want to know the number of sales by day in the first quarter of 2021. The number of sales is a value that could be found in the facts table.

To apply the previous steps to the design of the data warehouse in the *DigiEcoQuarry* project, we need to have a minimal understanding of the business we are modeling. For this example, we are considering a limited scenario where a simplified version of a service (based on an actual service that will be implemented in the project that uses AI and computer vision to calculate the volume of different stockpiles) is deployed. We are considering only one quarry that produces three types of aggregates (we are calling them *type A*, *type B*, and *type C*). When the day is over, the stockpile service we are using calculates the volume of the piles corresponding to each type of aggregate and updates the records in a storage system that hosts the volume of aggregates in each pile.

Using this simple scenario, the following sections provide the interpretation of the proposed steps according to the requirements imposed by the example question included in section 1.

[1] At the time of writing this document the second edition, from 2011, has been cited 6444 times.

[2] The term "granularity" is used instead of following the Kimball terminology, which uses "grain size". This is to avoid confusion since, in a quarry environment, which is the context of the *DigiEcoQuarry* project, grain size usually refers to the aggregates' sizes that are produced in the quarries.

What is the business process we aim to model?

Our example scenario is so simple so the business process can be directly extracted from the business question formulated in section 1. Just to make explicit the business process we are referring to, our focus is on the activities related to the stock of the aggregate piles. That is, we are interested in the current volume of the pile and in its increase or decrease rate.

What is the granularity of the data we want to model?

As a rule of thumb, the granularity of the facts in a data warehouse should include as much detail as possible to avoid not to answer a business question because it requires more detail as the data we are storing. The stockpile service implemented in this scenario provides volume calculations in a daily basis so that should be the granularity of the facts table used to model our business process.

There are two reasons that justify this granularity. First, we could not choose a finer grain, like volumes per hour, just because the service does not work that way! Second, we may want to store the volumes by week because we currently are running our business that way. However, that makes questions as the proposed to be impossible to answer on different circumstances.

We only considered time as the factor that determines the granularity of the data we are storing. This point of view may actually be right if we only consider time as the aggregation criterion. However, the question we aim to answer does not only consider time but also takes the type of aggregate into account. In general, we will want to consider all the aggregation and filtering criteria to choose the right granularity for our data warehouse.

In conclusion, in order to determine the granularity in this example we consider storing the volume values by hours for each type of aggregate we are producing.

What are the dimensions of the questions to be answered?

The business question we formulated in section 1 can be generalized as ***"What is the `[MEASURE]` of `[ENTITY]` by `[AGGREGATION]` in `[FILTER]`?"***. From each placeholder we used in that question reformulation we can extract information about the limits we are imposing to the desired `[MEASURE]`. Concretely, `[ENTITY]` tells us how to shape the resulting data, `[AGGREGATION]` provides the criterion we are using to join the data we want the results calculated for, and `[FILTER]` informs we are not interested in all data but only in those values that meet the filtering criteria.

The dimensions of our data are extracted from these placeholders. The last two placeholders, `[AGGREGATION]` and `[FILTER]` are related to the time when data makes sense. Since we are interested in days, quarters, and years we know the time dimension we need to consider those three attributes. From the `[ENTITY]` placeholder we get that it seems important (since we are using that information to came up with some groups) the type of aggregate we are storing in our piles.

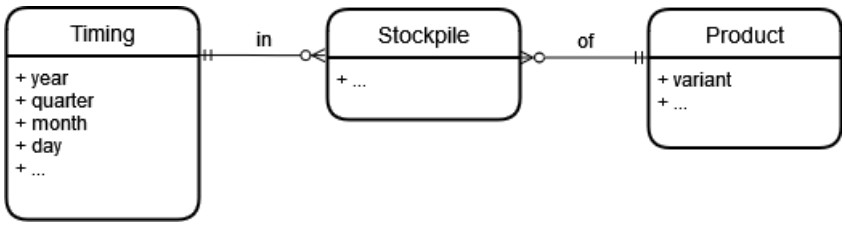

Figure 1. Detail of the dimension entities

Figure 1 shows the relationship between the dimensions we are using to structure our data and the facts (in this example we called "Stockpile" to the facts we are considering) that are the target of the business question. The "in" relationship between *Timing* and *Stockpile* means a single

stockpile entity is associated with one and only one timing entity and single timing entity can be associated with many stockpile entities (if it is associated with any of them). For the *Stockpile* and the *Product* entities the relationship is quite similar; a stockpile entity is associated with only one product and a single product may be associated to many stockpile entities.

What are the facts that result from the business process?

Our facts table needs to contain the information used to fulfil the proposed business question. In this example, the result for each question should be the volume of the pile for each aggregate so the facts table, in order to answer the question, needs at least to contain a number that represents the volume of each pile.

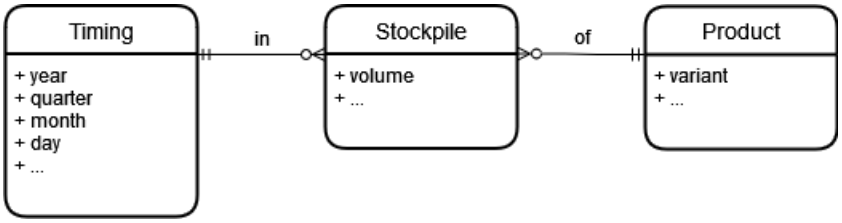

Figure 2. Detail of the relationship between facts and dimensions

Figure 2 shows the attributes we need to store about the `Stockpile` entity. We also included the relationships with the dimension entities to show the complete entity model of this data warehouse example.

The data model resulting from the design process

The entity model we built in the previous section allows us to understand what kind of information we are interested in and the relations between its components. However, we cannot get any answer from it. That conceptual model should be translated into a logical model that we can implement using a data warehouse solution.

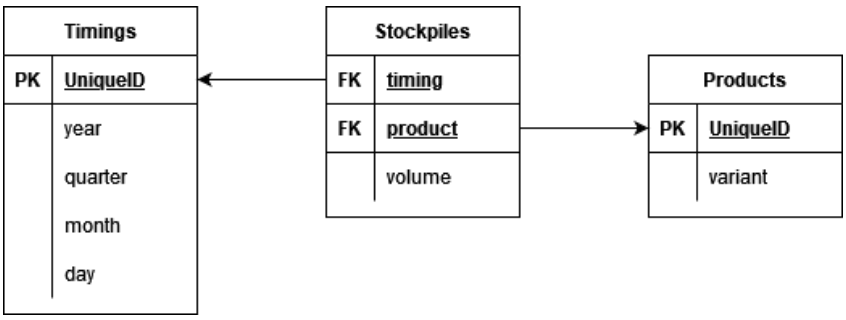

Figure 3. Tables resulting from the entity model

Figure 3 shows the logical model used to store the information we are interested in into a collection of tables we can create and manage using the SQL language. The details of how this database is created are included in appendix A.

4. Usage for the data warehouse

In section 3, we created conceptual and logical models to build a data warehouse that supports the information we require to model our business process. We are now ready to build the SQL query that will provide the answer to the example question included in section 1.

```sql
SELECT Products.variant AS `Aggregate`,
       Timings.day AS `Day`,
       AVG(Stockpiles.volume) AS `Volume`

FROM Volumes, Products, Timings

WHERE product = Products.id AND timing = Timings.id

  AND year = '2022'
  AND quarter = 'Q1'

GROUP BY `Aggregate`, `Day`

ORDER BY `Aggregate` ASC;
```

Figure 4. SQL query used to get the answer to the business question

Figure 4 shows the SQL query used to answer the proposed business question. The query can be decomposed into different parts that refer to the original question. The fields included in the **`SELECT`** statement extract just the interesting values **`FROM`** the tables that contain the information used to answer the business question. Using the **`WHERE`** statement we are filtering by the selected year and day to avoid considering information from other periods. As final steps, we are using the **`GROUP BY`** statement to aggregate the volume values by the correct criteria and then we use the **`ORDER BY`** statement to sort the results by the type of the aggregates the quarry produces.

5. Conclusions

In this document we provided a simple example about how the data warehouse fits in answering business questions from a concrete AI service from the *DigiEcoQuarry*. Despite the fact that the real service may not be exactly like the one used as an example, the design process and the usage of the data warehouse remain the same. We have shown how the design of the data warehouse and the implementation of the queries used to answer business questions are guided by the information we need to drive the business.

As we have seen, the information stored in the data warehouse directly depends on the questions we need to answer based on the data generated by the business. That is, the data warehouse should be oriented to business needs and is the cornerstone of the decision-making process. In other words, the data warehouse contains the information one may need to aggregate to run the business it is supposed to model.

References

Kimball, R., & Ross, M. (2013). *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling* (3rd ed.). Indianapolis, Indiana: John Wiley & Sons.

A. Script used to create the database

Figure 5 included in this appendix shows the SQL script that we used to create the table structure of the example data warehouse described in this document.

```sql
DROP DATABASE IF EXISTS dwh_example;
CREATE DATABASE dwh_example;

USE dwh_example;

DROP TABLE IF EXISTS Products;
CREATE TABLE Products (
    id INTEGER,
    variant VARCHAR(1) NOT NULL,
    PRIMARY KEY(id)
);

DROP TABLE IF EXISTS Timings;
CREATE TABLE Timings (
    id INTEGER,
    year VARCHAR(4) NOT NULL,
    quarter VARCHAR(2) NOT NULL,
    month VARCHAR(3) NOT NULL,
    day INTEGER NOT NULL,
    PRIMARY KEY(id)
);

DROP TABLE IF EXISTS Stockpiles;
CREATE TABLE Stockpiles (
    product INTEGER NOT NULL,
    timing INTEGER NOT NULL,
    volume DECIMAL(6,2) NOT NULL,
    FOREIGN KEY(product) REFERENCES Products(id),
    FOREIGN KEY(timing) REFERENCES Timings(id)
);
```

Figure 5 Script used to create the table structure of the example data warehouse