Design and implementation of open-data data warehouse Documentation

Release 0.0.1

Richard Banyi

May 28, 2016

Contents

CHAPTER 1

Introduction

1.1 Data

1.1.1 What is Data?

Data is all around us. But what exactly is? Data is a value assigned to a thing. What can we say about the balls in the picture? They are tennis balls, right? So one of the first data points we have is that they are used for tennis. Tennis is a category sport, so this helps us to to put the balls in a taxonomy. But there is more to them. We have the colour: "green", the condition "new". They all have sizes. All kind of objects have lot of data attached to them. Even people do: they have a name a date of birth, weight, height etc. All these things are data [1].

Fig. 1.1: Tennis Balls

Qualitative data is everything that refers to the quality of something. A description of experiences, a description of colours, and interview are all qualitative data. Data that can be observed but not measured.

Quantitative data is data that refers to a number. Data that can be measured. E.g. the number of tennis balls, the size, a score on a test etc.

[1] Author: School of Data Date: Sep 02, 2013 Organization: School of Data Available: Data Fundamentals

Categorical data puts the item you are describing into a category. For example the condition "new", "used", "broken" etc.

Discrete data is a numerical data that has gaps in it: e.g. the count of golf balls. There can only be whole numbers of tennis balls, there is no such thins as 0.5 golf balls.

Continues data is a numerical data with a continues range: e.g. size of a tennis balls can be any size (86.02mm), or the size of your foot (as opposed to your shoe size, which is discrete). In continues data, all values are possible with no gaps between .

1.1.2 From Data to information knowledge

Data, when collected and structured suddenly becomes a lot more useful.

Category	Contract
Date	2015
Amount	$1232.21
Recipient	Apple Inc.

Data: Content that is directly observable or verifiable; a fact - it's -5C outside.

Information: Content that represents analyzed data - "it's -5C outside I'll take a warm coat".

Knowledge: "I remember the last time when was this cold I got a cold. I'll therefore take a scarf and gloves. But first I'll check with Brian. He usually dress too light for this kind of weather." Knowledge is created when the information is learned, applied and understood.

Context: One thing incredibly important for data is context: A number or quality doesn't mean a thing if you don't give context. So explain what you are showing – explain how it is read, explain where the data comes from and explain what you did with it. If you give the proper context the conclusion should come right out of the data.

1.1.3 Finding Data

Data Source	Description
csv	Comma Separated values (CSV)
xls	MS Excel Spreadsheet
gdoc	Relational database table
mongodb	MongoDB database

There are three basic ways of getting hold of data:

1. **Finding data** - involves searching and finding data that has been already released
2. **Getting hold of more data** - asking for "new" data from official sources e.g. trough Freedom of Information requests. Sometimes data is public on a website but there is not a download link to get hold of it in bulk! This data can be liberated with what datawranglers call scraping.

3. **Collecting data yourself** - gathering data and entering it into a database or a spreadsheet.

1.1.4 Identify data source

In recent years *governments* have begun to release some of their data to the public - open data. Many governments host special (open) government data platforms for the data they create. For example the UK government started UK Data, or USA data.gov. Other sources of data are large *organisations*. The World Bank and the World Health Organization for example regularly release reports and data sets. Scientific projects and institutions release data to the scientific community and the general public. Open data is produced by NASA for example, and many specific disciplines have their own data repositories, some of which are open.

1.1.5 Problem with sources

There are plenty of places where you can get hand on open datasets which are open to public, however these sources are often unstructered, very messy, ambiguous, mis-use of class attributes, non-consistent, missing values, etc. The unstructured data growing quickiest than the other, and their exploitation could help in business decision.

```
5896 34513500 34513500-1     \N      \N Transport equipment
3031 48825000 48825000-7     \N      \N      Software package
5821 34144900 34144900-7     1.9721673495    \N      Transport equipment
```

1.2 Open Data

Do you know exactly how much of your tax money is spent on street lights or on public transportation? And what is in the air that you breathe along the way? Where in your region will you find the best job opportunities and the highest number of fruit trees per capita? When can you influence decisions about topics you deeply care about, and whom should you talk to?

New technologies now make it possible to build the services to answer these questions automatically. Much of the data you would need to answer these questions is generated by public bodies. However, often the data required is not yet available in a form which is easy to use - take for example our country Slovakia it still lack of data transparency and creating data sets which can be easy to used.

The notion of open data and specifically open government data - information, public or otherwise, which anyone is free to access and re-use for any purpose - has been around for some years.

1.2.1 Why open Data?

Open data, especially open government data, is a tremendous resource that is as yet largely untapped. Many individuals and organisations collect a broad range of different types of data in order to perform their tasks. Government is particularly significant in this respect, both because of the quantity and centrality of the data it collects, but also because most of that government data is public data by law, and therefore could be made open and made available for others to use. Why is that of interest?

There are also many different groups of people and organisations who can benefit from the availability of open data, including government itself. At the same time it is impossible to predict precisely how and where value will be created in the future.

It is already possible to point to a large number of areas where open government data is creating value. Some of these areas include:

- Transparency and democratic control
- Participation
- Self-empowerment
- Improved or new private products and services
- Innovation
- Improved efficiency of government services
- Improved effectiveness of government services
- Impact measurement of policies
- New knowledge from combined data sources and patterns in large data volumes

Open government data can also help you to make better decisions in your own life, or enable you to be more active in society. A woman in Denmark built findtoilet.dk, which showed all the Danish public toilets, so that people she knew with bladder problems can now trust themselves to go out more again. Services like 'mapumental' in the UK and 'mapnificent' in Germany allow you to find places to live, taking into account the duration of your commute to work, housing prices, and how beautiful an area is. All these examples use open government data.

Open data is also of value for government itself. For example, it can increase government efficiency. The Dutch Ministry of Education has published all of their education-related data online for re-use. Since then, the number of questions they receive has dropped, reducing work-load and costs, and the remaining questions are now also easier for civil servants to answer, because it is clear where the relevant data can be found. Open data is also making government more effective, which ultimately also reduces costs.

While there are numerous instances of the ways in which open data is already creating both social and economic value, we don't yet know what new things will become possible. New combinations of data can create new knowledge and insights, which can lead to whole new fields of application.

We have seen this in the past, for example when Dr. Snow discovered the relationship between drinking water pollution and cholera in London in the 19th century, by combining data about cholera deaths with the location of water wells.

This untapped potential can be unleashed if we turn public government data into open data. This will only happen, however, if it is really open, i.e. if there are no restrictions (legal, financial or technological) to its re-use by others. Every restriction will exclude people from re-using the public data, and make it harder to find valuable ways of doing that. For the potential to be realized, public data needs to be open data.

1.2.2 What is Open Data?

Open data is data that can be freely used, re-used and redistributed by anyone - subject only, at most, to the requirement to attribute and sharealike [3].

The full Open Definition gives precise details as to what this means. To summarize the most important:

- **Availability and Access:** the data must be available as a whole and at no more than a reasonable reproduction cost, preferably by downloading over the internet. The data must also be available in a convenient and modifiable form.
- **Re-use and Redistribution:** the data must be provided under terms that permit re-use and redistribution including the intermixing with other datasets.
- **Universal Participation:** everyone must be able to use, re-use and redistribute - there should be no discrimination against fields of endeavour or against persons or groups. For example, 'non-commercial' restrictions that would prevent 'commercial' use, or restrictions of use for certain purposes (e.g. only in education), are not allowed.

If you're wondering why it is so important to be clear about what open means and why this definition is used, there's a simple answer: **interoperability**.

Interoperability denotes the ability of diverse systems and organizations to work together (interoperate). In this case, it is the ability to interoperate - or intermix - different datasets.

Interoperability is important because it allows for different components to work together. This ability to componentize and to 'plug together' components is essential to building large, complex systems. Without interoperability this becomes near impossible — as evidenced in the most famous myth of the Tower of Babel where the (in)ability to communicate (to interoperate) resulted in the complete breakdown of the tower-building effort. We face a similar situation with regard to data. The core of a "commons" of data (or code) is that one piece of "open" material contained therein can be freely intermixed with other "open" material. This interoperability is absolutely key to realizing the main practical benefits of "openness": the dramatically enhanced ability to combine different datasets together and thereby to develop more and better products and services.

[3] Open Definition see

Providing a clear definition of openness ensures that when you get two open datasets from two different sources, you will be able to combine them together, and it ensures that we avoid our own 'tower of babel': lots of datasets but little or no ability to combine them together into the larger systems where the real value lies [4].

1.3 Data Warehouse Fundamentals

1.3.1 What is Data Warehouse

A data warehouse (DW or DWH) is a system used for reporting and data analysis. Data Warehouse's are central repositories of integrated data from one or more disparate sources. They store current and historical data and are used for creating analytical reports for knowledge workers throughout the enterprise. Examples of reports could range from annual and quarterly comparisons and trends to detailed daily sales analyses.

A data warehouse is a system that extracts, cleans, conforms, and delivers source data into a dimensional data store and then supports and implements querying and analysis for the purpose of decision making [5].

1.3.2 Operational Systems

An operational system directly supports the execution of a business process. By capturing details about significant events or transactions. A sales system, for example captures information about orders, shipments, and returns.

Operational systems must enable several types of database interaction, including inserts, updates, and deletes - these interactions are almost always atomic. For example, an order entry system must provide for the management of lists of products, customers, and salespeople; the entering of orders; the printing of order summaries, invoices, and packing lists; and the tracking order status. The operational system is likely to update as things change (if a customer moves, his/her old address is no longer useful so it is simply overwritten), and archive data ones it's operational usefulness has ended. Operational systems are implemented in a relational database, the design may called entity-relationship model, or ER model. The schema of operational systems are highly accepted to be in third normal form.

1.3.3 Analytic Systems

An analytical system supports the *evaluation* of a business process. How are orders trending this month versus last? Where does this put us in comparison to our sales goals for the quarter? Is a

[4] Open Knowledge Open Data Handbook

[5] The Data Warehouse ETL Toolkit, Ralph Kimball, Joe Casetra, Copyright 2004 by Wiley Publishing, Inc. All rights reserved., eISBN: 0-764-57923-1

particular marketing promotion having an impact on sales? Who are our best customers?

Interaction with an analytic system takes place through queries that retrieve data about business processes. Historic data will remain important to the analytic system long after its operational use has passed.

OPERATIONAL SYSTEM VS. ANALYTICAL SYSTEM

	Operational System	Analytic System
Purpose	Execution of a business process	Measurement of a business process
Primary Interaction Style	Insert, Update, Delete, Query	Query
Scope of Interaction	Individual transaction	Aggregated transactions
Querry Patterns	Predictable and stable	Unpredictable and changing
Temporal Focus	Current	Current and historic
Design Optimaziation	Update concurrency	High-performance query
Design Principle	Entity-relationship (ER) design in third normal form (3NF)	Dimensional design (Starschema or Cube)
Also Known As	Transaction System,Online Transaction Processing System (OLTP),Source System	Data Warehouse System,Data Mart

1.3.4 Analytic Databases and Dimensional Design

The dimensional model of a business process is made up of two components: *measurements* and their *context*. Known as facts and dimensions, these components are organized into a database design that facilities a wide variety of analytic usage. Implemented in a relational database, the dimensional model is called a star schema. Implemented in a multidimensional database, it is known as a cube. The core of every dimensional model is a set of business metrics that captures how a process is evaluated, and a description of the context of every measurement [6].

Purpose

Analytic systems and operational systems serve fundamentally different purposes. An operational system supports the execution of a business process, while and analytic system supports the evaluation of the process [7].

Measurement and Context

[6] Excerpt From: Adamson, Christopher. "Star Schema The Complete Reference." Copyright 2010 by The McGraw-Hill Companies, Inc. All rights reserved. ISBN: 978-0-07-174433-1

[7] Excerpt From: Adamson, Christopher. "Star Schema The Complete Reference." Copyright 2010 by The McGraw-Hill Companies, Inc. All rights reserved. ISBN: 978-0-07-174433-1

Dimensional design supports analysis of a business process by modeling how it is measured. Consider the following business questions:

- What are gross margins by product category for June?
- What is the average transaction by states level?
- What is the return rate by visitors?

These questions do not focus on individual activities or transactions. To answer them, it is necessary to look at a group of transactions - in a bigger picture. Each of these questions reveals something about how its respective business process is measured.

Every dimensional solution describes a process by capturing what is measured and the context in which the measurements are evaluated [8].

Facts and Dimensions

In a dimensional design, measurements are called facts, and context descriptors are called dimensions. Facts tend to be numeric in value. Elements that are aggregated, summarized, or subtotaled are facts.

FACTS	DIMENSIONS
Amount	Product
Min Amount	Agency
Max Amount	Award
	Geography

1.3.5 The Star Schema

A dimensional design for a relational database is called a star schema. Related dimensions are grouped as columns in dimension tables, and the facts are stored as columns in a fact table.

Dimension tables are not in third normal form. A dimensional model serves a different purpose from ER model. It is not necessary to isolate repeating values in an environment that doesn't support transaction processing. When additional normalization is performed within dimensions, in such cases, the schema is referred as a snowflake.

Dimension Tables

In a star schema, a dimension table contains columns representing dimensions. These columns provide context for facts.

Fact Table

At the core of a star schema is the fact table. Each row in the fact table stores facts at a specific level of detail. This level of detail is known as the fact table's grain

[8] Excerpt From: Adamson, Christopher. "Star Schema The Complete Reference." Copyright 2010 by The McGraw-Hill Companies, Inc. All rights reserved. ISBN: 978-0-07-174433-1

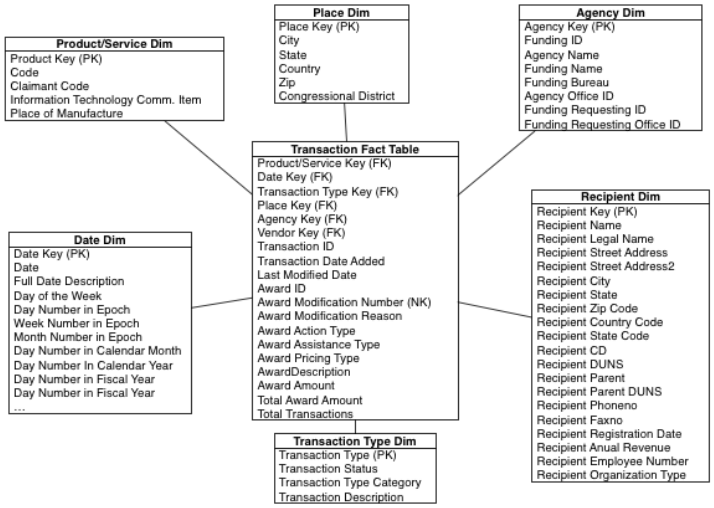

Fig. 1.2: Star Schema example

CHAPTER 2

Building The Data Warehouse

2.1 Plan

2.1.1 Goals of Data Warehouse

Before we delve into the details of dimensional modeling and implementation, it is helpful to focus on fundamental goals of the data warehouse. How can we focus on these these fundamental goals if we are missing the most important thing, the **data**.

Based on our experience, the data is the universum that drive the bedrock requirements for the data warehouse. The data comes first, than the technology and the bussiness model.

Finding the Data

After couple of weeks of searching we have identified several open to public data sources:

- UK Gov
- Open Spending
- U.S. Government's open data
- Usa Spending Gov

We have decided to looking for government data. We exemined couple of datasets and we picked Usa Spending Gov. Their data looked the most promicing and they provide great document information about the data fields, description and their formats.

Identifying the data source

All the prime recipient transaction data on *USAspending.gov <https://www.usaspending.gov>* is reported by the federal agencies making contract, grant, loan, and other financial assistance awards. After identifying the data source and examination of the data sets we have decided to use this data for our business model and data warehouse model.

Mission of data warehouse

We have concluded the following goals for our data warehouse:

The main mission of the data warehouse is to publish the federal organisations data. The key success of our data warehouse is whether the data warehouse effectively contributes to the general public. The success of a data warehouse begins end ends with its users.

The data warehouse is going to be open to public. **It must make the information easily accessible.** The content of the data warehouse must be understandable. The data must be intuitive and obvious to the business user, not merely the developer.

The data warehouse must present the federal organisations information consistently. Data must be carefully assembled from a variety of sources around the organization, cleansed, quality assured, and released only when it is fit for user consumption.

The data warehouse must be adaptive and resilient to change. We want to track changes of federal awards made by federal agencies. The data warehouse must be designed to handle this change.

2.2 Data warehouse Environment

It is helpful to understand the pieces of the data warehouse before we begin to combine them. Each component serves specific function.

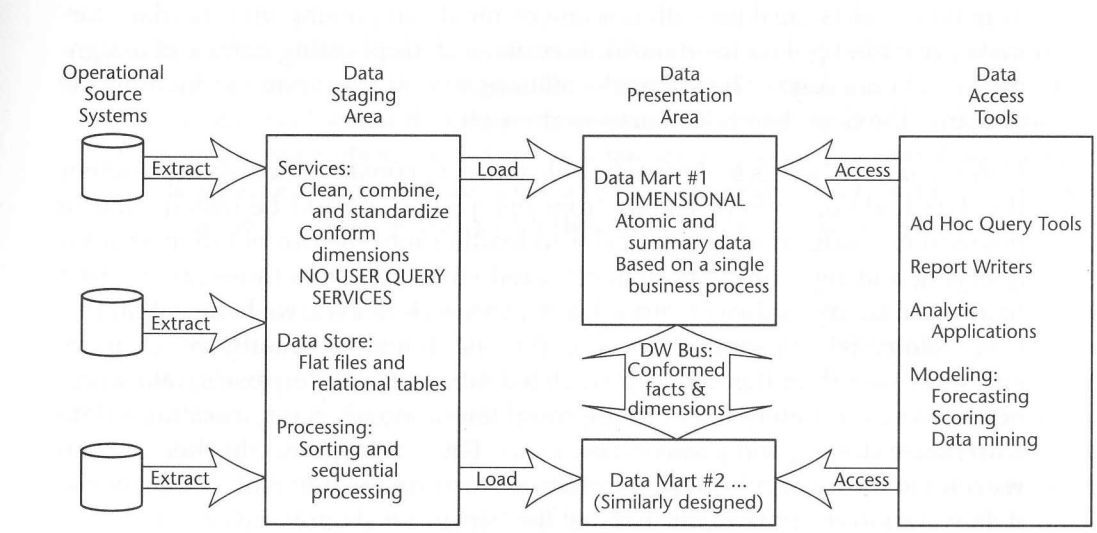

Figure 1.1 Basic elements of the data warehouse.

Fig. 2.1: Data warehouse components

2.2.1 Operational Source systems

The operational system directly captures the execution of a business process, in our case capture the transactions of federal agencies. The operational source system is outside the data warehouse, we have no control over the content and format of the data in these systems. In our case the operational system is usaspending.gov where we just download directly from they site a .csv flat file.

2.2.2 Data Staging Area

The data staging is a physical storage area and a set of extract-transformation-load (ETL) jobs. The data staging are is everything between the source systems and the data presentation area. This is the stage where we perform various process on the data to fit into data warehouse environment.

2.2.3 Data presentation

The data presentation area is where the structured data is organized, stored, and made available for querying and for analytical applications. The presentation area is based on online analytic processing (OLAP) technology, this means the data is stored in cubes.

2.2.4 Data access tools

The final piece of the data warehouse is the data access tool. Obviously the first access tool can be a simple query tool for querying. We provide to our end users the general public set of tools for development of reporting, analysis and browsing of data trough our web application using the concept of Cubes.

2.3 Data Staging Area

The ETL system is the foundation of the data warehouse. We have design an ETL system that extracts data from the source system, enforces data quality and consistency, conforms data so that separate flat files can be used together, and delivers data into presentation layer where we build the application so that end users can make decisions. We haven't used any ETL tools every process/job is all hand coded.

2.3.1 Extract

The usaspending.gov website doesn't provide any API, only single page with the download able link. The raw data coming from the source system is stored locally on the disk. We have down-

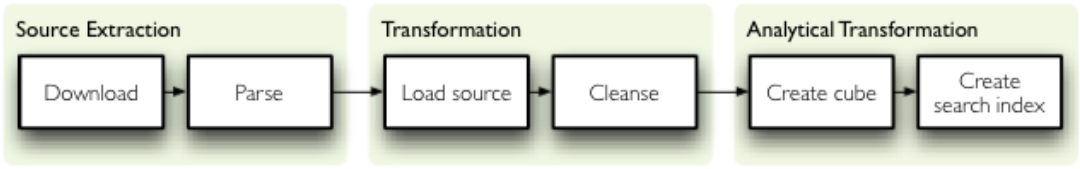

Fig. 2.2: ETL process flow

loaded 4 different .csv flat files.

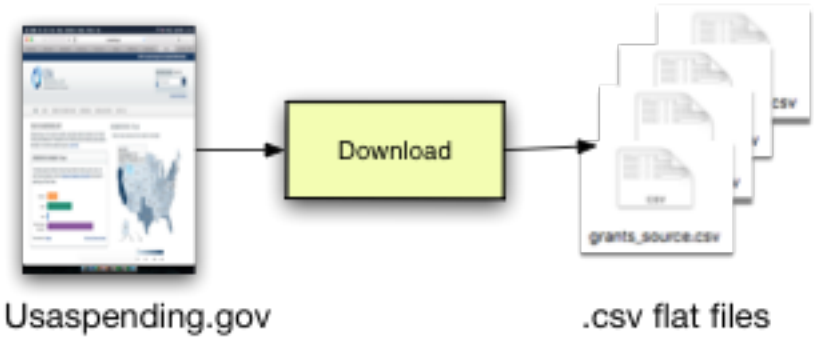

Fig. 2.3: Figure [5]

We have examined the .csv flat files. We have encountered various errors when we wanted to load the data into our database. Here is an example of a source flat file:

Listing 2.1: Example of raw data

```
"07: Direct loan","2",": Current record","","4967000","UTAH","SALT, LAKE","USA","84
```

The empty double quotes PostgreSQL treats as empty string and not as NULL value. When we wanted to insert the data into the database we have always encountered syntax type error for numeric types. We have decided to get rid of the double quotes in the flat files. For that reason we have made a little Python script which strip all the double quotes, in a way that we haven't disturb data integrity.

Listing 2.2: Script to strip double quotes

```
for file in csv_files:
  source = open(file, 'r')
  source_name = os.path.basename(file).split('_')
  cleaned_csv = open(path_cleaned+source_name[0]+'_cleaned.csv', 'w')
  reader = csv.reader(source)
  writer = csv.writer(cleaned_csv)
```

```
    for row in reader:
        writer.writerow(row)
    source.close()
    cleaned_csv.close()
```

After we have stripped all the double quotes from all the flat files we have managed to load the data into database. All the tables start with a prefix *base_* + name of the flat file from the source. For example for grants: *base_grants* table.

PICTURE

After we have created the base tables where we have loaded the extracted data from the source. We started with the highest level of business objectives, identify the likely data from the base tables, that we believe will support the decisions needed by the business model - which we have documented in the Dimensional Modelling sections. We started with identifying specific data elements we believed are the most important for the business model. We analysed the data in the base tables and come to conclusion that data in these tables must be examined for data quality, completeness, and fitness for the purpose of the data warehouse. For the simplicity we have designed logical data mapping document. The document contains the data from the source systems throughout to the target data table, and the exact manipulation of the data required to transform it from its original format to that of its target table.

TABLE

2.3.2 Clean

After we have populated our target table, we have created a various jobs to clean the data from the source and consolidate them. Staging table contains mostly raw text values and numeric only for amounts. Content of the table mostly matches information fromthe source.

Goal of these jobs more specifically:

- Cleanse DUNS number, zip code
- Cleanse all columns of unwanted special characters (+,#,@,!)
- Replace all foreign entities with a FRGN flag, there are lot of cases when the place of performance of agreements has been outside of United States of America. In most cases these values where mostly incorrect or have been unknown.
- Correct state names. There are cases when the state names for United States of America are given only by their abbreviations.
- Geography consolidation

Geography Consolidation

Table with geography country names sometime contain only their country codes. For those cases a mapping table have been created where we have specified a mapping of country codes to their

valid country names.

The process is depicted in the following image:

IMAGE

2.3.3 Load

After extracting, cleaning and transformation finally it's time to populate our dimension tables and fact table. First we have created the dimensions, we assign every dimension table a surrogate key, we have added a EXCEPT statement into our INSERT statement for a reason of avoiding duplicates to be inserted into the dimension tables.

This step consisted of the following creates and loads of all structures for analytical processing.

- fact table - fact is transcation
- dimensions: * geography * agency * recipient * date * type of transactions * award

Dimensions

Examples how the dimensions have been created:

Geogprahy Dimension

Listing 2.3: Geography Dimension - SQL

```
CREATE TABLE dm_geography (
    id SERIAL PRIMARY KEY,
    state text,
    city text,
    zip text,
    country text
);
```

Recipeint Dimension

Listing 2.4: Recipient Dimension - SQL

```
CREATE TABLE dm_recipient (
    id SERIAL PRIMARY KEY,
    name text,
    streetaddress text,
    state text,
    city text,
    zip text,
    country text,
    duns text
);
```

Fact Table

The initial situation was that we should have built a data warehouse, which is able archive large data by monthly basis. We have also known that the information is needed for various reports on weekly, monthly, three months periods, and so on. For this purpose of better query performance we have broke down the fact table into partitions. PostgreSQL supports partitioning via table inheritance [https://www.postgresql.org/docs/9.1/static/ddl-partitioning.html]. We have made the partitioning in a such a way that every child table inherits from a single master table.

The partitioning have been implemented in the following way:

1. We created the master fact table
2. We created the child fact tables, which inherits the master fact table and added checks for dates, because we wanted to ensure that we have only right data on each partition.Partitions starts from 2015-01-01 and ends 2015-02-01. Each partition contains one month data.

Listing 2.5: Child fact table

```
CREATE TABLE ft_spending_2_15M01(
    PRIMARY KEY (id, last_modified_date),
    CHECK ( last_modified_date >= DATE '2015-01-01' and last_modified_date < DATE
) INHERITS (ft_spending_2);
```

3. We created indexes to child fact tables to speed up day field usage, because almost all queries are triggered on the date field.

Listing 2.6: Child table indexes

```
CREATE INDEX idx_15M01 ON ft_spending_2_15M01 (last_modified_date);
```

4. We wanted our data warehouse to be able to say INSERT INTO *spending* and have the data be redirected into the appropriate partition table. We have arranged this by creating a suitable trigger function to the master table.

Listing 2.7: Trigger Function

```
CREATE OR REPLACE FUNCTION ft_partition_trigger()
RETURNS TRIGGER AS $$
BEGIN
  IF
    (NEW.last_modified_date >= DATE '2016-01-01' and NEW.last_modified_date < DATE
    INSERT INTO ft_spending_2_16M01 VALUES (NEW.*);
    ELSIF (NEW.last_modified_date >= DATE '2016-02-01' and NEW.last_modified_date
    INSERT INTO ft_spending_2_16M02 VALUES (NEW.*);
    ELSIF (NEW.last_modified_date >= DATE '2016-03-01' and NEW.last_modified_date
    INSERT INTO ft_spending_2_16M03 VALUES (NEW.*);
    ...
    ELSIF (NEW.last_modified_date >= DATE '2015-11-01' and NEW.last_modified_date
```

```
    INSERT INTO ft_spending_2_15M11 VALUES (NEW.*);
    ELSIF (NEW.last_modified_date >= DATE '2015-12-01' and NEW.last_modified_date
    INSERT INTO ft_spending_2_15M12 VALUES (NEW.*);
    ELSE
      RAISE EXCEPTION 'DATE OUT OF RANGE!';
  END IF;
  RETURN NULL;
END;
$$
LANGUAGE plpgsql;
```

After creating the trigger function, we created a trigger which calls the trigger function.

Listing 2.8: Trigger

```
CREATE TRIGGER insert_ft_spending_2
  BEFORE INSERT ON ft_spending_2
    FOR EACH ROW EXECUTE PROCEDURE ft_partition_trigger_2();
```

With all of these steps we ensured the our master fact table is available and all UPDATEs. INSERT's, SELECT's and DELETE's goes to the right child tables by date.

Finally it was time to populating the fact table. The fact table has been created simply by transforming cleansed data and joining with prepared dimensions.

2.4 Dimensional Modeling

2.4.1 Business Process

The first step in the design is to decide what business process to model by understanding of the business requirements with an understanding of the available data. In our open government case study, the general public wants to better understand how the government spends money, what kind of transactions are made in their neighbourhood. Thus our process is a transactional model. This transactional data will allow us to analyse what kind of awards are made by federal agencies in which states on what days and to whom. Brief description of business model that we'll use in our case study to make dimension and fact tables more understandable. Imagine a federal agency for example Department of Agriculture making a award for a recipient 1901 Combine Group, LLC for a combine harvester in Texas on 2015. To summarise it WHICH federal agency is awarding WHOM for WHAT and WHERE is the place of the performance of the transaction made.

2.4.2 Declare the Grain

Once the business process has been identified. we faced a serious decision about the granularity of the data warehouse. What level of data detail should be made available in the dimensional model? After identifying the data, we had couple of options to choose. We wanted tackling the data at it's lowest level, most atomic grain made the most sense. The more detailed and atomic the fact measurement, the more things we know for sure about federal awards. In this regard, atomic data was the perfect match for the dimensional approach. Atomic data provides the maximum analytic flexibility because it can be constrained and rolled up in every way possible. In our case study, the most granular data is an individual transaction made by federal agencies. Because of this level of grain we ensured maximum dimensionality and flexibility. Providing access to the transactions information gave us very detailed look at federal award changes. For example, the end users want to see how many transaction were made for one individual award or how the award has changed over period of time, if the agency made a modification to an award, reduced a portion of the original award amount or made additional funding.None of them could have been answered if we wouldn't elected the lowest granularity just the summarised data.

2.4.3 Choose the Dimensions

After we have declared the grain of the fact table, the recipient, agency, date, geography, award, dimensions fall out immediately. We assume that the calendar date is the date when the award was signed.

In our case study we have decided on the following dimensions:

Dimension tables are not in third normal form. A dimensional model serves a different purpose from ER model. It wasn't necessary to isolate repeating values in an environment that doesn't support transaction processing. If we would have made additional normalisation within dimensions, we would end up with the schema that is referred as a snowflake. We have encouraged to resist the urge to snowflake given our to primary design, ease of use and performance.

- Snowflaked tables makes for much more complex presentation.
- Database design will struggle with the complexity of the snowflaked schema.
- Numerous tables and joins usually translate into slower query performance.
- Minor disk space savings.
- Snowflaking slows down the user's ability to browse within the dimension.

Dimension tables also contain key columns that uniquely identify something in an operational system. These key columns are referred to as natural keys. The separation of surrogate keys and natural keys allows the data warehouse to track changes, even if the originating operational system does not.

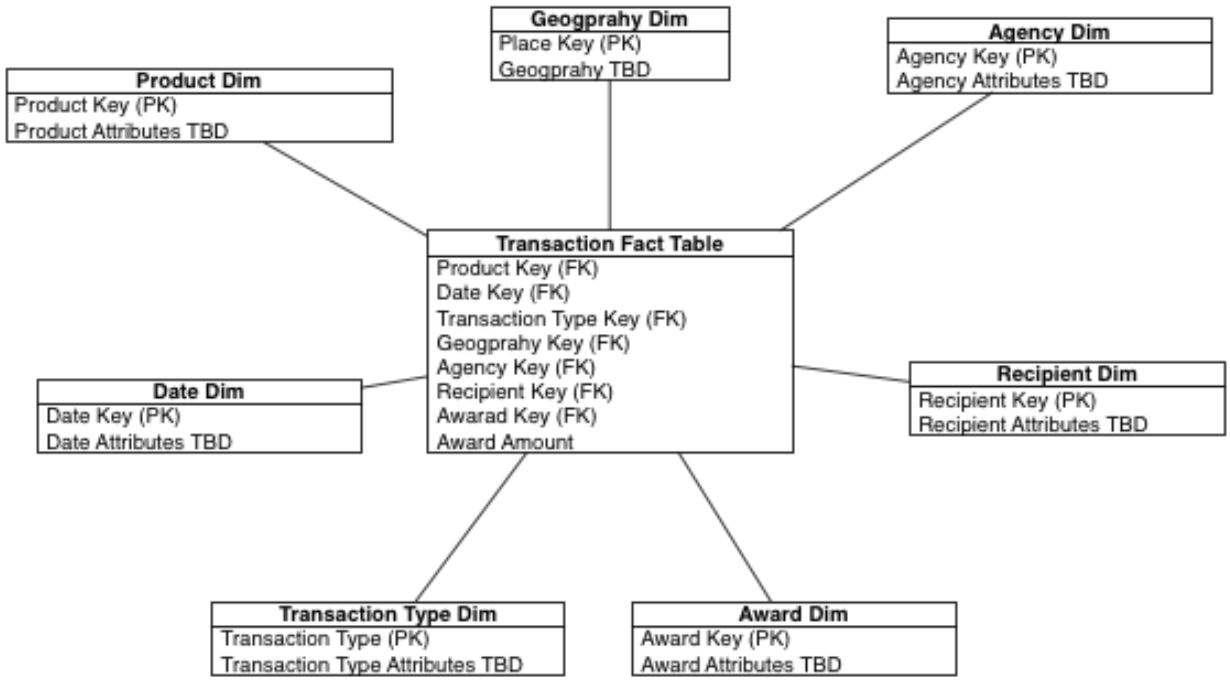

Fig. 2.4: Preliminary star schema.

2.4.4 Identify Facts

At the core of a star schema is the fact table. In addition to presenting the facts, the fact table includes surrogate keys that refer to each of the associated dimension tables. Each row in the fact table stores facts at a specific level of detail of our grain that we have declared. Facts tend to be numeric in value. We have made the decision that the award amount is going to be our fact measurement which will appear in our fact table. We have decided to stored physically in the data warehouse only one fact the award mount, which is additive across all dimensions.

2.4.5 Suroggate Keys

In the star schema, each dimension table is given a surrogate key. This column is a unique identifier, created exclusively for the data warehouse. The surrogate key is the primary key of the dimension table. In our case, surrogate keys are randomly generated integers that are assigned sequentially when populate dimension tables during the ETL process. For example, the first recipient record is assigned a recipient surrogate key with the value of 1, the next recipient record is assigned recipient key with value 2, and so forth. The surrogate keys serve to join the dimension tables to the fact table. One of the most important reasons why are we using surrogate keys and doesn't just rely on natural keys from the source system is to support handling changes to dimension table attributes.

2.5 Dimensional Table Attributes

2.5.1 Date Dimension

We have started with the date dimension, which is nearly guaranteed to be in every data warehouse. Unlike other dimensions, we could have build the date dimension in advance. We have put 10 - 20 years of rows representing days in the table so that we can cover the history we have stored, as well as several years in the future. From our data we know that first public awards come from the year 2008, so even 10 years worth of days is only about 3650 rows, which is relatively small dimension table.

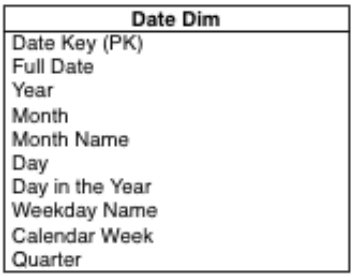

Fig. 2.5: Date Dimension

Each column in the date dimension table is defined by the particular day that the row represents. The full day columns represents the full date when the award have been signed in format YYYY-MM-DD it data type is date. The year columns represents the fiscal year of the award and it is a singe integer type. This column is useful for slicing, when we want filter a particular year, for example we want to show all the transactions made in the year of 2015. The month column represents the month number in the year (1, 2, . . . , 12). The month name columns represents the name of the month (January, February etc.). Similarly, as month we have a day column, which represents the day number in calendar month columns starts with 1 at the beginning of each month an it is depending on the particular month. The day in the year columns is a single integer which is representing the day number in the year, it starts from 1 and runs to 365, depending on what year is. The weekday name column represents the name of the day when the award have been signed, such as Wednesday. It has string data type. We have also included calendar week column. We have included quarter number (Q1,. . . , Q4). Columns like year, quarter, month, calendar week, day all these integers support as simple date filters. Figure 2.5 illustrates several rows from a partial date dimension table.

2.5.2 Product Dimension

The product dimension describes every product or service for a particular award. While a typical reasonable product dimension table would have 50 or more descriptive attributes, in our data warehouse we have only one product name attribute column. It is sourced from the operational

id	full_date	year	month	month_name	day	day_year	weekday_name	calendar_week	quarter
1	2000-01-01	2000	1	January	1	1	Saturday	52	Q1
2	2000-01-02	2000	1	January	2	2	Sunday	52	Q1
3	2000-01-03	2000	1	January	3	3	Monday	1	Q1
4	2000-01-04	2000	1	January	4	4	Tuesday	1	Q1
5	2000-01-05	2000	1	January	5	5	Wednesday	1	Q1
6	2000-01-06	2000	1	January	6	6	Thursday	1	Q1
7	2000-01-07	2000	1	January	7	7	Friday	1	Q1

Fig. 2.6: Sample of Date Dimension

Product Dim
Product Key (PK)
Product Name

Fig. 2.7: Product Dimension

system as a single attribute column with no hierarchy. The only function of the product dimension is to give a descriptive attribute of each award. In our case there is only 2492 different values in the product dimension table. Viewed in this manner, we can only drill down on one level of the product dimension which provides us information about the award amount and quantity by product name. For example here is simple report overview, we have summarised the award amount and quantity (count) by product name.

Product Name	Count	Amount
ammunition and nuclear ordnance boxes, packages and special containers	31	$ 207244.6300
construct/water supply	5	$ -4820.0000
r&d- energy: gas (applied research/exploratory development)	1	$ 9850.0000
medical- surgery	324	$ 24519499.3100
minerals, natural and synthetic	38	$ 561156.4700
construction of open storage facilities	15	$ 280345.0500
maint-rep-alt/r&d gogo facilities	1	$ -2297.4100
r&d- defense other: ammunition (engineering development)	7	$ 5300000.0000
quality control- aircraft components and accessories	6	$ 865716.2400
special studies/analysis- elderly/handicapped	1	$ 2000.0000

Fig. 2.8: Drill down sample of Product Dimension

2.5.3 Geography Dimension

The geography dimension describes every transactions places of performance for recipients. It is primary a geographic dimension in our case study. Each row can be thought of as a location where

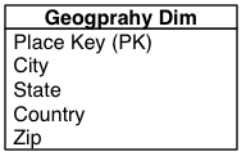

Fig. 2.9: Geography Dimension

an award has been made. Because of this we, can drill down / roll up any geography dimension attribute, such as country, state, city, zip code. These columns attributes are representing in the geography dimension a simple hierarchy for a single row.

2.5.4 Recipient Dimension

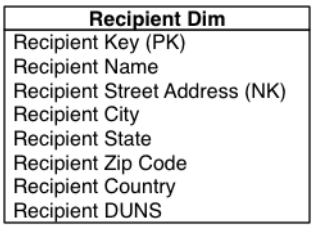

Fig. 2.10: Recipient Dimension

The recipient dimension describes recipient transactions. The dimension contains row for each recipient, along with descriptive attributes such as street address, state, city etc. We capture only the recipient name, address information and DUNS number. It is used to establish a business credit file, which is often referenced by lenders and potential business partners to help predict the reliability and/or financial stability of the company in question [http://www.dnb.com/duns-number.html]. Recipient geographic attributes have been complicated to dealing with. We wanted to avoid snowflaking as we have mentioned in Choose Dimension chapter. In our model recipients typically have multiple addresses, but every transaction is geocoded on the prime recipient address. This means we have unique row for each recipient based on his geographic attributes. The street address column is our natural key to identify each unique recipient.

A sample set of name and location attributes for individual recipient:

1	Abbe-wood	1002 west 23rd st.	Al-abama	Panama City	32405	United States of America	627189244

2.5.5 Transaction Type Dimension

The transaction type dimension describes the type of award. It is our smallest dimension consists only of 4 unique row. The transaction status columns describe the status of the transaction active/inactive. The category columns is our key attribute in the dimension which describe what kind

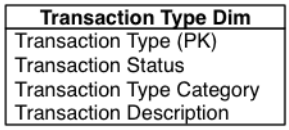

Fig. 2.11: Transaction Type Dimension

of transaction is reported by the federal agencies making contract, grant, loan, and other financial assistance award.

- Contract is an agreement between the federal government and a prime recipient to execute goods and services for a fee.
- Grant is type of federal award that requires an application process and include payments to non-federal entities for a defined purpose in which services are not rendered to the federal government.
- Loan is type of federal awards that the borrower will eventually pay back to the government.
- Other Financial Assistance includes direct payments to individuals, insurance payments, and other types of assistance payments.

The transaction description columns provides us descriptive information of types of awards.

2.5.6 Award Dimension

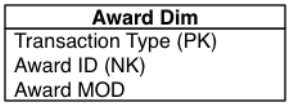

Fig. 2.12: Award Dimension

The award dimension is potentially the most important dimension in our schema. The grain of the fact table has been stated as "awards at the award modification level of detail". This has been achieved by adding award identifiers from the source system to identify individual awards: the award_id and award_mod. Together with this two attributes we have achieved to uniquely identify each fact table rows. Because of this approach, each fact table row represents exactly one award or award modification, the award dimension and the fact table contain the same number of rows.

Slowly Changing Dimension

As our grain dictates, we need to track changes over time. The award modification number is are natural key identifier carried over from the source system. The award modification number uniquely identify a corresponding entity in our source system. For example, an individual award is identified by award id in our source system, which uniquely identify a contract or agreement. With the combination of the natural key award mod from the source system we have achieved to identify each award modification. The use of surrogate key allows the data warehouse uniquely identify

each transaction and respond to changes in the source system. This allows us to track history changes of awards. With these dimension attributes we have specified a strategy to handle change. In other words, when an attribute value change in the operational system, we will respond to this change in our dimensional model. We refer this kind of change as slowly changing dimension. We made the claim earlier about our goals of the data warehouse was to represent awards correctly. Thats why we have decided to use a technique called SCD Type 2. Using the type 2 approach, when an agency make a modification to an award (for example restrict the original award or make additional funding) we create a new award dimension row for the given award to reflect new award amount.

1	1PIC355199	c5e18f14-82cf-4e64-b1b0-cd767735e330
2	1PIC355199	c8130052-2d0b-41f2-9b2b-84f0f3da9d18
3	1PIC355199	c8c24821-38fe-463f-9fe5-2f3f6db054fc
4	1PIC355199	cc809f09-595b-455a-af92-3498a3c97a90
5	Geography	cd4f8125-a714-4565-8d72-4700f522d54a

Now we can see why the the award dimension key can't be the Award ID natural key. We need different surrogate keys for the same award id. Each surrogate keys identifies a unique transaction that was true for a span of time. This method accurately track slowly changing dimensions attributes.

2.5.7 Federal Award Transactions

As we have moved from the dimensional design, this is how our federal award transaction business model looks like for now.

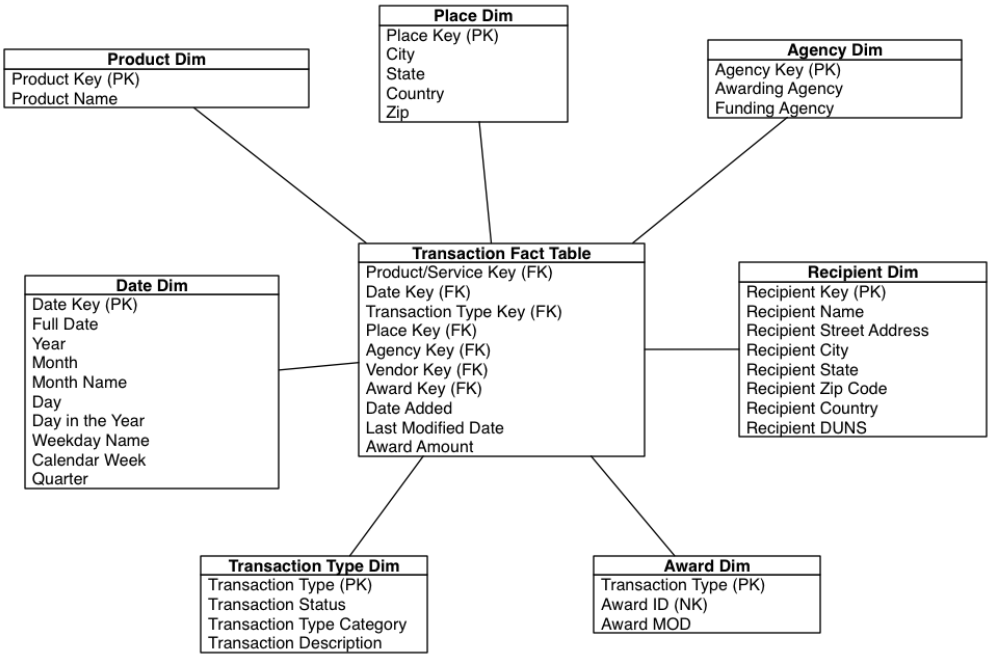

Fig. 2.13: Star Schema

CHAPTER 3

OLAP

The presentation area is based on online analytic processing (OLAP) technology, the data is stored in cubes.

3.1 Cubes - OLAP Framework

Cubes is a light-weight Python framework and set of tools for development of reporting and analytical applications, Online Analytical Processing (OLAP), multidimensional analysis and browsing of aggregated data *[Cubes]*.

The framework models the data as a cube with multiple dimensions:

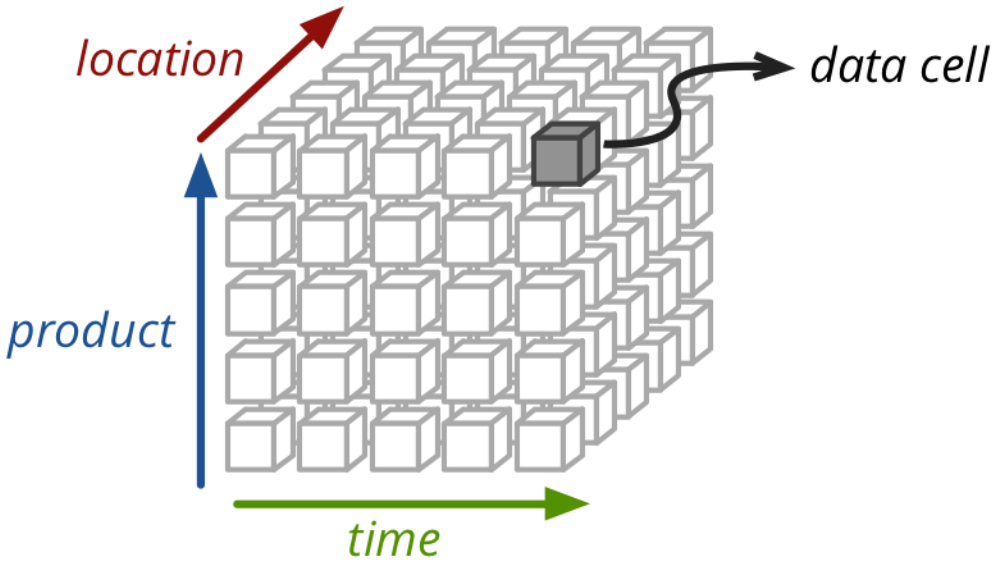

Fig. 3.1: Cubes Dimension and Cell

Core cube features that we have implemented:

- **Workspace** - Cubes analytical workspace
- **Model** - Description of data (metadata): cubes, dimensions, hierarchies, attributes, labels etc.
- **Browser** - Aggregation browsing, slicing-and-dicing, drill-down.
- **Backend** - built-in ROLAP backend which uses Postgres SQL database using SQLAlchemy.
- **Server** - WSGI HTTP server for Cubes

3.2 Analytical Workspace

Everything in Cubes happens in an analytical workspace. It contains cubes, maintains connections to the data stores (with cube data), provides connection to external cubes and more *[workspace]*.

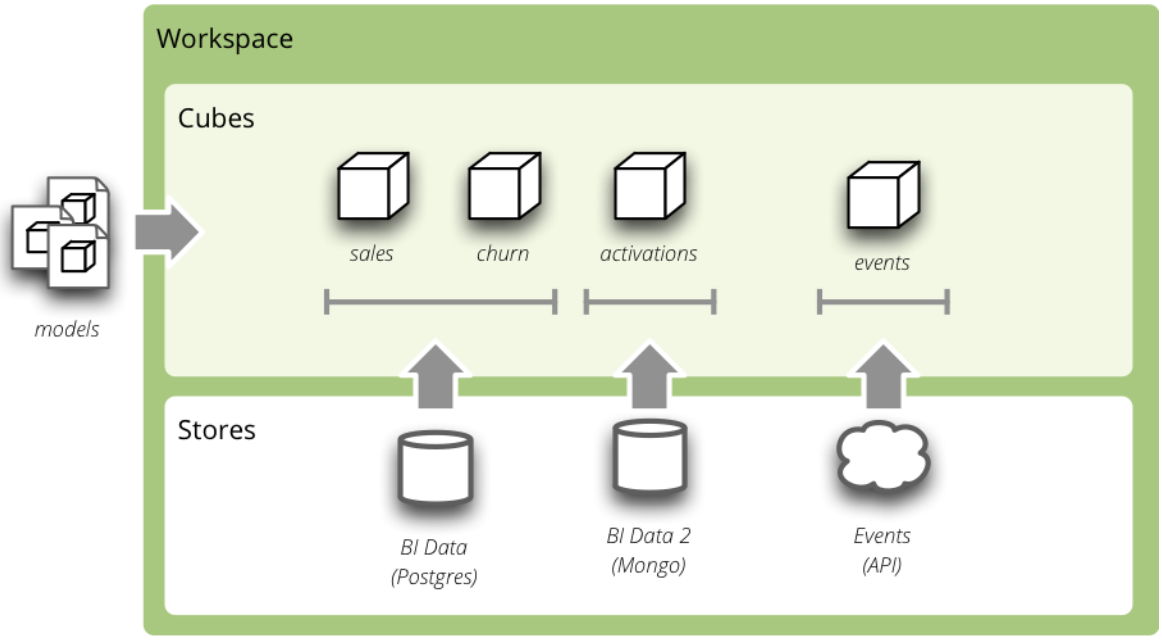

Fig. 3.2: Cubes Workspace

The workspace properties are specified in a configuration file *slicer.ini*. First we have specified the workspace - cubes workspace configuration. We have provided the workspace our *model.json* metadata dictionary. We have added a logging path where to log file for better understanding of debug logs.

Listing 3.1: slicer.ini

```
[workspace]
log: error.log
```

```
log_lever: debug
model: model.json
```

For the server section we have defined the HTTP WSGI OLAP Server. The host where the server runs. Port on which the the server listens. We have allowed Cross-origin resource sharing header for the CubesViewer.

Listing 3.2: slicer.ini

```
[server]
host: localhost
port: 5000
prettyprint: true
reload: true
allow_cors_origin: http://localhost:8000
```

We have specified our data store - the database containing the cube's data. We have defined the prefix for dimension *dm_* and for fact *ft_* this allows the mapper to find dimension and fact tables in the database.

Listing 3.3: slicer.ini

```
[store]
type: sql
url: postgresql://richardbanyi:dpcu923@localhost:5432/usaspending
fact_prefix: ft_
dimension_prefix: dm_
debug: true
```

The model section contains list of models. We have defined only one model.

Listing 3.4: slicer.ini

```
[models]
main: model.json
```

3.3 Logical Model and Metadata

Logical model describes the data from user's or analyst's perspective: data how they are being measured, aggregated and reported. Model is independent of physical implementation of data. This physical independence makes it easier to focus on data instead on ways of how to get the data in understandable form *[model]*.

Analyst or report writers, anyone who will access the web application do not have to know where name of an government organisation or recipient name is stored, nor do they have to know where

is data stored they only ask for *agency.name* or *recipient.name*.

Example: User wants to find out the award amounts by geography which has four levels:country level, state level, city level and zip level. The original data is stored in the physical database in the geography table. User doesn't have to know where the data are stored, he just queries for the geography.state and geography.city and will get the proper data.

3.4 Model

The logical model is described using model metadata dictionary. The content is description of logical objects, physical storage and other additional information [9].

Listing 3.5: Model Example

```
{
    "name": "usaspending",
    "label": "Federal Award Transactions",
    "description": "Federal Award Transactions of United States of America"
    "cubes": [...]
    "dimensions": [...]
}
```

In our case the logical part of the model description consists of the following attributes: **Name** is the name of our model, **label** and **descriptions** is used for human readable label/descriptions it's optional. **Cubes** is a list of cubes metadata, we have defined only one cube for our case study spending. **Dimensions** is a list of dimension metadata.

The physical part of the model description consists of the following attributes:

Joins we have only specified backend join specification, it is used to match joins in the cube.

3.4.1 Cubes

Cube descriptions are stored as a dictionary for key *cubes*.

Listing 3.6: Cubes Example

```
  {
      "name": "usaspending",
      "label": "Federal Award Transactions",
      "dimensions": [ "date", ... ],
      "measures": [...],
      "aggregates": [...],
```

[9] Author: Stefan Urbanek, Date: 2010-2014, Online: Model metadata Availailable: Cubes Model Metadata

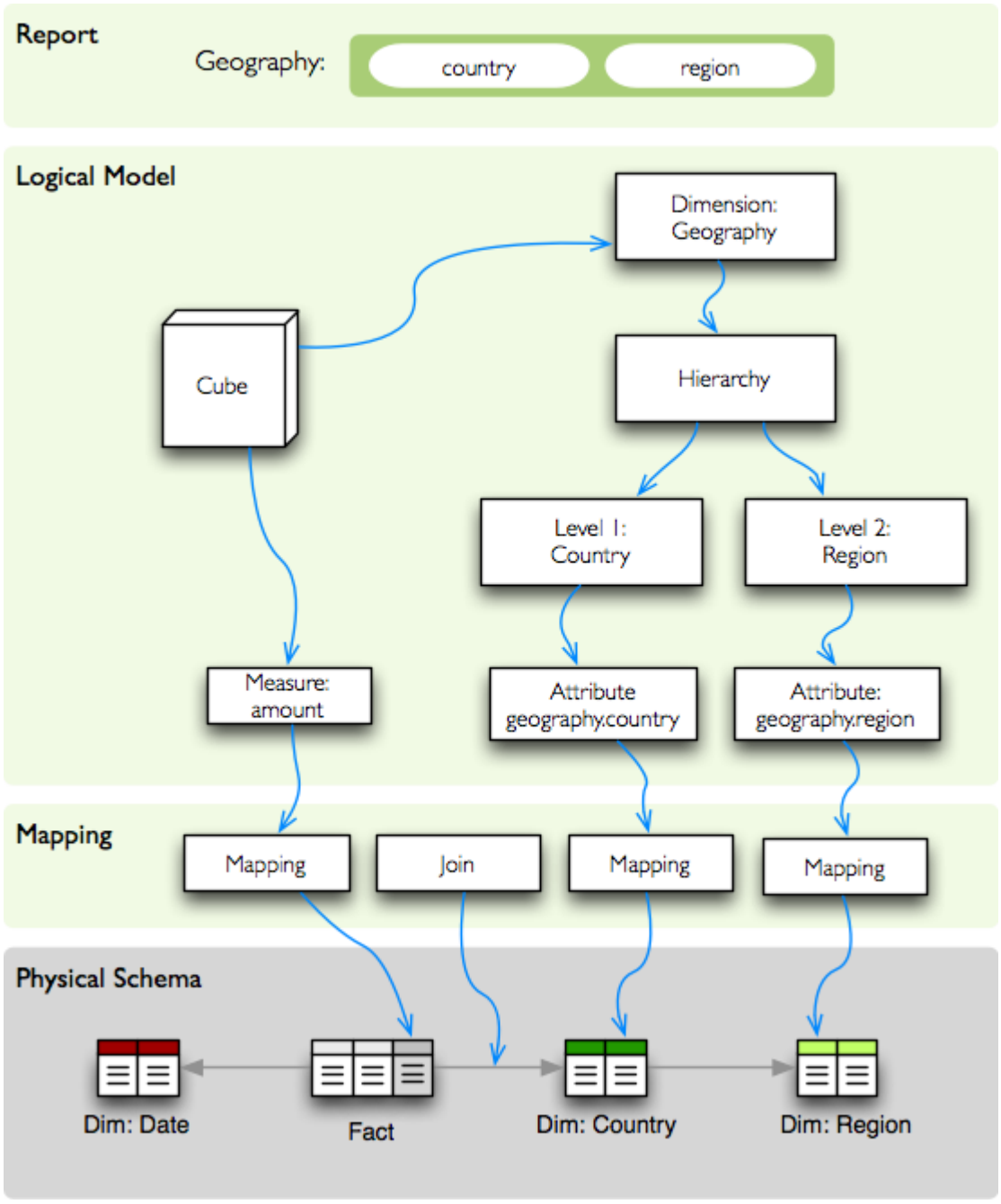

Fig. 3.3: Logical to Physical Mapping

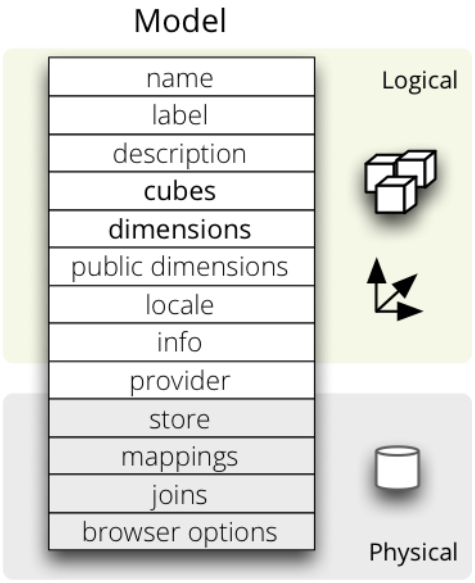

Fig. 3.4: Cubes - Model Metadata

```
      "details": [...],
  }
```

3.4.2 Measures and Aggregates

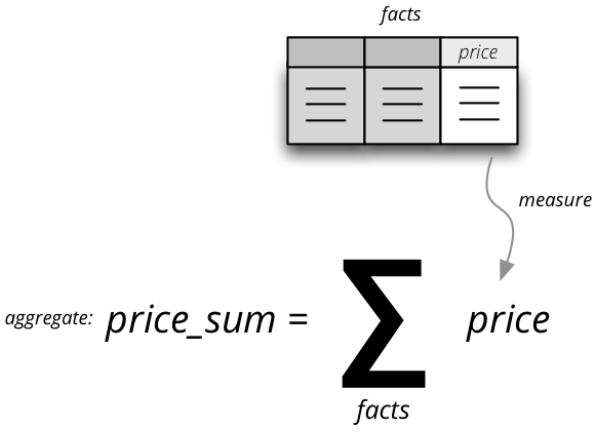

Fig. 3.5: Cubes - Measures vs. Aggreates

The measure is numerical property of a fact. It represent the award_amount column in the physical model.

Listing 3.7: Measures

```
"measures": [
   { "name": "award_amount" , "label": "Award amount"}
 ]
```

Aggregates is a list of aggregates that we have provided for the measure.

Listing 3.8: Aggregates

```
 "aggregates":  [
   {
       "name": "award_sum",
       "label": "Total Award Amount",
       "function": "sum",
       "measure": "award_amount"
   },
   {
       "name": "transaction_count",
       "label": "Total Transactions",
       "function": "count"
   },
   {
       "name": "award_min",
       "label": "Min Amount",
       "measure": "award_amount",
       "function": "min"
   },
   {
       "name": "award_max",
       "label": "Max Amount",
       "measure": "award_amount",
       "function": "max"
   }
 ]
```

Note that item_count aggregate - it counts number of the facts within the cell. No measure required as a source for the aggregate. It helps to count the number of the transactions.

3.4.3 Mappings

The most important part of the OLAP on top of the start schema is mapping of the logical attributes to their physical attributes. In SQL database the physical attributes are stored in columns, which belongs to tables, which are part of the starcshema.

Example:

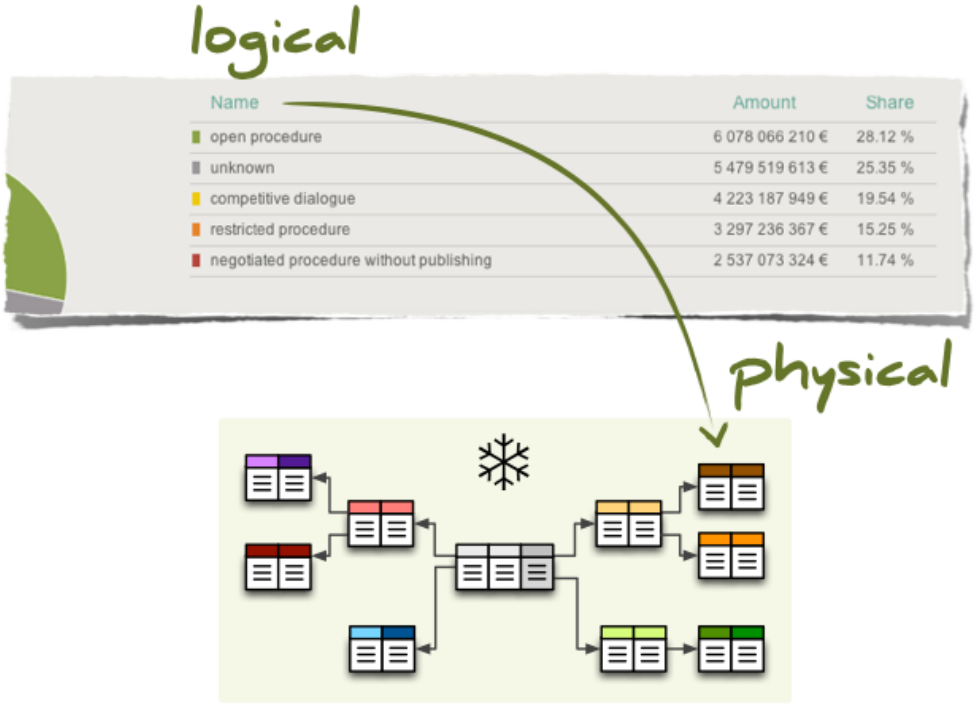

Fig. 3.6: Cubes - Mapping Logical to Physical

For data browsing, we have defined mappings, so that Cubes framework has know where logical attributes are physically stored. With this the Cubes framework know which tables are related to the cube spending and how they are joined together.

The are two ways how the mapping is being done: * Imlicit * Explicit

We have chosen the implicit declarations which has been most straightforward and simple. With implicit mapping we have matched a database schema with logical model and have avoided additional specific mapping metadata.

This is how it would looked like if we would have chosen explicit mappings:

Listing 3.9: Explicit Mapping

```
"mappings": {
   "product.name": "dm_products.product_name"
}
```

The mapping process look like this:

3.4.4 Facts

Cubes looks for fact table with the same name as cube name. We have specified prefix for every fact table with *ft_* in the workspace configuration *slicer.ini*.

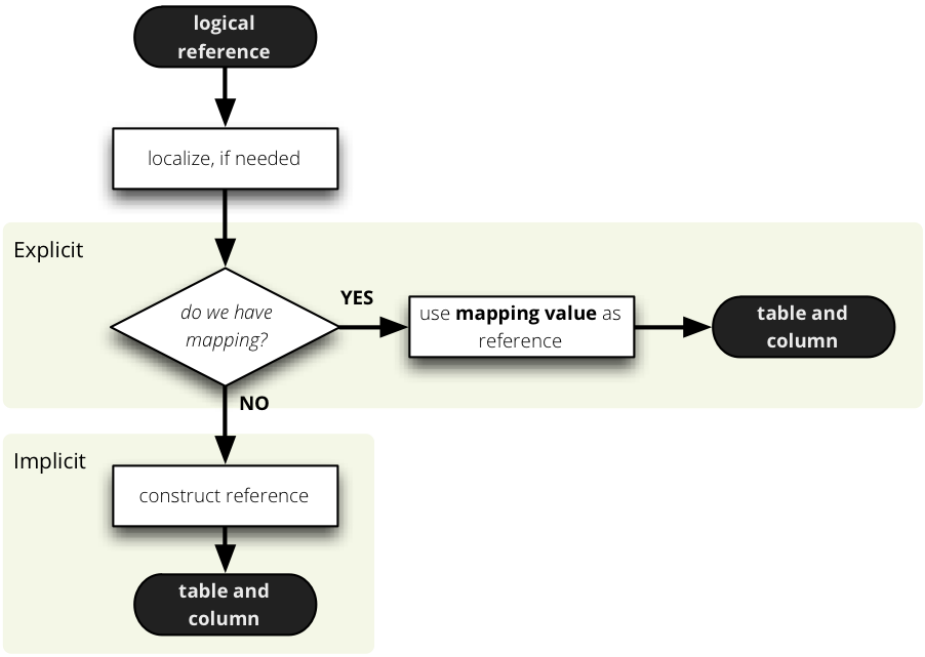

Fig. 3.7: Cubes - Mapping Overview

- Cube is named *spending*: the framework looks for a fact table named *spending*.
- **fact table name = fact table prefix + fact table name**

3.4.5 Dimensions

Same as for fact tables cubes looks for dimension table with the same name as dimension name. We have specified prefix for every dimension table with *dm_* in the workspace configuration *slicer.ini*. We have design our dimension columns to have the same name as dimension attributes.

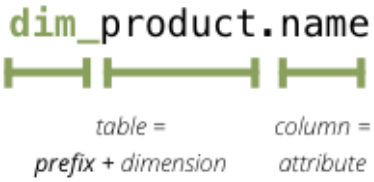

Fig. 3.8: Cubes - Dimension attribute prefix

- **dimension table name = dimension prefix + dimension name**

An example of product dimension:

Listing 3.10: Product dimension

```
{
  "name": "product",
  "label": "Product",
  "attributes": [
      { "name": "id" },
      { "name": "product_name" }
  ],
  "levels": [ ... ],
  "hierarchies": [ ... ]
}
```

3.4.6 Joins

Tables are joined by matching single-column - surrogate keys. Joins are defined as an ordered list. We had to define the column reference for both master table and a table with details. The order of joins has to be from master to detail.

Listing 3.11: Join example

```
"joins":  [
    {
        "master": "date_id",
        "detail": "dm_date.id"
    },
]
```

Cubes supports three join methods *match, detail and master*.

match (default) – the keys from both master and detail tables have to match – INNER JOIN

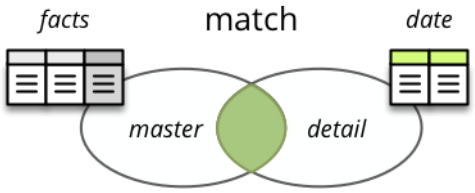

Fig. 3.9: Cubes - Join Match

3.4.7 Hierarchies and Levels

Dimensions can have more than one level. Date dimension has 8 levels. Which every of these levels has it's own attributes. The month level is represented by two attributes month an integer

data type which describes the mont of the year and second attribute month name.

Listing 3.12: Levels - Month

```
"levels":  [
    {
        "name": "month",
        "label": "Month",
        "key": "month",
        "label_attribute": "month_name",
        "attributes":  [
            { "name": "month" },
            { "name": "month_name" }
        ]
    }
```

For date dimension we have created multiple ways of organising attributes into hierarchies. The date can be composed of *year-month-day* or *year-quarter-month-day*. We have defined four different hierarchies.

First we have defined all possible level.s Then created list of hierarchies where we specified order of the levels for the particularly hierarchy.

Listing 3.13: Hierarchies - Year-Month-day

```
"hierarchies":  [
    {
        "name": "ymd",
        "label": "Y-M-D",
        "levels": ["year", "month", "day"]
    }
]
```

3.4.8 User-oriented Metadata

For a better understanding to users we have added labels for parts of the model that are being displayed to the users. They are used for report tables as column headings or as a filter of description. We have specified them for every object model (cube, dimension, level, attribute etc.) with the label attribute. The *key* attribute is used for filtering and *label_attribute* is used for the data to be displayed in the user interface (labels).

Listing 3.14: Labels

```
"levels":  [
    {
        "name": "month",
```

```
            "label": "Month",
            "key": "month",
            "label_attribute": "month_name",
            "attributes":  [
                { "name": "month" },
                { "name": "month_name" }
            ]
        }
```

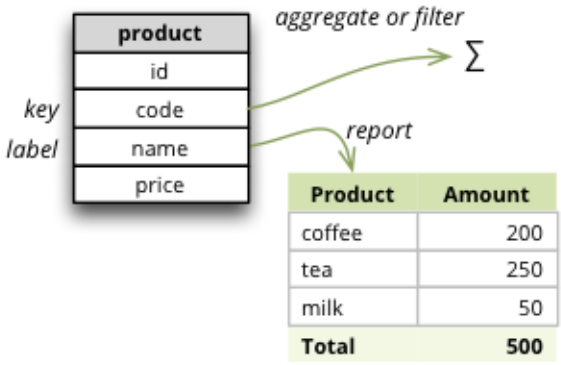

Fig. 3.10: Cubes - Labels

3.5 Slicer Server

Cubes framework provides a WSGI server which covers most of the Cube logical model metadata and aggregation browsing functionality. We have already configured the server at beginning of the OLAP section.

To get a version of the server we just simple request a GET request.

Request *GET /version*.

Listing 3.15: Slicer Server - GET /version

```
{
    "version": "1.1",
    "api_version": 2,
    "server_version": "1.1"
}
```

To get a list of information about served cubes.

Request *GET /cubes*.

Listing 3.16: Slicer Server - GET /cubes

```
[
    {
        "label": "spending",
        "category": null,
        "name": "spending",
        "info": {}
    }
]
```

Aggregation and Browsing

The core data and analytical functionality is accessed thought the following requests:

- */cube/<name>/aggregate* - aggregate measures, summary, generate drill-down, slice&dice.
- */cube/<name>/members/<dim>* - list of dimension members.
- */cube/<name>/facts* - list facts within cell.
- */cube/<name>/fact* - return a single fact.
- */cube/<name>/cell* - describe cell.

The cells - part of the cube we are interested in are specified by cut. The cut in the URL are given as a parameter cut.

For example:

- *cut=date:2015*
- *cut=geography:usa,california*

Aggregate

The server always returns the aggregation results as JSON. The result contain keys: summary - represents the aggregation of the whole cell specified in the cut aggregates - list of aggregate names that are considered in the aggregation cell - list of dictionaries describing the cell cuts.

Listing 3.17: Slicer Server - Aggregate

```
{
    "summary": {
        "award_sum": 275669527432.41,
        "award_max": 9999999999.0,
        "award_min": -1604823842.0,
        "transaction_count": 1550767
    },
    "remainder": {},
    "cells": [],
    "aggregates": [
```

```
            "award_sum",
            "transaction_count",
            "award_min",
            "award_max"
        ],
        "cell": [
            {
                "type": "point",
                "dimension": "date",
                "hierarchy": "ymd",
                "level_depth": 1,
                "invert": false,
                "hidden": false,
                "path": [
                    "2015"
                ]
            }
        ],
        "attributes": [],
        "has_split": false
    }
```

3.6 Slicing and Dicing

3.6.1 Browser

The aggregation, slicing, dicing, browsing of the multi-dimensional data is being done by an AggregationBrowser.

Listing 3.18: Workspace & Browser initilization

```
from cubes import Workspace

workspace = Workspace("slicer.ini")
browser = workspace.browser()
```

Here we initialised our workspace, and create and initialises the aggregation browser.

3.6.2 Cells and Cuts

Cell defines a point of interest – portion of the cube to be aggergated or browsed.

There are three types of cells: point – defines a single point in a dimension at a particular level;

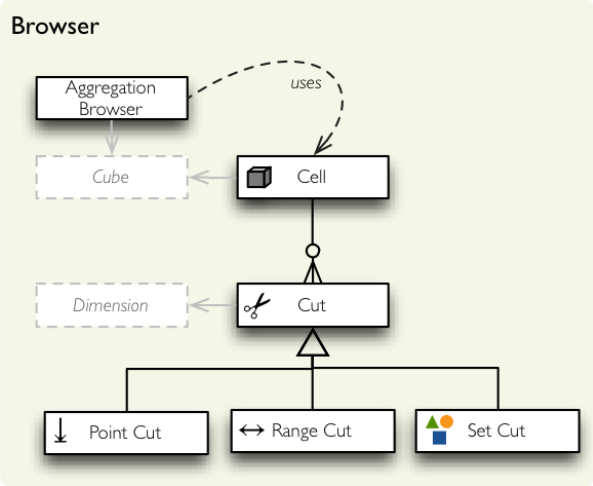

Fig. 3.11: Cubes - Browser aggregate

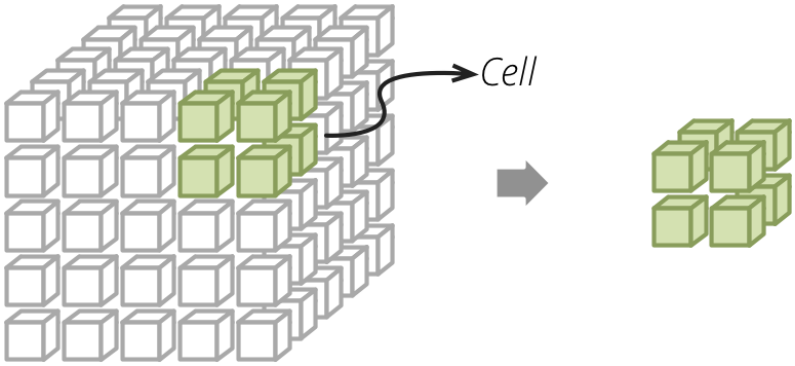

Fig. 3.12: Cubes - Slice and Dice cell

range – defines all points of an ordered dimension (such as date) within the range and set – collection of points *[cells]*.

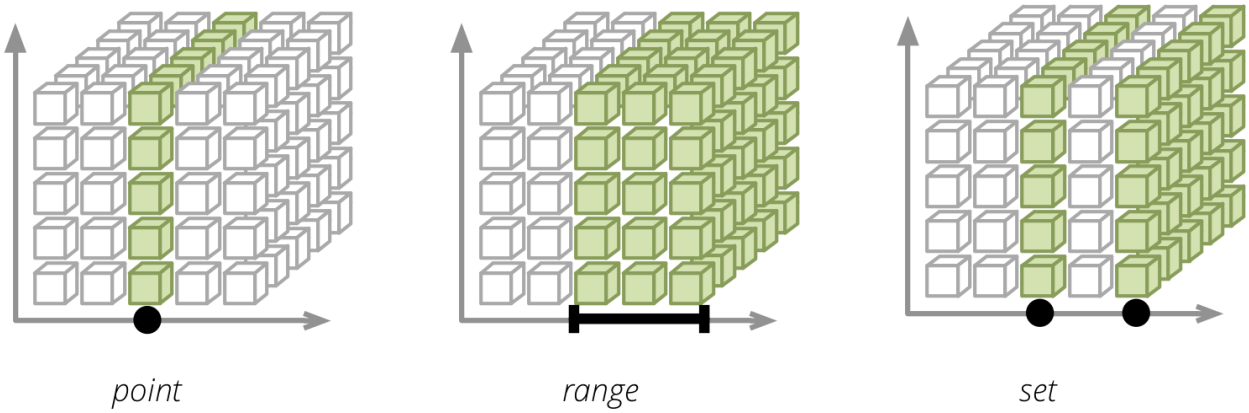

Fig. 3.13: Cubes - Cuts

Points are defined as dimension paths – list of dimension level keys. For example a date path for 3rd of June 2015 would be: [2015, 6, 3]. For the month June it would be [2015, 6] and for the whole year of 2015 it would be [2015]. In python the cuts for "transactions from June of 2010 to June of 2012 for geography United States of America" are defined as:

Listing 3.19: Point cut example

```
cuts = [
   PointCut("geography", ["united states of america"]),
   PointCut("date", [2010, 6], [2012, 6])
   ]
```

A range might look like this:

Listing 3.20: Range cut example

```
cuts = [
    PointCut("date", [2010], [2012, 12, 24])
]
```

3.6.3 Aggregate

Aggregate of a cell:

Listing 3.21: Aggregate of a Cell

```
 cuts = [
     PointCut("geography", ["sk"])
```

```
    PointCut("date", [2010, 6], [2012, 6]),
]
cell = Cell(cube, cuts)
result = browser.aggregate(cell)
```

We have a situation when a different hierarchy is desired than the default, so we have defined the hierarchy we wanted:

Listing 3.22: Aggregate of a Cell with hierarchy

```
cuts = [
    PointCut("date", [2010, 2], [2012, 2], hierarchy="yqmd")
]
```

3.6.4 Drilldown

Drill-down – get more details, group the aggregation by dimension members. For example:

Listing 3.23: Drilldown

```
cut = PointCut("date", [2010])
cell = Cell(cube, [cut])
result = browser.aggregate(cell, drilldown=['date'])
```

Implicit

The way we are drilling down, is that cubes knows the next level of the drilled dimension. We have stated that our geography dimension has 4 levels: country, state, city, zip. This means that the first level of the dimension is country. The "next level" is determined as the next level after the deepest level used in a cut. The next level is by state, and so on. If the cut is at its deepest level (zip), it is not possible to to drill-down deeper.

Explicit

For explicit drilling down, the cut is not considered for the drill down level. We have just implicitly declared the drill-down levels.

3.7 Drill Down Tree

We have created a function that traverse a dimension hierarchy and print-out aggregations (count of transactions and award amounts) at the actual browsed location.

Attributes:

- dimension - dimension to be traversed trough all levels.

```
def drilldown(dim_name=None):
```

First we need to get the dimension hierarchy to know the order of levels. Most dimensions have only one hierarchy, thought.

```
dimension = cube.dimension(dim_name)
hierarchy = dimension.hierarchy()
```

Parser the cut request parameter and convert it to a list of actual cube cuts.

```
cutstr = request.args.get("cut")
hierarchy = dimension.hierarchy()
cell = cubes.Cell(browser.cube, cubes.cuts_from_string(cube, cutstr))
```

We get the cut of actually browsed dimension, so we know "where we are".

```
cut = cell.cut_for_dimension(dimension)

if cut:
    path = cut.path
else:
    path = []
```

Now we do the actual aggregation of the cell.

```
levels = hierarchy.levels_for_path(path)
if levels:
    next_level = hierarchy.next_level(levels[-1])
else:
    next_level = hierarchy.next_level(None)
```

We have created a flag, so that we know that we are at the deepest level.

```
is_last = hierarchy.is_last(next_level)
```

And finally we, render the template

```
return render_template('drilldown.html', dimensions=cube.dimensions, dimension=dime
                       levels=levels, next_level=next_level, result=result,
                       cell=cell, is_last=is_last, details=details, pagination=pag
                       )
```

CHAPTER 4

Web Application

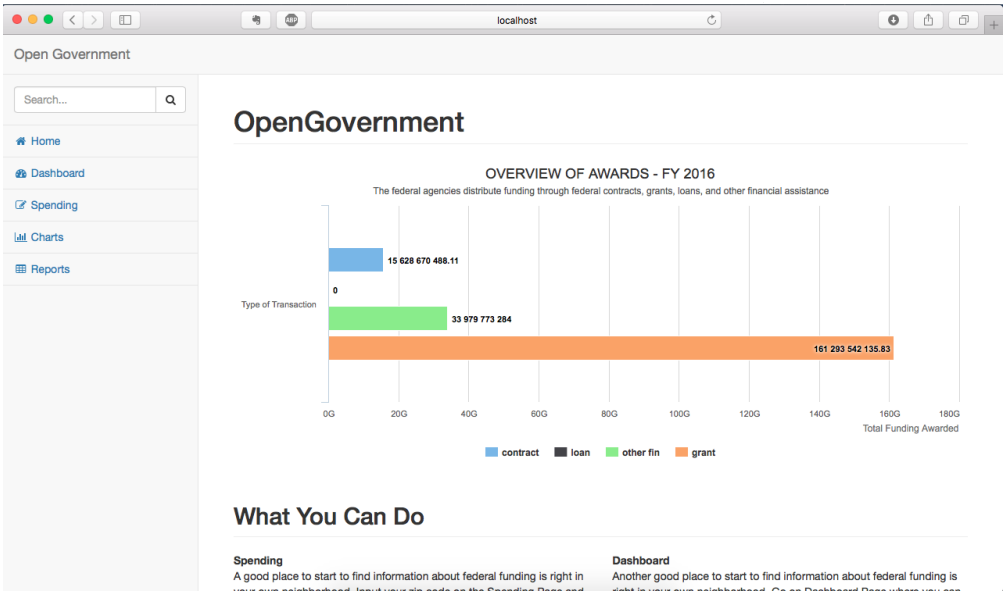

Fig. 4.1: Opengovernment - web application

The final major component of the implementation environment is the web application. We have build a web application that provide variety of capabilities to general users to leverage the presentation area for analytical decision making.

For the web development of the application we have used a micro web development framework for Python called Flask. For the front end we have used bootstrap framework to design the layouts and the interface. We are also relying on some javascript libraries for the charts.

The open government folder is where we have dropped our files. We have put directly in this folder our data warehouse schema, models for cubes, as well as the main application module. The static folder is a place where we have placed css, javascript files. Inside the templates folder the application looks for the templates.

```
/opengovernment
   /static
   /templates
   /postgres
   /models
   flask_app.py
```

The actual application module, we have named it flask_app.py is the core of the back end. This is where the all web development functions are implemented. We have integrated Cubes Slicer Server with the application to provide raw analytical data. We have provided Slicer as a flask Blueprint to the application - a module that is plugged in.

```
if __name__ == '__main__':

    # Create a Slicer and register it at http://localhost:5000/slicer
    app.register_blueprint(slicer, url_prefix="/slicer", config="slicer.ini")
    app.run(host="localhost", port=8000, debug=True)
```

We have implemented the following 7 features in our web application:

- **Spending** - A good place to start to find information about federal funding. On Spending Page users just have to input a zip code which they are interested in and see all the federal transactions in their backyard. They can also see the transactions throughout state, county, as well as by federal agency and award type.
- **Dashboard** - On Dashboard Page we enabled users to drill down from one place to another, they will find information to detailed data by focusing on something. For example to drill down through a series of place where the work was/is performed.
- **Reports** - We have implemented a visual tool for exploring and analysing our OLAP Cube. We have enabled users to explore data, create their own reports. It enables data exploration, data auditory, generation of reports, chart design and simple analytics.
- **Charts** - We have also created our own charts using javascript libraries. On Chart pages users can drill down/roll up donut charts. Or they can look at Agencies Profile to see the total amount of money an agency awarded or State Summary to find out the total dollar amount for federal transactions for cities where the money was distributed.

CHAPTER 5

Conclusion

The bachelor thesis was aimed to implement an open-data data warehouse using open government data. As we have been started through searching for data, we have studied the metadata of the source, the process in detail and observe the transactions of government institutes, such as contracts, grants, loans and other financial assistance, we have decided that the federal award transaction is thebusiness process to be modelled.

Once the business model has been identified, we faced a serious decision about the granularity of the data warehouse. We ensured maximum dimensionality and flexibility by choosing the most granular data - individual transactions made by federal agencies. After we have declared the grain of the fact table, the dimensions fall out immediately.

We have made various transformations on data to enforce data quality, integrity and consistency, conformed data so that separate flat files can be used together, and delivered data into the presentation layer.

Then, we have designed the multidimensional model of the presentation layer, which is based on online analytic processing (OLAP) technology, so that the data is stored in cubes. We have properly specified the analytical workspace of the Cubes. Following we have provided logical model metadata to the workspace.

Finally, the major component of the the implementation was to make it accessible and available to the end users. For this reason, we have build a web application using a micro web development framework for Python called Flask. We have implemented various features for the end users. Visual tool for exploring and analysis, which enables data exploration, data auditory, generation of reports, chart design and simple analytics. By this we have enabled the general public to evaluate the business process of the government.

CHAPTER 6

Resume in Slovak language

CHAPTER 7

References

[1] Authors: School of Data, Organization: School of Data Date: Sep 02, 2013 Available from: Data Fundamentals

CHAPTER 8

Appendix A

8.1 Instalation Manual

In this section of appendinx, user guide to setting up the environment and installation is described. This sections covers all the pre-requireties and procedures for demonstration of the bachelor thesis.

8.1.1 Python

Before we start, we will need Python on your computer, but you may not need to download it. First of all check that you don't already have Python installed by entering python in a command line window. If you see a response from a Python interpreter it will include a version number in its initial display.

Listing 8.1: Python in command line

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 23 2015, 02:52:03)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

If you need to install Python, you may as well download the most recent stable version Python 3.

If you're running Windows: the most stable Windows downloads are available from the *https://www.python.org/downloads/windows/*.

If you are using a Mac: see the *https://www.python.org/downloads/windows/*.

For Debian or Ubuntu: For Debian or Ubuntu, install the python3.x and python3.x-dev packages

8.1.2 Install Packages

This section covers the basics of how to install Python packages (i.e. a bundle of software to be installed).

If you have Python 3 >=3.4 installed from python.org, you will already have pip and setuptools, but will need to upgrade to the latest version.

On Linux or OS X:

Listing 8.2: Pip install

```
pip install -U
```

On Windows:

Listing 8.3: Pip install on Windows

```
python -m pip install
```

8.1.3 Creating Virtual Environments

Python "Virtual Environments" allow Python packages to be installed in an isolated location for a particular application, rather than being installed globally.

The basic commands:

Listing 8.4: Creating Virtual Environment

```
pip install virtualenv
virtualenv <DIR>
source <DIR>/bin/activate
```

8.1.4 PostgreSQL

PostgreSQL is an object-relational database management system (ORDBMS). It supports a large part of the SQL standard and offers many modern features.

For Windows distribution: *http://www.enterprisedb.com/products-services-training/pgdownload*

After downloading the source installation just simple run the setup.exe and follow the instructions.

For OS X distribution: *http://postgresapp.com*

For MAC OS X we recommend to download Postgres.app which is a simple application that runs a menubar without the need of any installation.

8.1.5 Cubes

You may install Cubes with the minimal dependencies:

Listing 8.5: Install Cubes

```
pip install cubes
```

The cubes has optional requirements:

- **SQLAlchemy** for SQL database
- **Flask** for Slicer OLAP HTTP server

To install these requirements:

Listing 8.6: Cubes requirements

```
pip install Flask SQLAlchemy
```

There is also a customized installation with requirements:

Listing 8.7: Customized installation

```
pip install python-dateutil jsonschema expressions grako click
```

To run Cubes OLAP HTTP server:

Listing 8.8: OLAP HTTP server

```
slicer serve slicer.ini
```

And try to do some queries:

Listing 8.9: OLAP HTTP server

```
curl "http://localhost:5000/cube/spending/aggregate"
curl "http://localhost:5000/cube/spending/aggregate?drilldown=year"
curl "http://localhost:5000/cube/spending/aggregate?drilldown=country"
```

8.1.6 Flask

In order to run the web application you have install Flask which is a micro webdevelopment framework for Python.

Enter the following command to get Flask activated in your virtualenv

Listing 8.10: Install Flask

```
pip install Flask
```

Flask depends on two external libraries Werkzeug and Jinja2. The web application depends also on other external packages. For successfuly running the application install the following packages

Listing 8.11: Install external packages

```
pip install Werkzeug jinja2 Flask-WTF
```

8.1.7 Web application

After all these installations, you should be able to run the application. Now just copy the *opengovernment* folder, which is located on the DVD, into your home directory of your virtual environment. Open the *postgres* folder and run the following command to load the data into database

Listing 8.12: Load Data

```
pg_restore -Fc -C pg_dump.gz
```

Move up to folder *opengovernment*. Run the Cubes OLAP HTTP server.

Listing 8.13: OLAP HTTP server

```
slicer serve slicer.ini
```

Now run the actual Flask application *flask_app.py*

Listing 8.14: Run Flask App

```
python flask_app.py
```

CHAPTER 9

Appendix B

9.1 Contents of DVD

The content of DVD includes the following files:

- **/opengovernment/** the main directory which contains all the source code of the bachelor thesis.
- **/opengovernment/static/** folder contains all the static files for the web application such as css, javascript, images.
- **/opengovernment/data/** folder contains the source data and the cleaned data.
- **/opengovernment/postgres/** contains dump of example data and schema for Postgres database also the ETL jobs.
- **/opengovernment/models/** model metadata
- **/opengovernment/docs/html/index.html** documentation of the bachelor thesis in html format.
- **/opengovernment/slicer.ini** OLAP HTTP WSGI Server configuration file.
- **/opengovernment/tempaltes/** templates for the web application.
- **/opengovernment/flask_app** the main web application modul.

Bibliography

[Cubes] Author: Stefan Urbanek Availailable: Cubes Overview

[workspace] Author: Stefan Urbanek Availailable: Cubes Analytical Workspace

[model] Author: Stefan Urbanek Availailable: Cubes Model

[cells] Author: Stefan Urbanek Availailable: Slicing and Dicing