

Building data warehouses using open source technologies

Draft version 197

Michel Jansen

(mjansen@betterbe.com)



1 Table of Contents

1 Table of Contents.....	2
2 Introduction.....	3
2.1 Intended audience.....	3
2.2 Why build a data warehouse.....	3
3 Building a data warehouse.....	5
3.1 Designing a dimensional data warehouse.....	5
3.1.1 Asking questions.....	5
3.1.2 Modeling structures.....	5
3.1.3 Picking a fact grain.....	5
3.1.4 Adding dimensions.....	6
3.2 Constructing the data warehouse.....	6
3.2.1 Designing transformations using Spoon.....	7
Some preparations.....	7
Updating “type 1” dimensions.....	7
Updating “type 2” dimensions and the fact table.....	8
Aggregating the data.....	10
3.2.2 Putting it all together.....	10
4 Using the data warehouse.....	11
4.1 Preparing for on line analytical processing.....	11
4.1.1 From relational to dimensional.....	11
4.1.2 Doing it in Mondrian.....	11
4.2 Asking multidimensional queries.....	12
4.3 Visualization and presentation.....	13
5 References.....	14
Appendix A: Technology overview.....	15
Kettle.....	15
Mondrian.....	15
Jpivot.....	15
Appendix B: Generating a date dimension using JavaScript.....	16
JavaScript code.....	16
Appendix C: Example Mondrian XML schema.....	17
License.....	18

2 Introduction

This article is about building data warehouses. A data warehouse is a computer database that collects, integrates and stores an organization's data with the aim of producing accurate and timely management information and supporting data analysis¹. It explains the importance of a good data warehouse and cover the process of building such a specialized database using open source technologies.

2.1 *Intended audience*

This article is meant for software developers, database administrators, integrated software vendors or other people who are facing the challenge of making the analysis of large amounts of data generated by a business or an information system possible. It has been written with the assumption of a basic understanding of the covered concepts like databases, information systems and business transactions in mind.

2.2 *Why build a data warehouse*

There are many reasons to justify building a data warehouse, but almost all of them boil down to the same basic wish: provide means for analysis of data to support management decisions. You are probably already providing management with data like site usage statistics, referrer trends and the number of registered users of their information system. This is basic information, which can be retrieved directly from the information system itself. However, this has some important drawbacks.

First of all, since all of this data is retrieved directly from the information system, it places a considerable burden on this system. Analysis often requires huge amounts of data to be processed, which is often a problem if, for instance, the system's database tables get locked for retrieval. A data warehouse can be completely detached from the information system, even running on a different system.

Secondly, an OLTP²-system's data model is rarely optimized for analysis. We all learn to develop systems to use a normalized database modeled after our entity relationships. This is a good thing for the information system, since the underlying model is a close reflection of the system itself, but it makes querying for large sums of aggregated data a costly operation. Furthermore, redundancy is rarely part of this database design, because redundancy is hard to maintain, often causing data inconsistencies or worse. For data analysis, redundancy can be great, because it speeds up the process.

Moreover, data in an OLTP system might change over time. A customer might move to another country, leaving you, the data-analysis provider with an impossible choice: either you update the

1 http://en.wikipedia.org/wiki/Data_warehouse

2 On Line Transaction Processing

customer's record, discarding his previous state and invalidating previous analysis or you have to somehow create a new record for the on-line system and change all references to it. Neither of them are desirable, but in an off-line data warehouse you can keep both the old and the new state of the customer and specify at what period in time it has changed.

Finally, there are probably a lot of data sources your information system isn't using that could be useful in data analysis. A data warehouse can provide central storage for all this data, so all the collected information can be queried in one step.

3 Building a data warehouse

In this chapter, I will guide you through the different phases in building a data warehouse. I will illustrate this by using a simplified web-based information system as an example for creating a data warehouse. This system contains familiar entities such as “requests”, “users” and “pages”. The data model of this system is shown in figure <TODO>.

3.1 Designing a dimensional data warehouse

3.1.1 Asking questions

The most important step in building a data warehouse is designing it. You have to ask yourself: “What does the management want to know?”. First, you'll have to figure out which questions need to be answered by analysis of the data warehouse to be³. For example, is there a correlation between users of a web-based system and the pages it provides? Do certain groups of users visit other pages than others? Where do the visitors from different pages come from? My belief is at least 80% of all management questions about the data generated by an information system can be answered by a decent data warehouse.

Since you are reading this, I assume your company's management has already formulated this kind of questions. If not, make sure they do. Don't think too lightly about this phase, as it is the basis of what the resulting data warehouse can do.

3.1.2 Modeling structures

Once you get a good view on what questions you'll want to ask the data warehouse later on, you have to determine the different data-arrangements that come with these questions. For example, a question about sales will have to operate on a different structure than one about employment. These structures are called “cubes” in the world of OLAP⁴, because they are essentially an extension to the two-dimensional array of data that is stored in a conventional (for example SQL) database. Depending on the data, a cube may have more than two or even three dimensions. We'll model these cubes in a relational database as a star schema, containing a single *fact table* linking together multiple *dimension tables*.

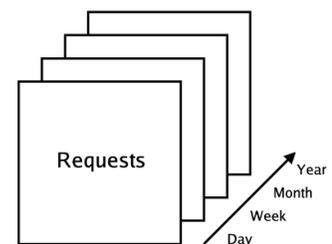


Illustration 1: Request cube containing only a time dimension

³ This is not a typo

⁴ On Line Analytical Processing

3.1.3 Picking a fact grain

The art in making up this structure is finding a good basis for a fact table. The more aggregated the chosen “fact” element for each line in the fact table is, the harder it becomes to add useful dimensions. The ideal basis for a fact table is the lowest possible atomic grain of data. This is often a single transaction, payment or , as in my example, a page request.

3.1.4 Adding dimensions

Now that we have a central starting point for our data cube, it is time to decorate the basic facts by adding new dimensions, containing aspects that can be attributed to the facts. If you can't think of any dimension that is valid for every value in your fact table, then you've probably chosen the wrong grain to begin with. The most trivial of all dimensions is that of time. Every request takes place on a certain point in time, so we create dimension table “time” and have every request entry in the fact table link to the entry in this table corresponding to the time it took place. If two requests took place on the same moment, they will link to the same entry in the time dimension's table.

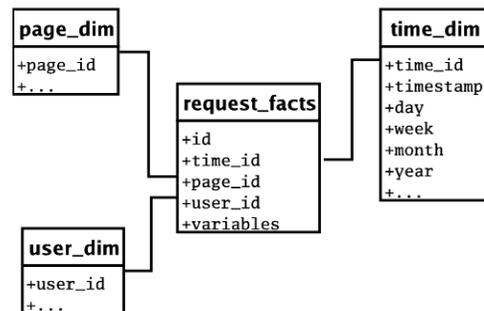


Illustration 2: A cube modelled as an OLAP-star in a relational database

Another dimension we want to add is that of pages. For each distinct page associated to a request in the fact table, an entry will exist in the page dimension table. The same goes for referrers.

Finally, we'll want to link requests to the users that made them. Every request in the fact table was made either by a known user, or by no user at all, which will be a special entry in the user dimension's table.

As we create and link dimensions, an important rule is to *never* use the existing keys from the on-line system, as we have no control over how they might change or even disappear. Instead, every dimension will get it's own surrogate key called the “technical key” which is unique to the data warehouse. This also goes for the fact table.

3.2 Constructing the data warehouse

Now that we know which cubes we want to express as a star-join schema, it's time to get to work. In this step, we'll populate a data warehouse with data from the OLTP system. This phase of the process is known as ETL, which stands for Extract, Transform, Load. This is exactly what needs to be done. Extract the data needed for the fact and dimension tables from all different data sources, transform it to fit our needs and load it into the data warehouse so it can be queried.

We'll perform these actions with KETTLE, an open source ETL tool. Kettle is roughly made up out of two packages. Spoon and Pan, which are used to create and execute transformations and Chef and Kitchen, which let you define jobs from transformations and schedule and execute them.

3.2.1 Designing transformations using Spoon

We start by using Spoon to make the transformations that will populate our data warehouse. In order to be able to fill the central fact table, the keys to all of the dimensions must be known. Therefore we make a distinction between two types of dimensions (not to be confused by Ralph Kimball's slowly changing dimension types):

1. Dimensions consisting of data already known to the on-line system.
2. Dimensions that are to be generated from the fact data and surrounding sources.

While we generate or update the dimensions of type two while filling our fact table and thus know its keys at that time, we cannot do this for the dimensions of the first kind.

In our example, the time and page dimensions are of the second kind, meaning they are generated when updating our fact table. The user dimension has to be known before then, because it is based on some independent tables in the on-line system. Because of this, we will fill our data warehouse in two separate transformations, ensuring first the existence of our type 1 user dimension and later the other dimensions and the fact table itself.

Some preparations

Before we can begin using Spoon, we have to define the data sources. In our case, we have only one data source: the database of our on line system. We need to add this database as a "Connection" to Spoon as described in its documentation about "Database connections". We also define the connection to our target database this way.

Updating type 1'd imensions

In our example, the user dimension is the only one that cannot be updated while filling the fact table. The data already exists in the source system as a single table, so all we have to do is read it and run it through Spoon's great "Dimension lookup/update" step.

This step is capable of creating, updating and looking up "slowly changing dimensions" as described by Ralph Kimball. For sources that contain changing data, such as the customer records mentioned earlier, it can do this in two ways:

- I. Overwriting the existing record by the updated one.

II. Creating another dimension record, maintaining multiple copies of changed records over time.

This second type is implemented by adding a “version”, “date_from” and “date_to” field to the dimension's table and it is almost always the most useful one.

In our example, we want the “user” dimension to be of type two, so the transformation in Spoon will look like Illustration 3. First, the data is read from the on-line system's user table, then the data we want to put in our dimension table is filtered using a “select” step and finally, it all gets inserted into our data warehouses user_dim table. Because all of the steps are quite accurately described in Spoon's documentation, we will only explain how to configure the “dimension lookup/update” step for this example.

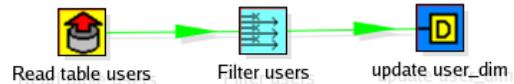


Illustration 3: The “update user dimension” transformation in Spoon

Because we create this dimension directly from a table from the on-line system, the key field to use in this dimension is trivial. It is the key used in the on-line system, which in our case is the user's email. As said before, this field will not be used as a key for our data warehouse, as we replace it by a technical key unique to the warehouse. It is merely needed to be able to retrieve this technical key later, as we will need it to link the dimension to the fact table. In our case, this technical key field is called “user_id”.

In the “Fields” tab, we specify the fields that are of importance to this dimension, such as hierarchical data. In our case, we keep track of information like the user's full name, the company he is part of and his or hers sex.

In case of “type II” dimensions, the “Version field” is used to keep track of the different versions in a slowly changing dimension. The date ranges will be used to indicate the period of validity for each version in that case. If the on-line system keeps track of when records change, it is useful to use the “Stream Datefield” for better validity, but there are many more applications possible, which are not in the scope of this document. Usually, the default (now) will suffice.

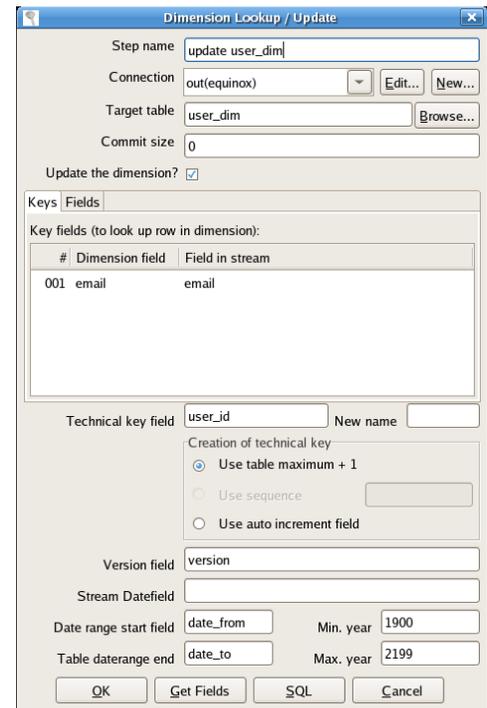


Illustration 4: Dimension Lookup / Update dialog

Updating type 2 dimensions and the fact table

Now that all independent dimensions have been prepared it is time to populate the fact table. In the

process, the remaining dimensions can be updated as well, having their technical keys ready when needed.

We start by reading the basic “grain” for the fact table from the request log.

Then, the date dimension is generated, which is done by using the JavaScript step on the timestamp field in the request log, as described in Appendix B: Generating a date dimension using JavaScript. Using the “Dimension Lookup/Update” step, we put this data into the database, and keep references to it in the form of the generated technical key “time_id”, which is added to the stream as an extra column.

Next up is the user dimension. Remember, we already updated that one in the previous step, so now all we have to do is link its corresponding entry for each request in the fact table. Through a complex lookup using the session key to match each request to a login action from the actions table, we are able to finally match a user to each request. If no user can be found for this session, we turn its key (the email) to NULL, so it will match a special case created by Spoon's “Dimension Lookup/Update” step: the unknown user. Because the filtering creates separate threads, we sort the user streams afterwards using Spoon's sort step. When it reaches the “Lookup user_dim” step, the stream contains an extra field named “email”, which contains the email of each request, or NULL if it is not known. Remember we specified “email” as the lookup key when updating this dimension earlier, so using this field, the transformation step can find a technical key for each entry in the stream, adding it as an extra field called “user_id”.

All of the information needed to fill the page dimension is already in the request log's stream. While filling the dimension table with hierarchical data about this request's domain, path and page, the newly generated technical key field “page_id” is added to the stream.

The same is done for a referrer dimension we added for fun, which was not mentioned earlier, but is similar to the page

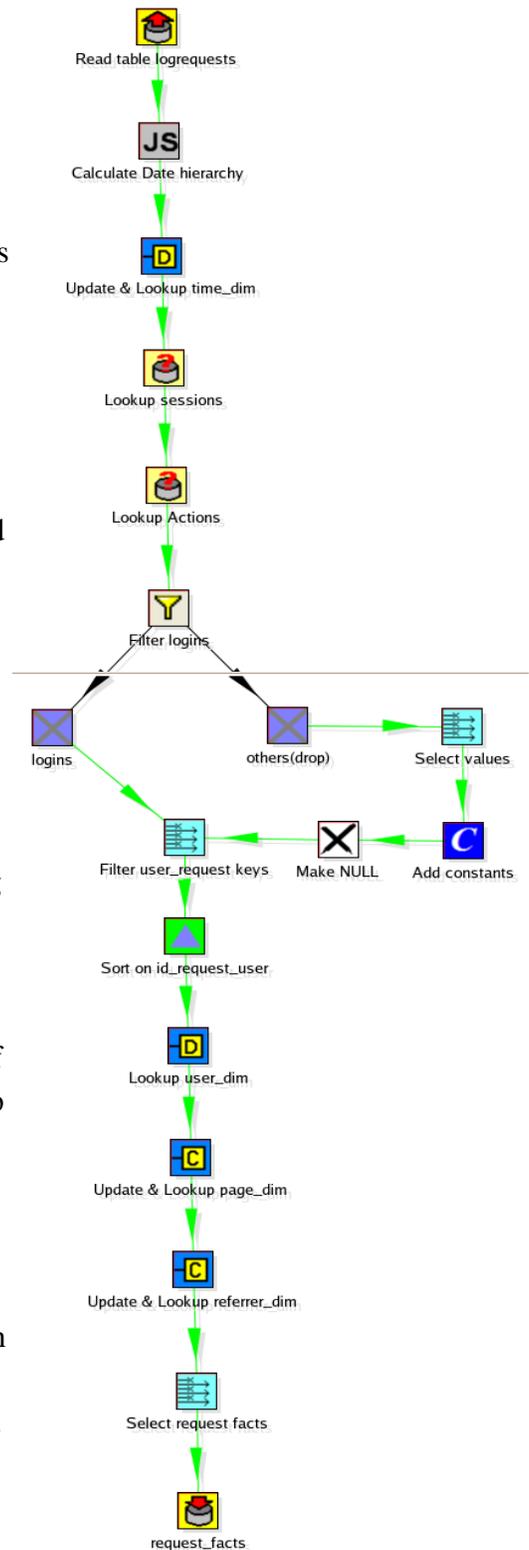


Illustration 5: Dimension and fact update transformation

dimension.

Finally, the data that is to go into the fact table is filtered so that only the technical keys to the dimensions and the grain's facts remain. It is then inserted into a table, request_facts in this example.

Aggregating the data

TODO: Explain how to make aggregate tables for speeding up analysis.

3.2.2 Putting it all together

Now that we've defined the transformations that fill our data warehouse, it's time to prove that one plus one equals three by using Kettle's Chef and Kitchen to combine the transformations into one job, adding logging and debugging functionality in the process.

The tool Chef doesn't require much explanation, as its interface is almost equal to that of Spoon. Where Spoon is used to combine steps to create a transformation, Chef links together transformations to make a job. Because Chef is already pretty well-documented, a simple screen shot says it all for this example.

Because there are two types of dimensions in our case, the process of filling the data warehouse is divided in two transformation phases, as described earlier. First, Chef will execute the transformation that updates our “type 1” User dimension. On failure, it will send an e-mail with more details on what went wrong. If not, it will continue with the second phase: the transaction that updates the other dimensions and the fact table. In this example, I also wanted to receive an e-mail on success, so I also linked the last transformation to the “Mail” step.

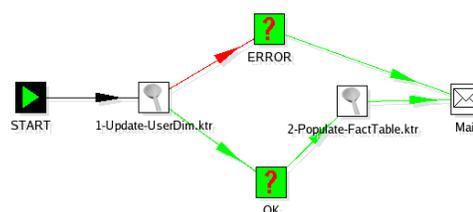


Illustration 6: The Chef job to fill the data warehouse

A final feature of Chef is that its jobs can be scheduled. Using Kitchen as a command line tool, this makes Kettle a very powerful system for creating, updating and maintaining data warehouses.

4 Using the data warehouse

This chapter describes how to perform analysis on a data warehouse. It assumes a star-joined database structure, like the one described in the previous chapter's example, is in place.

4.1 *Preparing for on line analytical processing*

In section 3.1.2: Modeling structures, we started by structuring data cubes in the data warehouse as a relational star schema, containing a single *fact table* linking together multiple *dimension tables*. This way, each dimension is linked to every other dimension in an n-n way for every cube it is part of.

Because of the special way the data is organized, it is possible to query the data in a generalized form. This is the task for an OLAP server. In our example, we will use the open source relational OLAP server Mondrian to do so.

4.1.1 From relational to dimensional

As explained earlier, the data in the warehouse is stored in a relational form. A fact table links together multiple dimensions, each in their own table, containing their own data. In an OLAP *cube*, data is organized in a multidimensional form that consists of all of these tables together.

Each cube can contain multiple *measures*, that can be obtained from the fact table. These measures are used to calculate aggregations and group data. A cube's measures can be anything, like sales quantity, product price or a calculated total of these facts. In the case of factless fact tables* there doesn't even have to be a measure, it can be "1" all the time.

Every cube also has multiple *dimensions*. A dimension is one way of looking at the data, constructed from usually one dimension table in the relational database. The most trivial example of a dimension is of course, that of time.

Every dimension can have one or more *hierarchies*. A hierarchy is, as you might expect, an ordered set of increasingly aggregated *levels* of information about the dimension. In case of a time dimension, one hierarchy might look like: *day – month – quarter – year*. Hierarchies are very useful in grouping and aggregating data while analyzing.

These concepts can be used by an OLAP server to query the data in a multidimensional way. I will now tell you how to configure the Mondrian OLAP server to do so.

4.1.2 Doing it in Mondrian

After the installation of the Mondrian OLAP server, which is trivial and outside the scope of this document, we need to tell it about our data warehouse's structure. This is done using a fairly well-

documented XML format, containing information about how the OLAP-cubes and their dimensions, hierarchies and measures are to be built from tables in a relational database.

Let's do this for our example. As can be seen in Appendix C: Example Mondrian XML schema, the configuration starts by telling Mondrian the name of the schema. In our case, the schema contains only one *cube*, which is called “Requests”. The requests cube is linked directly to the “request_facts” table, and has one *measure* and four *dimensions*.

Because the only facts in the fact table are the variables and we don't really care about those right now, the *measure* is a trivial one: The count of the number of id's of each request, which is always equal to one.

The four *dimensions* are defined as being part of the cube. In our case we have a “User”, “Page”, “Referrer” and “Time” dimension. Each containing one or more hierarchies. For every dimension, Mondrian needs to know which by which table it is represented and by what key it is linked to the fact table. For each dimension, a table column or calculated value is to be defined.

Once Mondrian knows how to interpret the data warehouse's relational database, it can act as a translation layer, allowing multidimensional queries to be run on a relational database.

4.2 Asking multidimensional queries

The dimensional nature of the OLAP cubes Mondrian uses, makes it possible to ask complex questions to our data warehouse. To do this, Mondrian supports a dialect of the MDX language, which is a multidimensional variant to SQL. It's syntax is roughly like that in Illustration 7: Basic MDX syntax.

```
SELECT [<axis_specification>
      [, <axis_specification>...]]
FROM  [<cube_specification>]
[WHERE [< slicer_specification>]]
```

Illustration 7: Basic MDX syntax

MDX differs from SQL in that SQL queries always return a two dimensional data set: a single table with a fixed number of columns on one axis and one or more rows on another axis, whereas MDX queries operate on a multidimensional data set and therefore return a result with any number of axes. An example of what such a query might look like when we want to know what companies visited our on line system in which months, is shown in Illustration 8: Example MDX query.

```
SELECT {[Time].[Months].Members} ON COLUMNS,
       {[Users].[Company Name].Members} ON ROWS
FROM   [Requests]
WHERE  [Time].[2006]
```

Illustration 8: Example MDX query

4.3 Visualization and presentation

The last step in providing a useful system for analysis to the end user is adding a presentation layer that is capable of formulating multidimensional queries and visualizing their results. One of the tools capable of doing so is JPivot.

JPivot is included in recent distributions of Mondrian, so it requires no additional set-up. It's configuration is easy: just specify the dataset to use in a JSP file and start querying.

Anything is possible. As more projects like JPivot and JFreereports are being embedded in the Pentaho project, it will become easier to generate the kind of presentation required for any situation.

5 References

A Dimensional Modeling Manifesto – Ralph Kimball 1997

Slowly Changing Dimensions – Ralph Kimball 1996

Spoon Documentation – Pentaho 2001-2006

6 History of this document

v146 - 1st of September 2006 – Initial release

The first version to be released to the community, still in draft form. I would like to thank my employer, Better be, for giving me the time and resources to work on this document. Without their support, this document would not exist. Also, I would like to thank, in advance, anyone who will contribute to this document in the future. You make open source to what it is today.

- Michel Jansen

v197 - 6th of October 2006

A revision with a lot of the sharp edges removed.

1. Fixed the calculation of quarters in the JavaScript example – Thanks Paul Keenan
2. Corrected the “Asking questions” section – Thanks Ronald van der Blink

- Michel Jansen

Appendix A: Technology overview

In this chapter, I'll explain some important technologies used and their place in the data warehouse creation process.

Kettle

<http://www.kettle.be>

Kettle is an open source ETL-suite for Extracting data from various sources, Transforming it and Loading it into the data warehouse.

Kettle consists of two sets of tools. Spoon and Pan are used to respectively create and execute transformations on data. Chef and Kitchen are used to organize transformations into jobs and execute them in a scheduled way.

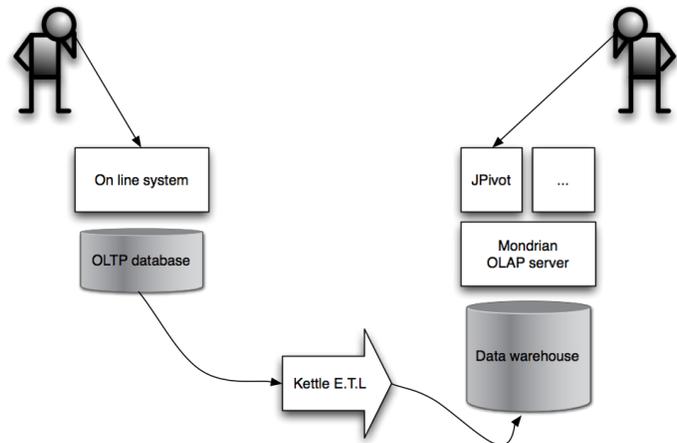


Illustration 9: Used technologies and their place

Mondrian

<http://mondrian.sourceforge.net>

Mondrian is a relational OLAP server, which acts as a layer on top of a relational database to allow for multidimensional queries to be performed on the data.

Jpivot

<http://jpivot.sourceforge.net>

JPivot is a tool that can act as a presentation layer on top of Mondrian. It generates multidimensional queries and displays their results as interactive pivot tables.

Appendix B: Generating a date dimension using JavaScript

A dimension present in almost all data warehouses is that of time. Spoon's JavaScript step makes it easy to extract hierarchical data from date or time fields in the source system. While most of this is described in Spoon's documentation, an example of how to do this is given here.

JavaScript code

```
// The fields we want to calculate
var day_of_month;
var week_of_year;
var month_of_year;
var year;
var quarter;
var name_day;
var name_month;

// Calculate!
day_of_month = dateTime.Clone().dat2str("dd");
week_of_year = dateTime.Clone().Clone().dat2str("ww");
month_of_year = dateTime.Clone().dat2str("MM");
year = dateTime.Clone().dat2str("yyyy");

name_day = dateTime.Clone().dat2str("E").getString();
name_month = dateTime.Clone().dat2str("MMMM").getString();

if(month_of_year <= 3) {
    quarter = "Q1";
}
else if(month_of_year <= 6) {
    quarter = "Q2";
}
else if(month_of_year <= 9) {
    quarter = "Q3";
}
else {
    quarter = "Q4";
}
```

Appendix C: Example Mondrian XML schema

```

<?xml version="1.0"?>
<Schema name="DataWareHouseTest">
  <Cube name="Requests">
    <Table name="request_facts" />
    <Dimension name="Users" foreignKey="user_id">
      <Hierarchy hasAll="true" allMemberName="All Users" primaryKey="user_id">
        <Table name="user_dim" />
        <Level name="Company Branche" column="companybranche" uniqueMembers="false"/>
        <Level name="Company Name" column="companyname" uniqueMembers="false"/>
        <Level name="Role" column="role" uniqueMembers="false"/>
      </Hierarchy>
      <Hierarchy hasAll="true" primaryKey="user_id">
        <Level name="Sex" column="sex" uniqueMembers="false"/>
        <Level name="User" column="email" uniqueMembers="true">
          <Property name="First Name" column="firstname"/>
          <Property name="Infix" column="infix"/>
          <Property name="Last Name" column="lastname"/>
        </Level>
      </Hierarchy>
    </Dimension>
    <Dimension name="Page" foreignKey="page_id">
      <Hierarchy hasAll="true" primaryKey="page_id">
        <Table name="page_dim" />
        <Level name="Domain" column="domain" />
        <Level name="Path" column="path" />
        <Level name="Page" column="page" />
      </Hierarchy>
    </Dimension>
    <Dimension name="Referrer" foreignKey="referrer_id">
      <Hierarchy hasAll="true" primaryKey="referrer_id">
        <Table name="referrer_dim" />
        <Level name="Domain" column="referrerDomain" />
        <Level name="Path" column="referrerPath" />
      </Hierarchy>
    </Dimension>
    <Dimension name="Time" type="TimeDimension" foreignKey="time_id">
      <Hierarchy hasAll="false" primaryKey="time_id">
        <Table name="time_dim" />
        <Level name="Year" column="year" type="Numeric" uniqueMembers="true"
          levelType="TimeYears"/>
        <Level name="Quarter" column="quarter" uniqueMembers="false"
          levelType="TimeQuarters"/>
        <Level name="Month" column="month_of_year" uniqueMembers="false" type="Numeric"
          levelType="TimeMonths"/>
      </Hierarchy>
      <Hierarchy hasAll="true" name="Weekly" primaryKey="time_id"
        defaultMember="[Time.Weekly].[All Time.Weeklys]">
        <Table name="time_dim" />
        <Level name="Year" column="year" type="Numeric" uniqueMembers="true"
          levelType="TimeYears"/>
        <Level name="Week" column="week_of_year" type="Numeric" uniqueMembers="false"
          levelType="TimeWeeks"/>
        <Level name="Day" column="day_of_month" uniqueMembers="false" type="Numeric"
          levelType="TimeDays"/>
      </Hierarchy>
    </Dimension>
    <Measure name="Pages" column="id" aggregator="count" datatype="Integer" formatString="#,##" />
  </Cube>
</Schema>

```

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

1. "Collective Work" means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.

2. "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.

3. "Licensor" means the individual or entity that offers the Work under the terms of this License.

4. "Original Author" means the individual or entity who created the Work.

5. "Work" means the copyrightable work of authorship offered under the terms of this License.

6. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

7. "License Elements" means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, ShareAlike.

2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

1. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;

2. to create and reproduce Derivative Works;

3. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;

4. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission Derivative Works.

5.

For the avoidance of doubt, where the work is a musical composition:

1. Performance Royalties Under Blanket Licenses. Licensor waives the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work.

2. Mechanical Rights and Statutory Royalties. Licensor waives the exclusive right to collect, whether individually or via a music rights society or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions).

6. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor waives the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions).

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such

Building data warehouses using open source technologies - Michel Jansen

modifications as are technically necessary to exercise the rights in other media and formats. All rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

1. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any credit as required by clause 4(c), as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any credit as required by clause 4(c), as requested.

2. You may distribute, publicly display, publicly perform, or publicly digitally perform a Derivative Work only under the terms of this License, a later version of this License with the same License Elements as this License, or a Creative Commons iCommons license that contains the same License Elements as this License (e.g. Attribution-ShareAlike 2.5 Japan). You must include a copy of, or the Uniform Resource Identifier for, this License or other license specified in the previous sentence with every copy or phonorecord of each Derivative Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Derivative Works that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder, and You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Derivative Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Derivative Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Derivative Work itself to be made subject to the terms of this License.

3. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Derivative Works or Collective Works, You must keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or (ii) if the Original Author and/or Licensor designate another party or parties (e.g. a sponsor institute, publishing entity, journal) for attribution in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; the title of the Work if supplied; to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Derivative Work or Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE MATERIALS, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

1. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Derivative Works or Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

2. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the

Building data warehouses using open source technologies - Michel Jansen

terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

1. Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

2. Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

3. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

4. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

5. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.