Open Object Business Intelligence

Release 1.0

Tiny SPRL

2009-04-09

CONTENTS

Part I

Part 1 : Introduction

CHAPTER

ONE

GOAL OF THE PROJECT

In simple words Business Intelligence or BI is the act of capturing raw data, then transforming and combining that data into information that can be proactively used to improve business. So,the goal of BI is to empower decision-makers, allowing them to make better and faster decisions. After all **Better decisions make better business!**

In general any Business Intelligence Solution must provide :

1. The first challenge business intelligence faces is gathering the necessary data about the business. The key for gathering data is automating the process. Gathering data was very time and money consuming in the past, but with today's modern computer systems, it's much easier to collect data from various sources.
2. The second business intelligence task is to analyze the collected data and to extract information from it. The extracted information is then turned into business knowledge.
3. The third and final business intelligence task is to use the newly gathered business knowledge to improve the business.

Our goal of OpenObject BI is to provide python based BI Solution that can

1. Provide all BI functanality.
2. Creating cube on the fly with minimum steps.
3. Can access any type of database to make Schema and Cube.
4. Reports can be used directly in OpenOffice.
5. Making everything as object for easy expandability.

CHAPTER

TWO

WHAT IS FOR USER?

The Open Object Business Intelligence system aims to be a full featured open source Business Intelligence system written in Python. It implements a HOLAP (Hybride OLAP = ROLAP + MOLAP) cube and a MDX query engine based on SQLAlchemy.

Comparing to most current business intelligence software in the market, our goal is to produce a BI for the mid market. It has to be:

2.1 For the end-user:

- Is easy and fast to use: a simple web-interface that does not require any dependencies and can be integrated in proprietary softwares, and an OpenOffice interface for complex dashboards creation.
- Is easy to install: auto-installation on Windows and Linux, a few dependencies
- Integrated and independent from Open ERP.

2.2 For the administrator user:

- A cube designer within Open ERP (application and web-client)
- Easy to configure: Automatic cube definition (5 clicks, using introspection on database),
- Easy to maintain: the application must be smart enough that do not require any fine tuning in the cube definition: run well on bad indexes, no need to explicitly define aggregated table, no need to define axes.
- No intervention at all from developers: everything through interfaces for end-user.

2.3 For the developer:

- Everything (dimensions,) must be object oriented with a module system to allow to add your own code to extend the software, like in Open ERP.
- It must support main database engine and aggregation of multiple database: PostgreSQL, MySQL, Oracle, MSSQL etc... to do reporting for any application.

CHAPTER

THREE

OLAP

Online Analytical Processing (OLAP) means analysing large quantities of data in real-time. Unlike Online Transaction Processing (OLTP), where typical operations read and modify individual and small numbers of records, OLAP deals with data in bulk, and operations are generally read-only. The term 'online' implies that even though huge quantities of data are involved — typically many millions of records, occupying several gigabytes — the system must respond to queries fast enough to allow an interactive exploration of the data. As we shall see, that presents considerable technical challenges.

OLAP performs multidimensional analysis of business data and provides the capability for complex calculations, trend analysis, and sophisticated data modeling. Whereas a relational database stores all data in the form of rows and columns, a multidimensional dataset consists of axes and cells. Consider the dataset

Year	**2000**		**2001**		**Growth**	
Product	Dollar sales	Unit sales	Dollar sales	Unit sales	Dollar sales	Unit sales
Total	$7,073.00	2693	$7,636.00	3008	8.00%	12.00%
Books	$2,753.00	824	$3,331.00	966	21.00%	17.00%
Fiction	$1,341.00	424	$1,202.00	380	-10.00%	-10.00%
Non-fiction	$1,412.00	400	$2,129.00	586	51.00%	47.00%
Magazines	$2,753.00	824	$2,426.00	766	-12.00%	-7.00%
Greetings cards	$1,567.00	1045	$1,879.00	1276	20.00%	22.00%

The rows axis consists of the members 'All products', 'Books', 'Fiction', and so forth, and the columns axis consists of the cartesian product of the years '2000' and '2001', and the calculation 'Growth', and the measures 'Unit sales' and 'Dollar sales'. Each cell represents the sales of a product category in a particular year; for example, the dollar sales of Magazines in 2001 were $2,426.

This is a richer view of the data than would be presented by a relational database. The members of a multidimensional dataset are not always values from a relational column. 'Total', 'Books' and 'Fiction' are members at successive levels in a hierarchy, each of which is rolled up to the next. And even though it is alongside the years '2000' and '2001', 'Growth' is a calculated member, which introduces a formula for computing cells from other cells.

The dimensions used here — products, time, and measures — are just three of many dimensions by which the dataset can be categorized and filtered. The collection of dimensions, hierarchies and measures is called a cube.

The above simple example or outlook shows how one can get the smallest details from the data stored for years in the database in form of relation. It helps in managing resources, forming policies , budgeting , Business Process Management,designing strategic marketing policies, forecasting and many more. The options are endless

The multidimensional is above all is a way of presenting data. Although some multidimensional databases store the data in multidimensional format, I shall argue that it is simpler to store the data in relational format and manipulate it using OLAP.

3.1 Who uses OLAP and Why?

OLAP applications span a variety of organizational functions. Finance departments use OLAP for applications such as budgeting, activity-based costing (allocations), financial performance analysis, and financial modeling. Sales analysis and forecasting are two of the OLAP applications found in sales departments. Among other applications, marketing departments use OLAP for market research analysis, sales forecasting, promotions analysis, customer analysis, and market/customer segmentation. Typical manufacturing OLAP applications include production planning and defect analysis.

Important to all of the above applications is the ability to provide managers with the information they need to make effective decisions about an organization's strategic directions. The key indicator of a successful OLAP application is its ability to provide information as needed, i.e., its ability to provide "just-in-time" information for effective decision-making. This requires more than a base level of detailed data.

Just-in-time information is computed data that usually reflects complex relationships and is often calculated on the fly. Analyzing and modeling complex relationships are practical only if response times are consistently short. In addition, because the nature of data relationships may not be known in advance, the data model must be flexible. A truly flexible data model ensures that OLAP systems can respond to changing business requirements as needed for effective decision making. Although OLAP applications are found in widely divergent functional areas, they all require the following key features:

1. Multidimensional views of data
2. Calculation-intensive capabilities
3. Time intelligence

3.1.1 Multidimensional views of data

Multidimensional views are inherently representative of an actual business model. Rarely is a business model limited to fewer than three dimensions. Managers typically look at financial data by scenario (for example, actual vs. budget), organization, line items and at sales data by product, geography, channel, and time.

A multidimensional view of data provides more than the ability to "slice and dice"; it provides the foundation for analytical processing through flexible access to information. Database design should not prejudice which operations can be performed on a dimension or how rapidly those operations are performed. Managers must be able to analyze data across any dimension, at any level of aggregation, with equal functionality and ease. OLAP software should support these views of data in a natural and responsive fashion,insulating users of the information from complex query syntax. After all, managers should not have to understand complex table layouts, elaborate table joins, and summary tables.

Whether a request is for the weekly sales of a product across all geographical areas or the year-to-date sales in a city across all products, an OLAP system must have consistent response times. Basic aggregation is performed on some of the dimensions (product, customer, and channel). More complex calculations are performed on other dimensions. The measure dimension computes ratios and averages. Variances are computed along the scenario dimension. A complex model based on historical performance is used to compute the forecast scenario. Consistently quick response times to these kinds of queries are key to establishing a server's ability to provide multidimensional views of information.

3.1.2 Calculation-intensive capabilities

The real test of an OLAP database is its ability to perform complex calculations. OLAP databases must be able to do more than simple aggregation. While aggregation along a hierarchy is important, there is more to analysis than simple data roll-ups. Examples of more complex calculations include share calculations (percentage of total) and allocations(which use hierarchies from a top-down perspective).

Key performance indicators often require involved algebraic equations. Sales forecasting uses trend algorithms such as moving averages and percentage growth. Analyzing the sales and promotions of a given company and its competitors requires modeling complex relationships among the players. The real world is complicated – the ability to model complex relationships is key in analytical processing applications.

Whereas transaction processing systems are judged on their ability to collect and manage data, analytical processing systems are judged on their ability to create information from data.

3.1.3 Time intelligence

Time is an integral component of almost any analytical application. Time is a unique dimension because it is sequential in character (January always comes before February). True OLAP systems understand the sequential nature of time. Business performance is almost always judged over time, for example, this month vs. last month, this month vs. the same month last year. The time hierarchy is not always used in the same manner as other hierarchies. For example, a manager might ask to see the sales for May or the sales for the first five months of 1995. The same manager might also ask to see the sales for blue shirts but would never ask to see the sales for the first five shirts. Concepts such as year-to-date and period over period comparisons must be easily defined in an OLAP system.

In addition, OLAP systems must understand the concept of balances over time. For example, if a company sold 10 shirts in January, five shirts in February, and 10 shirts in March, then the total balance sold for the quarter would be 25 shirts. If, on the other hand, a company had a head count of 10 employees in January, only five employees in February, and 10 employees again in March, what was the company's employee head count for the quarter? Most companies would use an average balance. In the case of cash, most companies use an ending balance.

CHAPTER

FOUR

TERMINOLOGIES

This page define all terminolgies. Objects in the OLAP cube use this convention.

Schema

A schema is a collection of N dimensions. It's the meta description of cubes..

Hierarchy

A schema is divided in hierarchy, which are divided in dimensions. The main use of hierarchy is to check that different axis can not use dimensions of the same hierarchy.

Dimension

A dimension is an attribute, or set of attributes, by which you can divide measures into sub-categories. It's a tree structure that define the axis of the cube. They can be explicitly defined: partner_id.country_id.state_ids or recursive 'parent_id'. A dimension is divided in levels.

Level

One level of sub-categories defined by dimensions.

Measure

Meta data of the quantity your are measuring. (value) A measure may be complex, ex: the tuple (quantity,uom) Attributes which are also objects:

> Agregator: an SQL function that define how we aggregate measures "sum", "count", "min", "max", "avg", and "distinct-count" FormatString DataType (the measure/value datatype)

Cube

A cube is a collection of N axis. A cube is an instance of a schema. A cube is mapped to a 'SQL' query through the use of his axis. (or several)

Member

A member is a point within a dimension determined by a particular set of attribute values. (instances) A member is able to compute a part of the SQL query.

Axis

An axis is composed by one or a set of members. In others terms, the axis is defined by the part of the query preceeding the "on rows", "on colmuns", "on pages"... The MDX result is also a cube coumposed of axis.

Value

A value is an instance of a measure. (one particular case of the cube).

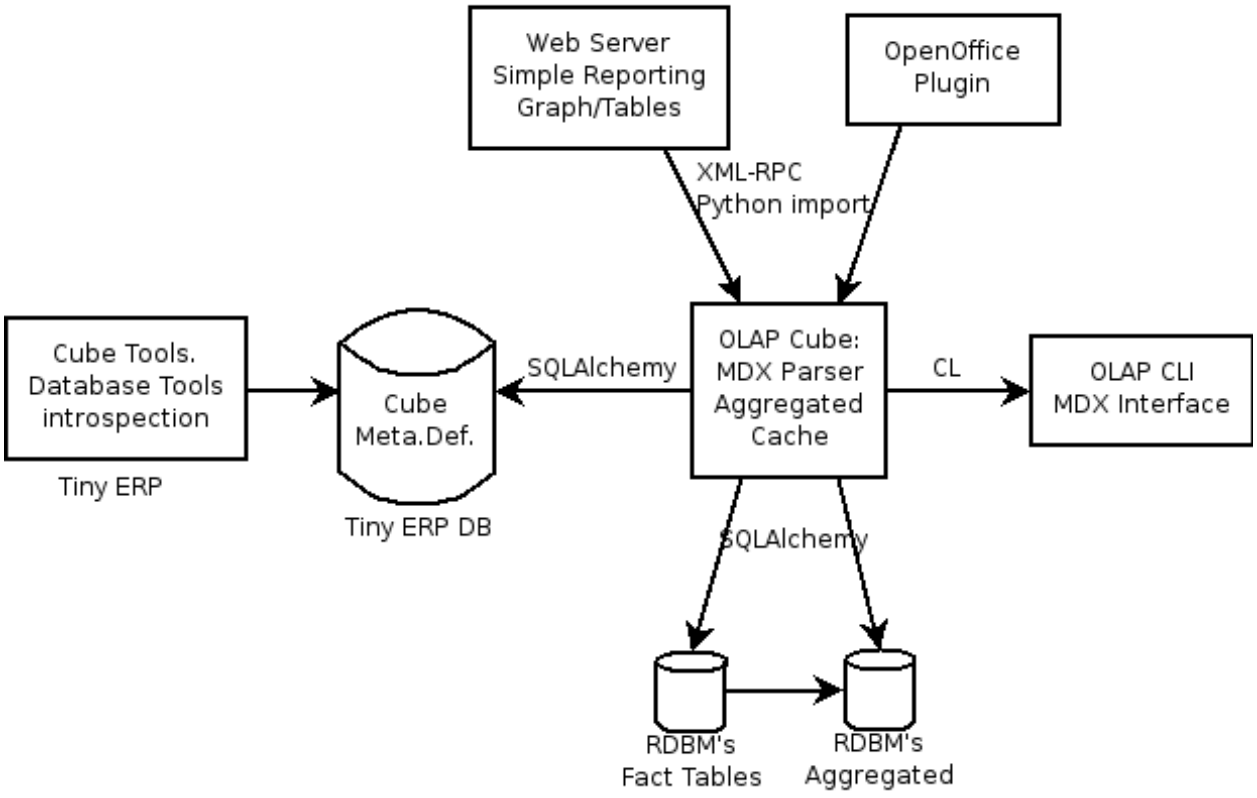
Part II

Part 2 : Architecture

CHAPTER

FIVE

SCHEMA



CHAPTER

SIX

COMPONENTS

6.1 The Cube

The cube is based of the following component:

- **A MDX parser that will transform an MDX expression to RDBMs queries:**
  - **Computed using a mix of:**
    * Using star flow snake like in mondrian (based on joins)
    * Using space hyerarchy cutting like in cubulus
- **A memory cache system**
  - On space hyerarchies (dimensions with space cutting)
- **An aggregation system**
  - Ability to create aggregated table to speed up all queries (automatic or user-defined)
  - Queries will be computed on fact tables or aggregated tables
- A MDX Output (or several) to output the result

The cube will use:

- SQLAlchemy for all database communications
- XML-RPC for his external interfaces
- PyParser for MDX parsing

6.2 The CLI interface

Allows user to test MDX queries in this CLI command line interface. Simple script in python that will send XML-RPC queries and print the result.

6.3 The Cube Definition

The meta data of the cube definition will be stored in the Open ERP database. The user interface to edit cubes is in Open ERP. We will use the same concept of the one defined in the ... XML standard. So that we will be able, in a futur phase, to import such files.

This must not depend on any module of Open g ERP so that if you want to use the BI library independently, you may not use Open ERP if cubes are defined. If cubes are not defined, you just install the minimal version of Open ERP

that includes: the olap module, user management, workflow managements, access rights management, ... (the base module)

The goal is that the user never have to create the cubes himself. We will create a wizard that will compute cubes based on introspection on the RDBM's. The steps of this wizard:

- Selection of the database (type of db, then selection box like in the login of Open ERP)
- Selection of the factable (selection box)
- Selection of the measures and their attributes (selection box, aggregation func)
- Selection of the dimensions (click on a tree structure)

Then it's done, the cube is computed. The aggrgated table may be also auto-matically computed by Open ERP.

The goal is to create new cube on the fly from the Open ERP client on every object, on user demand. This will also server the online demo server.

The cube creation can be stored in the server of kept in memory for one time usage.

6.4 The Web Client

The web client is a web-server that display cubes and provide tools to browse them, it must provide at least these operations:

- switch view
- different type of charts
- drill up/down
- slice
- dice

6.5 The OpenOffice plugin

Similar to Palo but all operation of contruction and manipulation of cubes remains in Open ERP to limit development on OOo. The development on OOo just contains functions to:

- Insert new data (based on selection of dimensions and filters)
- Drill up/down functions
- Slice function

6.6 The Open ERP interface

From Open ERP, you should be able to right click/drag and drop any field to trigger the cube definition wizard to create your own cube on demand. For this, we will use the web client of the bi system.

We will intergate this on the gtk and web client of Open erp. For the GTK one, it will open the browser to browse the cube.

CHAPTER

SEVEN

EXTRA LIBRARIES

Libraries we will use:

- Turbogears for the web client to browse cube
- Mathplotlib for rendering graphs
- PyParsing to parse MDX Expressions
- SQLAlchemy to construct SQL queries and RDBMS connections
- XMLRPC lib for communication with the cube server
- PÿUNO for the OOo integration

We will use an object relationnal mapping system on all objects: dimensions, ...

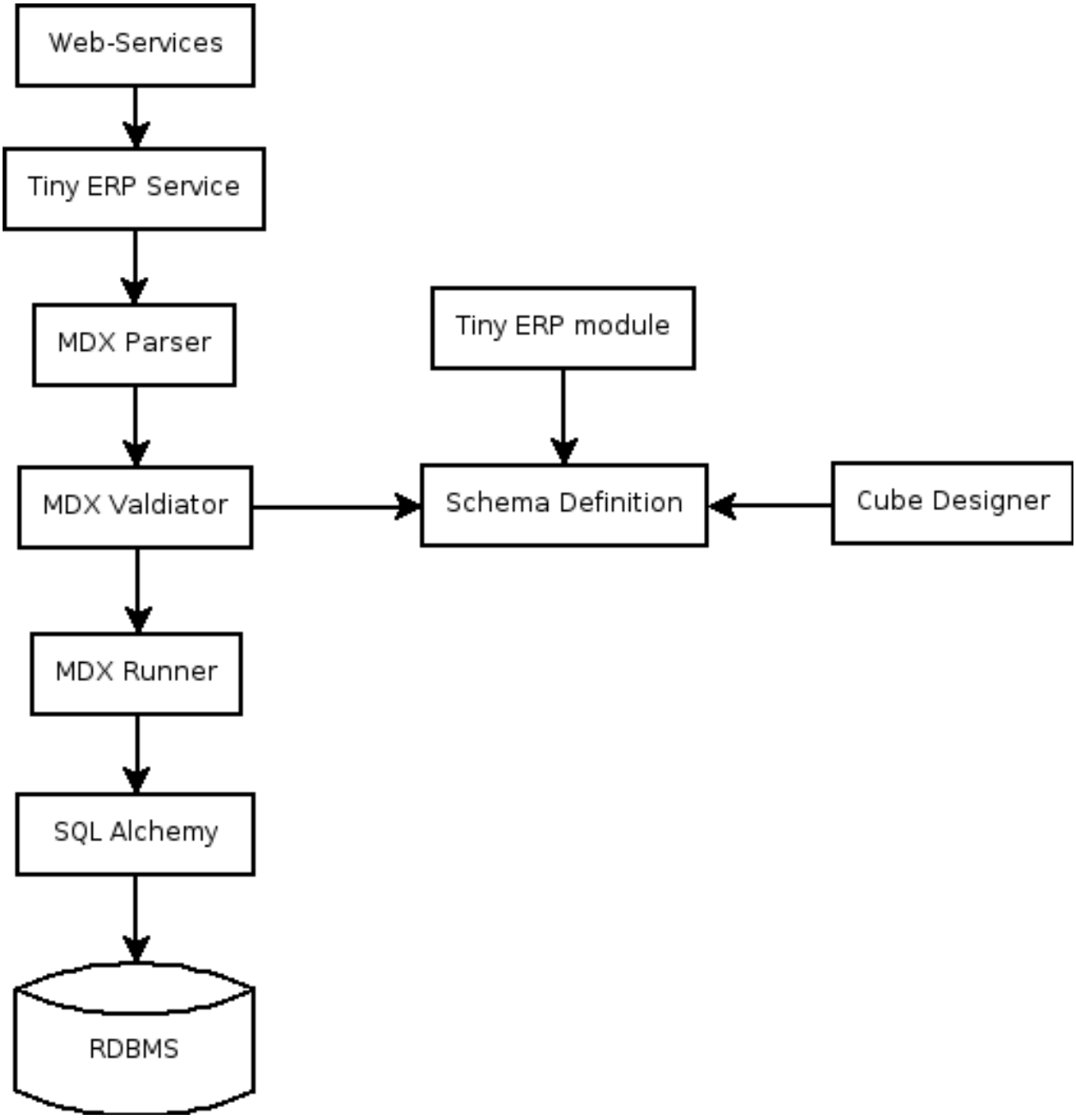
CHAPTER

EIGHT

INTRODUCTION TO THE OPENOBJECT MODULE

The OLAP module is used in validating , running and formatting the output of MDXExamples

The general flow is of OLAP module is shown in following diagram:



8.1 Explanation of the components

8.1.1 Web-Services

This is the layer provided by the base of Open ERP, protocols: NET-RPC (fast binary), XML-RPC, over HTTP or HTTPS

8.1.2 Services

Layer provided by Open ERP that provides: authentification (normal/ldap), users management, access rights, workflows, module management, ...

8.1.3 MDX Parser

It parses the MDX query and convert it in the form of python objects. It uses pyparsing module of python to do this . It split the query in form of objects of axis, level, sub level, slicer (if any) and measures.

8.1.4 MDX Validator

It parse all the objects created and map it to the browse object ofOpen ERP resource. For example, the axis object will receive a link to the Open ERP browse record on the related olap.axis object.

8.1.5 MDX Runner

It will run the query on the basis of objects using SQLAlchemy and return different subsets. On the basis of it the cube is virtually made in the form of matrix. And it fills the cube by values using axis mapping

8.1.6 RDBMS connectors

The layer provided by SQL Alchemy, it supports: mysql, postgresql, oracle, ...

The schema definition is in the Open ERP database.

Part III

Part 3 : MDXAlchemy

CHAPTER

NINE

MDXOVERVIEW

MDX stands for Multidimensional Expressions. You use it to query OLAP databases. In a nutshell, MDX is to OLAP databases as SQL queries are to relational databases.

OLAP databases primarily consist of OLAP cubes, which store fact tables, measures (such as sales, purchase, etc.) and dimensions/hierarchies. An OLAP database is often an aggregation of a relational database; as a result, you can write MDX queries to retrieve key calculations that measure company performance, often with less code than standard SQL.

Because of the nature of OLAP databases, we need to write MDX code to retrieve data in far fewer lines of code than would be required using SQL. This is a segue into the role that OLAP databases and MDX play in the world of business intelligence.

MDXAlchemy is developed taking care of all the aspects of becoming a complete OLAP Engine, to execute MDX query and fetch data efficiently. MDXAlchemy is a complete MDX engine that provides your database with full MDX capabilities.

MDXAlchemy use the services of SQLAlchemy to provide few of important feature that makes MDXAlchemy a full capable MDX Engine. The major is removing the clause of database dependency.

The dimensional meta data facility addresses the issue that although this application store dimensional data in relational tables (usually in the form of fact and dimension tables), the user doesnot have to worry for database not aware of the dimensionality, or OLAP semantics, of this data. It provides a comprehensive meta data facility to define these semantics and an XML capability to enable meta data interchange with other external OLAP products.

9.1 Independent yet Integrated to OpenERP

MDXAlchemy engine is totally independent of OpenERP. It does not rely on OpenERP modules for its functionality. Yet being so diversified from OpenERP, it is fully integrated to OpenERP.

MDXAlchemy can be installed/intergrated as an internal module of OpenERP and it makes itself ready to assist OpenERP.

It is made of two major components, a cube engine and SQL Alchemy. It uses SQL Alchemy for connecting to the database and cube engine processes the data to form the cube for the user.

9.2 Supported Databases

OpenObject BI takes care of independent database functionality.

It supports connectivity with most of the leading Databases programs.

Just a compatible Dialect and valid connection parameters is all that is needed to use a database.

The Dialect is used to describe how to talk to a specific kind of database. Dialects are included with SQLAlchemy for SQLite, Postgres, MySQL, MS-SQL, Firebird, Informix, and Oracle; these can be described as Python modules present in the sqlalchemy.databases package. Each dialect requires the appropriate DBAPI drivers to be installed separately.

Downloads for each DBAPI to connect to supported Databases are as follows:

- Postgres: psycopg2
- SQLite : pysqlite
- MySQL : MySQLDB
- Oracle : cx_Oracle
- MS-SQL : pyodbc (recommended) adodbapi pymssql
- Firebird: kinterbasdb
- Informix: informixdb

CHAPTER

TEN

A LOOK AT FEW EXAMPLES TO WRITE MDX QUERIES

Writing a simple MDX on a SALES Schema:

```
select
  {[partner_country].[all]} on rows,
  {[measures].[Items Sold]} on columns
from sale_order_line
```

Gives results as,

/	Items Sold
All partner_country	[43.0]

Expanding the partner country to its children, we change the MDX as.:

```
select
   {[partner_country].[all],[partner_country].children} on rows,
   {[measures].[Items Sold]} on columns
from sale_order_line
```

Gives result as,

/	Items Sold
All partner_country	[43.0]
Belgium	[30.0]
China	[4.0]
France	[9.0]

One more example bit complex with slicer:

```
select
  {[date_order].children,[date_order].[2008].children,[date_order].[2008].[Q2].children},
  {[measures].[Items Sold]}
from sale_order_line
where ([date_order].[2008])
```

Gives results as:

/	Items Sold
2008.0	[43.0]
Q1	[43.0]

Part IV

Part 4 : Installing

CHAPTER

ELEVEN

INSTALLATION OF BI

To make BI working on any system. We need a OpenERP Server and preferably OpenERP GTK client if needed running on the system.

1. *Installing Open ERP Server and OLAP module* (in *Open Object Business Intelligence*)
2. *Installing Web Interface* (in *Open Object Business Intelligence*)

Part V

Part 5 : Configuration

CHAPTER

TWELVE

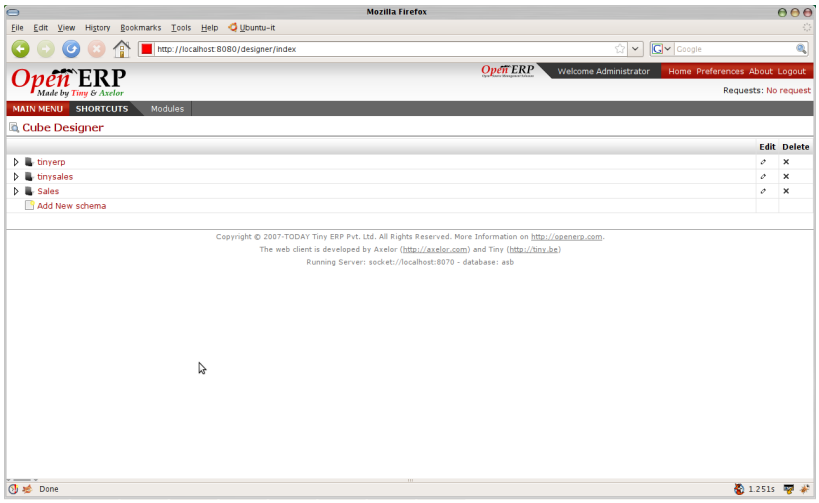
CONFIGURATION INTERFACE

The main goal of any user connecting to OpenObject BI is to fetch the data from database using the powerful MDX queries.

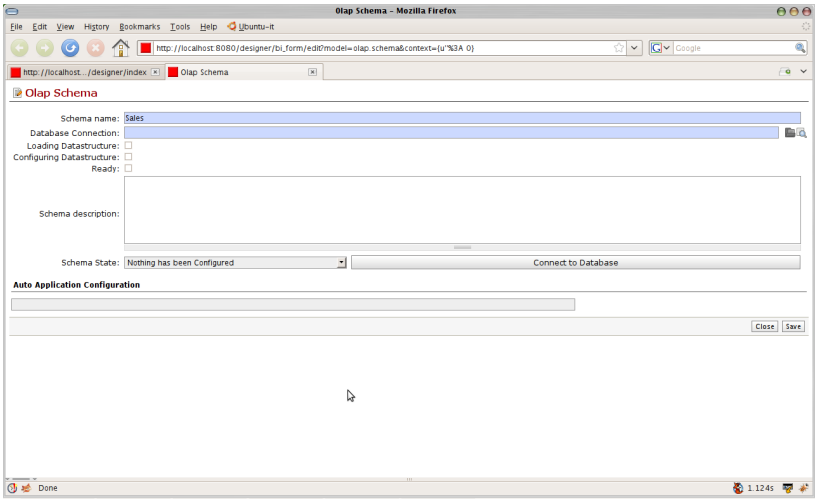
To run any MDX Query there is a need to make a cube and the user can define / configure its own custom cube using two interface :

12.1 Introduction

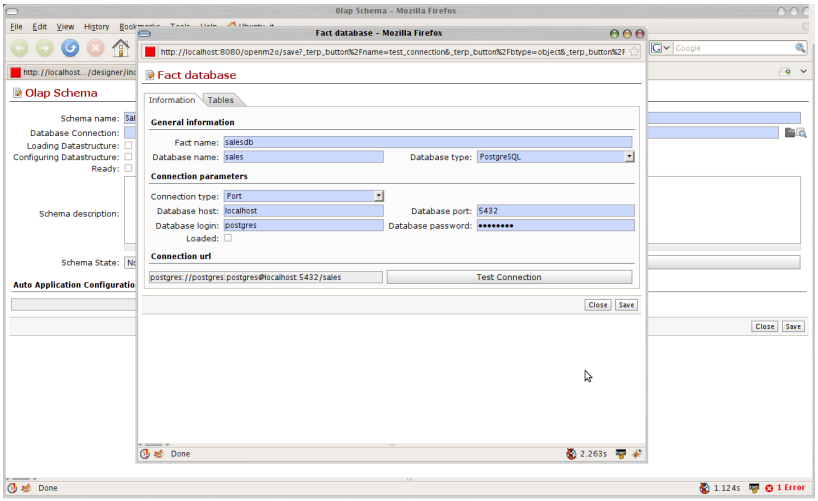
Designer by default displays all schema in the tree form and provide options for adding the new.:



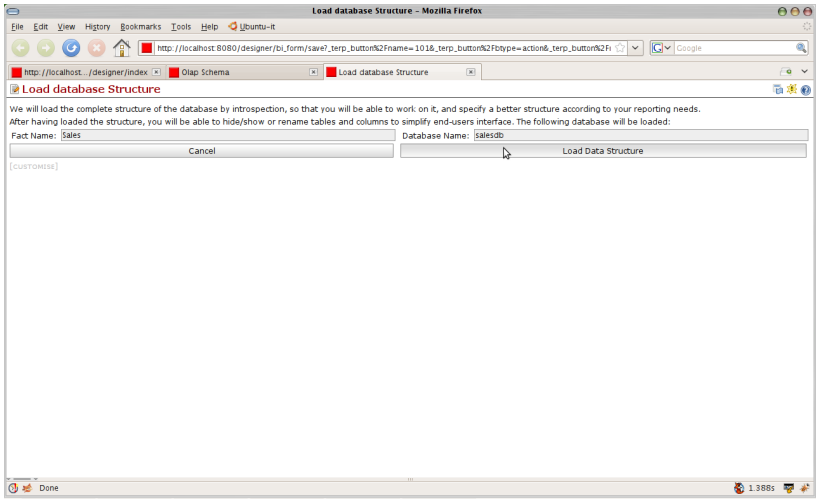
Creating the Schema : Schema defines the database from where the data is to be fetched. It gives a meaning ful name to the database connection.:



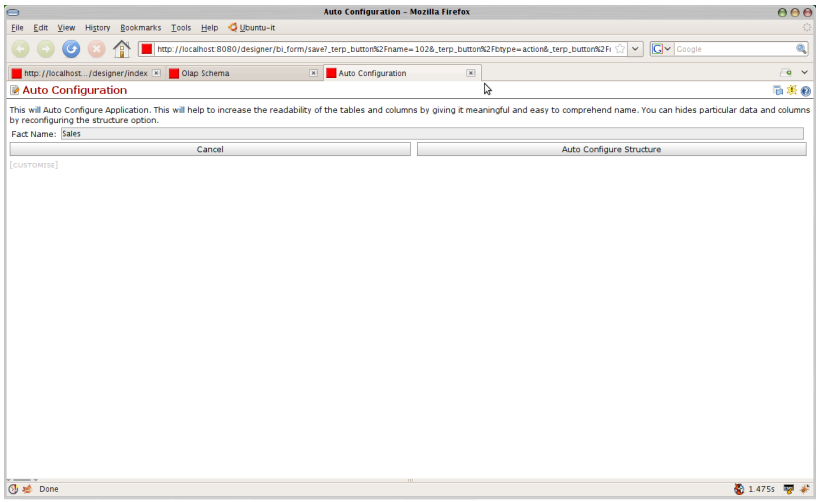
Database Connection specified the paramaters for connecting to the database. It generally includes type of the database (postgres,oracle,mysql), username, password , database to use.:



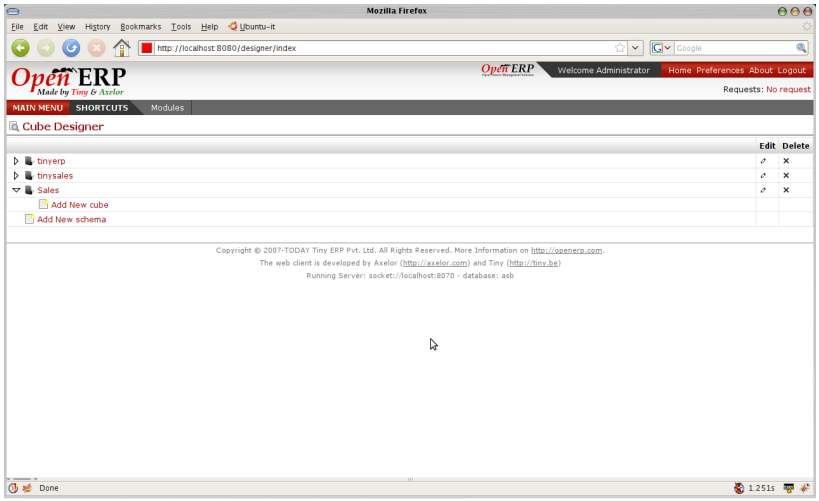
Once we configure the database connection the next step is to load the database using introspection. This will load the structure of the database. by structure we mean tables, columns and the relations. This will help in defining cube easily. As the structure is loaded their will be no query to the database again and again:



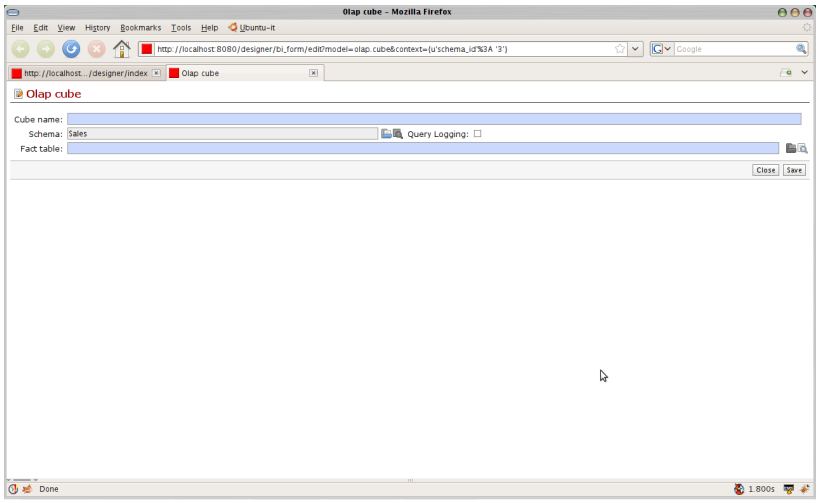
The next step is to configure the database loaded. This is useful to hide unnecessary table and columns. If database is of openerp it can be auto configured:



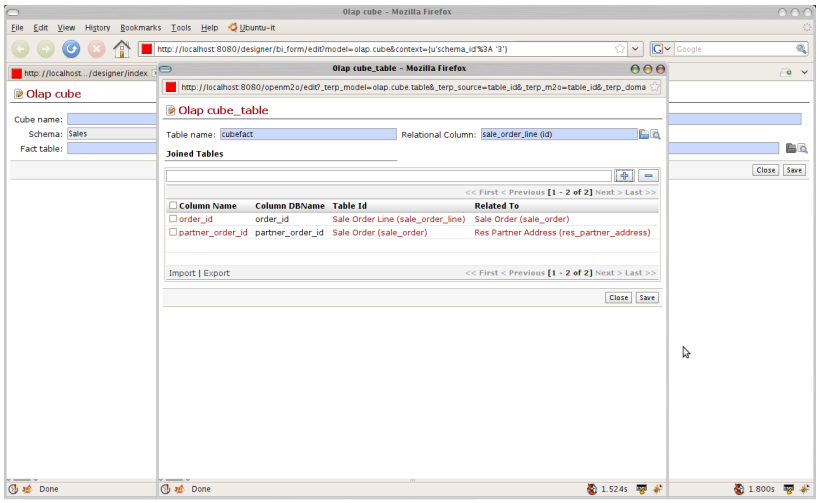
Once the cube schema is created we can go for creating the cube:



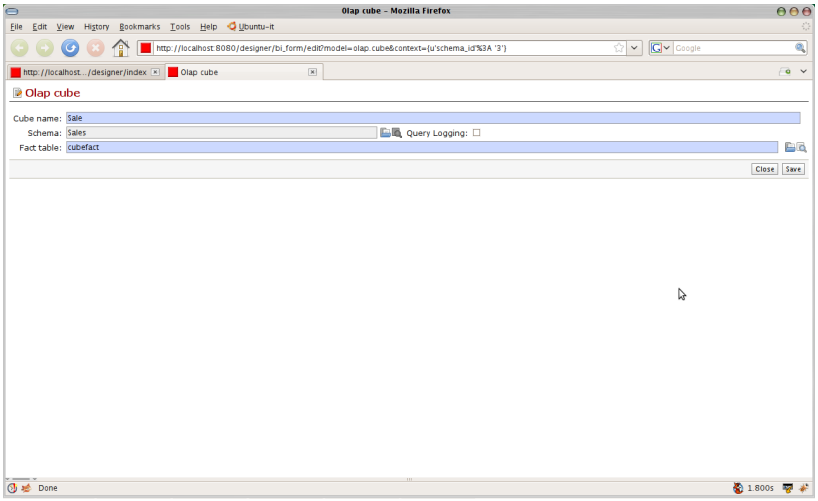
Cube is the structure that is based on the schema (database), It will configure the way to retrieve the data:



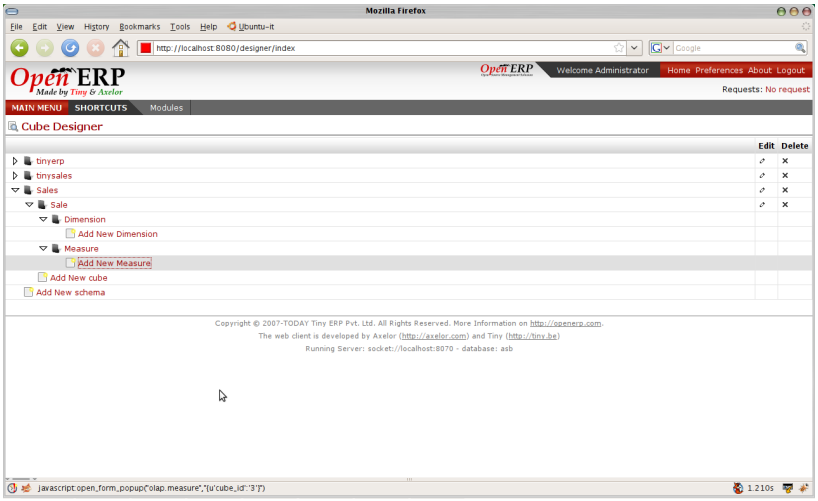
Cube requires the fact table to be define. Fact table are the key tables in which measures are stored and we can branch to other tables for other parameters. For example for sales we can define sale_order as our fact table as it will gives the details of the sales. Fact table can be join of tables. The fact table is given meaningful name:



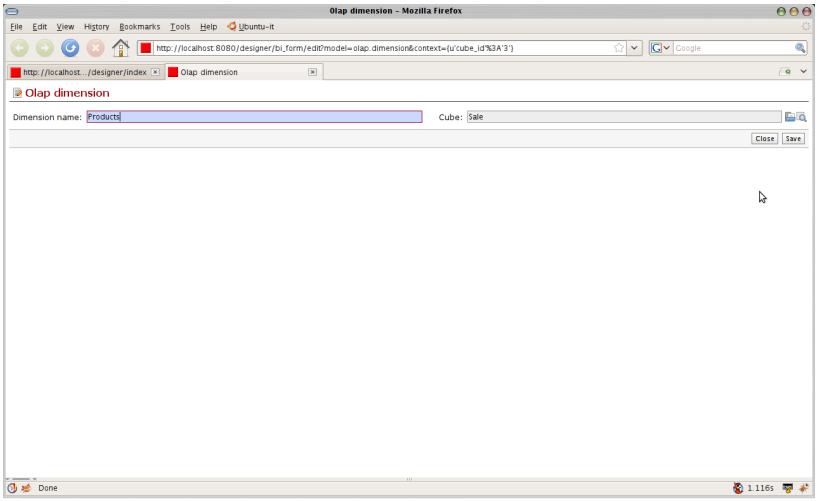
And the cube screen will be



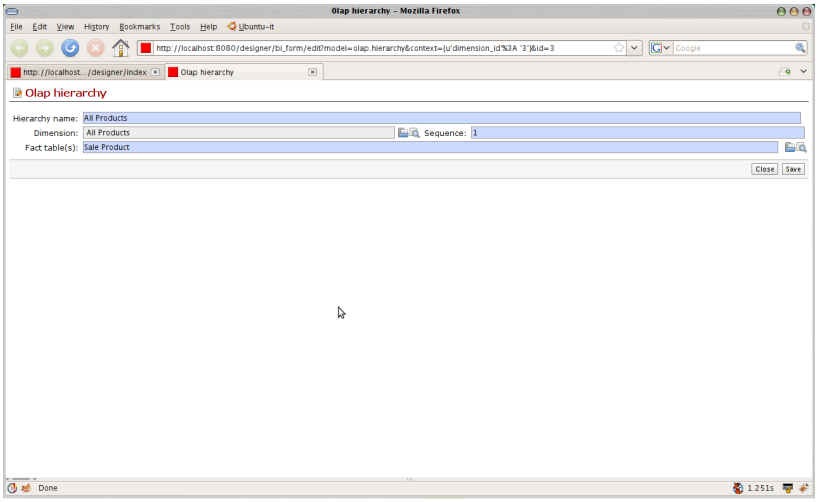
After cube we can decide upon the dimensions to be used for the cube. For example we want to look on products sold , Dates, City etc.. to analyis the sales accordingly. We decide what are the measures to be used. For example items sold. So we can decide the dimension and measures:



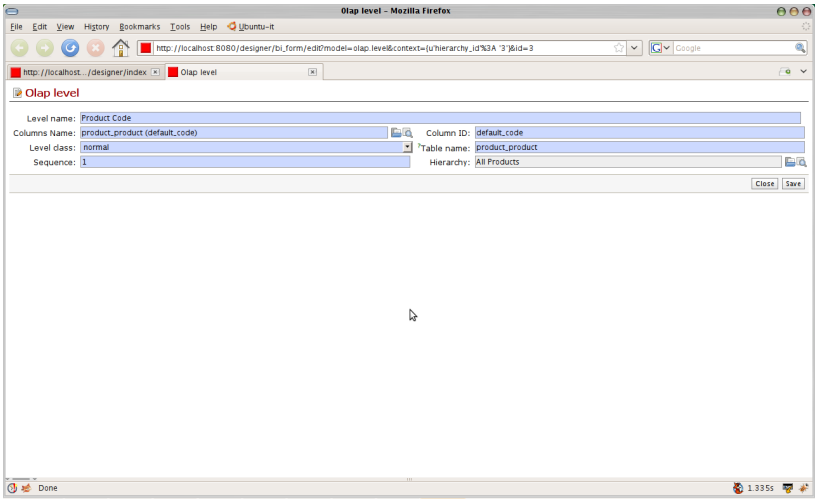
Adding the dimension Products. So we will be able to see product wise item sold:



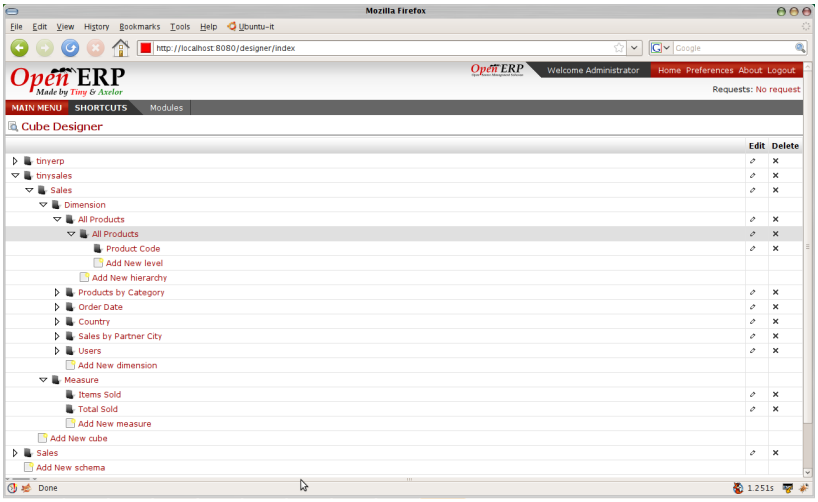
After dimension we explain how to get the prodcuts details in the hierarchy. It requires to configure the fact table:



After adding the hierarchy we decide from which field the product name will come:



The fully configured cube tree will look like:



12.2 Connecting to an Existing Database

One can very easily connect to the existing database. The details requiered are

1. Fact Name : Logical Name of the database
2. Database Name: Pyhsical Database name to be used
3. Database type : Type of the database it can be PostgreSQL, MySQL, Oracle etc.
4. Connection type : Port or Socket
5. Database Host : Server name like localhost
6. Database Port : Port to be used for making connection to the database
7. Database Login: Login name for accessing a database
8. Database Password:Password for the user in login

Giving this detail will generate a string like ‘’postgres://postgres:postgres@localhost:5432/terp’‘

Strings so generated is a connection string for making connection to the database.

12.3 Writing a Schema

`What is Schema ?`

Schema in general means shape or more generally plan . In the context of OpenObject BI it defines the logical model, consisting of cubes, hierarchies, and members, and a mapping of this model onto a physical model.

The logical model consists of the constructs used to write queries in MDX language: cubes, dimensions, hierarchies, levels, and members.

The physical model is the source of the data which is presented through the logical model. It is typically a star schema, which is a set of tables in a relational database; later, we shall see examples of other kinds of mappings.

12.3.1 Making Schema

In OpenObject BI schemas are represented in a XML file. It can be designed in the way Open ERP does. The details of XML file can be seen at *Creating XML*

CHAPTER

THIRTEEN

DEFINING SCHEMA IN XML

Why XML for Schema?

- XML is a meta language used to describe the structure and content of documents.
- XML, although originally a document markup language, is increasingly used for data exchange on the Web.
- The application of XML as a standard exchange format for data available on the Web makes it attractive to use in conjunction with OLAP tools.
- An XML document has both structure and content, and XML provides a means for separating one from the other in the electronic document.
- The structure of a document is given by use of matching tag pairs (termed an element) and the information between matching tags is referred to as the content of the element. Furthermore, an element is permitted to have additional attributes, where values are assigned to the attributes in the start tag of the element.
- XML documents can also contain a description of their logical structure, which is called a document type definition (DTD). A DTD is a context free grammar defining, in terms of element content specifications, all allowable elements,their attributes, and the elements nesting structure. Given a DTD it can be verified that an XML document conforms to the DTD, and if so, the XML document is said to be valid.

So all these features make XML a attracted language to make use of it in making a schema.

CHAPTER

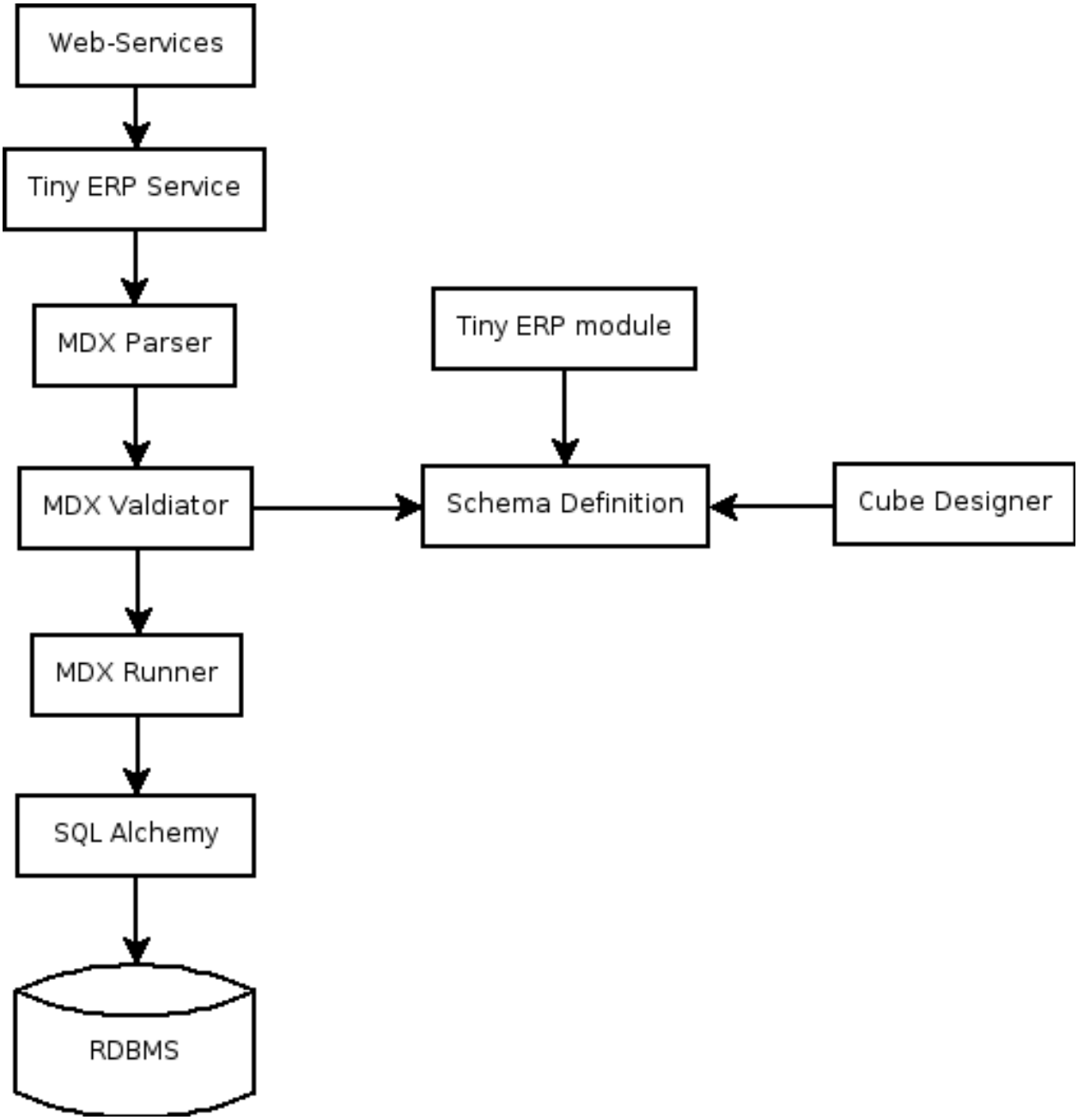
FOURTEEN

XML IN OPEN OBJECT

14.1 Introduction to the OpenObject Module

The *OLAP* module is used in validating , running and formatting the output of *MDXExamples/MDX Queries*

The general flow is of *OLAP* module is shown in following diagram:



14.2 Explanation of the components

Web-Services

This is the layer provided by the base of Open ERP, protocols: NET-RPC (fast binary), XML-RPC, over HTTP or HTTPS

Services

Layer provided by Open ERP that provides: authentification (normal/ldap), users management, access rights, workflows, module management, ...

MDX Parser

It parses the MDX query and convert it in the form of python objects. It uses pyparsing module of python to do this . It split the query in form of objects of axis, level, sub level, slicer (if any) and measures.

MDX Validator

It parse all the objects created and map it to the browse object ofOpen ERP resource. For example, the axis object will receive a link to the Open ERP browse record on the related olap.axis object.

MDX Runner

It will run the query on the basis of objects using SQLAlchemy and return different subsets. On the basis of it the cube is virtually made in the form of matrix. And it fills the cube by values using axis mapping

RDBMS connectors

The layer provided by SQL Alchemy, it supports: mysql, postgresql, oracle, ...

The schema definition is in the Open ERP database.

CHAPTER

FIFTEEN

CREATING CUBE DEFINITION USING XML FILE

Things to know

Before going through XML details its good to have an idea of all *Terminolgies* of OLAP.

Lets understand XML file in details

Step: 1. The first step is to specify the database it will use with parameter like name, database name , database login and database password

```
<record model="olap.fact.database" id="fact_databases_BI">
     <field name="name">Tiny ERP databases</field>
     <field name="db_name">Sales</field>
     <field name="db_login">postgres</field>
     <field name="db_password">postgres</field>
</record>
```

- This will create the connection parameters needed for connecting to the database.

Step: 2. Defining Schema

```
<record model="olap.schema" id="schema_main_sales">
     <field name="name">tinysales</field>
     <field name="state">none</field>
     <field name="database_id" ref="fact_databases_BI"/>
</record>
```

- This will create schema name tinysales for fact_database_BI made in step 1

Step: 3. Defining fact table to be used (In this case, sale_order_line)

```
<record model="olap.cube.table" id="table_sales_order_line">
     <field name="name">sale_order_line</field>
              <field name='schema_id' ref='schema_main_sales'/>
</record>
```

Step: 4. Making Cube on fact_table

```
<record model="olap.cube" id="cube_sales_order_line">
     <field name="name">sale_order_line</field>
     <field name="table_id" ref="table_sales_order_line"/>
     <field name="schema_id" ref="schema_main_sales"/>
</record>
```

- This will create cube name sale_order_line

Step: 5. Creating Dimension product

- This will used to fetch and make MDX Query on all the product

```
 <record model="olap.dimension" id="dimension_product_template">
      <field name="name">Products</field>
      <field name="cube_id" ref="cube_sales_order_line"/>
</record>

 <record model="olap.cube.table" id="table_product_template">
      <field name="name">product_product</field>
 </record>
```

Step: 5a. Creating Hierarchy for the Dimension Product

```
 <record model="olap.hierarchy" id="hierarchy_product_template">
      <field name="name">All Products</field>
      <field name="dimension_id" ref="dimension_product_template"/>
      <field name="primary_key_table">product_product</field>
      <field name="table_id" ref="table_product_template"/>
</record>
```

Step: 5b Creating Level for the Dimension Product

First,We Create Column.

```
<record model="olap.database.columns" id="columns_product_product_default_code">
        <field name="name">default_code</field>
        <field name="column_db_name">default_code</field>
        <field name="type">varchar</field>
        <field name="table_id" ref="table_product_template>
        <field name="active">True</field>
</record>
```

Now,Level.

```
 <record model="olap.level" id="level_product_template">
     <field name="name">default_code</field>
     <field name="column_name" ref="columns_product_product_default_code"></field>
     <field name="hierarchy_id" ref="hierarchy_product_template"/>
     <field name="table_name">res_partner</field>
     <field name="column_id_name">name</field>
</record>
```

Step: 6 Creating Dimension date_order up to the quarters

```
<record model="olap.dimension" id="dimension_sales_order">
    <field name="name">Order Date</field>
    <field name="cube_id" ref="cube_sales_order_line"/>
 </record>

 <record model="olap.cube.table" id="table_sales_order">
    <field name="name">sale_order</field>
 </record>

 <record model="olap.hierarchy" id="hierarchy_sales_order">
    <field name="name">Order Date</field>
    <field name="dimension_id" ref="dimension_sales_order"/>
    <field name="primary_key_table">sale_order</field>
    <field name="table_id" ref="table_sales_order"/>
 </record>

 <record model="olap.database.columns" id="columns_sale_order_date_order">
    <field name="name">date_order</field>
    <field name="column_db_name">date_order</field>
    <field name="type">date</field>
    <field name="table_id" ref="table_sale_order"/>
    <field name="active">True</field>
  </record>
```

Making levels in Order Date so to get details as per year,quarters and months.

```
<record model="olap.level" id="level_sales_order">
   <field name="name">date_order</field>
   <field name="column_name" ref="columns_sale_order_date_order"></field>
   <field name="column_id_name">date_order</field>
   <field name="type">date_year</field>
   <field name="sequence">1</field>
   <field name="table_name">sale_order</field>
   <field name="hierarchy_id" ref="hierarchy_sales_order"/>
</record>

<record model="olap.level" id="level_sales_order_q">
   <field name="name">date_order</field>
   <field name="column_name" ref="columns_sale_order_date_order"></field>
   <field name="column_id_name">date_order</field>
   <field name="type">date_quarter</field>
   <field name="sequence">2</field>
   <field name="table_name">sale_order</field>
   <field name="hierarchy_id" ref="hierarchy_sales_order"/>
</record>
<record model="olap.level" id="level_sales_order_m">
   <field name="name">date_order</field>
   <field name="column_name" ref="columns_sale_order_date_order"></field>
   <field name="column_id_name">date_order</field>
   <field name="type">date_month</field>
   <field name="sequence">3</field>
   <field name="table_name">sale_order</field>
   <field name="hierarchy_id" ref="hierarchy_sales_order"/>
</record>
```

Step: 7 Creating Dimension res_country

```
 <record model="olap.cube.table" id="table_sale_order">
   <field name="name">sale_order</field>
           <field name='schema_id' ref='schema_main_sales'/>
</record>

<record model="olap.cube.table" id="table_partner_address_0">
   <field name="name">res_partner_address</field>
   <field name='schema_id' ref='schema_main_sales'/>
</record>
<record model="olap.cube.table" id="table_partner_address_1">
   <field name="name">res_country</field>
   <field name='schema_id' ref='schema_main_sales'/>
</record>

<record model="olap.cube.table" id="table_partner_address">
   <field name="name">res_partner_address</field>
   <field name='schema_id' ref='schema_main_sales'/>
</record>

<record model="olap.cube.table" id="table_partner_country">
   <field name="name">sale_order_country</field>
   <field name='schema_id' ref='schema_main_sales'/>
</record>

<record model="olap.dimension" id="dimension_partner_country">
   <field name="name">Sales From Partners</field>
   <field name="cube_id" ref="cube_sales_order_line"/>
</record>

<record model="olap.hierarchy" id="hierarchy_partner_country">
   <field name="name">partner_country</field>
   <field name="dimension_id" ref="dimension_partner_country"/>
   <field name="primary_key_table">sale_order</field>
   <field name="table_id" ref="table_partner_country"/>
</record>

<record model="olap.level" id="level_partner_country">
   <field name="name">country_id</field>
   <field name="column_name" ref="columns_sale_order_date_order"></field>
   <field name="column_id_name">name</field>
   <field name="table_name">res_country</field>
   <field name="hierarchy_id" ref="hierarchy_partner_country"/>
</record>
```

Step: 8 Creating Dimension res_parnter_address

```
  <record model="olap.database.columns" id="columns_res_partner_address">
            <field name="name">name</field>
            <field name="column_db_name">name</field>
            <field name="type">varchar</field>
            <field name="table_id" ref="table_sales_order"/>
            <field name="active">True</field>
</record>
 <record model="olap.cube.table" id="table_address">
    <field name="name">res_partner_address</field>
    <field name='schema_id' ref='schema_main_sales'/>
 </record>
```

```
<record model="olap.cube.table" id="table_address_country">
   <field name="name">sale_order_country</field>
   <field name='schema_id' ref='schema_main_sales'/>
</record>

<record model="olap.dimension" id="dimension_partner_address_country">
   <field name="name">Sales by Order Address</field>
   <field name="cube_id" ref="cube_sales_order_line"/>
</record>

<record model="olap.hierarchy" id="hierarchy_partner_address_country">
   <field name="name">address_country</field>
   <field name="dimension_id" ref="dimension_partner_address_country"/>
   <field name="primary_key_table">sale_order</field>
   <field name="table_id" ref="table_address_country"/>
</record>

<record model="olap.level" id="level_address_country">
   <field name="name">country_id</field>
   <field name="sequence">1</field>
   <field name="column_name" ref="columns_res_partner_address"></field>
   <field name="column_id_name">country_id</field>
   <field name="table_name">res_partner_address</field>
   <field name="hierarchy_id" ref="hierarchy_partner_address_country"/>
</record>

<record model="olap.level" id="level_address_partner">
   <field name="name">partner_id</field>
   <field name="sequence">2</field>
   <field name="column_name" ref="columns_res_partner_address"></field>
   <field name="column_id_name">partner_id</field>
   <field name="table_name">res_partner_address</field>
   <field name="hierarchy_id" ref="hierarchy_partner_address_country"/>
 </record>
```

Step: 9 Creating Dimension res_user

```
<record model="olap.database.columns" id="columns_res_user_name">
  <field name="name">name</field>
  <field name="column_db_name">name</field>
  <field name="type">varchar</field>
  <field name="table_id" ref="table_sales_order"/>
  <field name="active">True</field>
</record>

<record model="olap.dimension" id="dimension_sales_user">
   <field name="name">user</field>
   <field name="cube_id" ref="cube_sales_order_line"/>
</record>

<record model="olap.cube.table" id="table_sales_res_users">
   <field name="name">res_users</field>
</record>

<record model="olap.hierarchy" id="hierarchy_sales_user">
   <field name="name">user</field>
   <field name="dimension_id" ref="dimension_sales_user"/>
```

```
   <field name="primary_key_table">res_users</field>
   <field name="table_id" ref="table_sales_res_users"/>
</record>

<record model="olap.level" id="hierarchy_sales_user_level">
   <field name="name">name</field>
   <field name="column_name" ref="columns_res_user_name""></field>
   <field name="hierarchy_id" ref="hierarchy_sales_user"/>
</record>
```

Step: 10 Creating Measures Item Sold and Total Sold

```
  <record model="olap.database.columns" id="columns_sale_order_line_product_uom_qty">
              <field name="name">product_uom_qty</field>
              <field name="column_db_name">product_uom_qty</field>
              <field name="type">numeric</field>
              <field name="table_id" ref="table_sale_order_line"/>
              <field name="active">True</field>
  </record>

 <record model="olap.measure" id="measure_item_sold">
      <field name="name">Items Sold</field>
      <field name="cube_id" ref="cube_sales_order_line"/>
      <field name="value_column" ref="columns_sale_order_line_product_uom_qty"></field>
      <field name="value_column_id_name">product_uom_qty</field>
      <field name="table_name">sale_order_line</field>
      <field name="agregator">sum</field>
 </record>
 <record model="olap.measure" id="measure_total_sales">
      <field name="name">Total Sold</field>
      <field name="cube_id" ref="cube_sales_order_line"/>
      <field name="value_column" ref="columns_sale_order_line_price_unit"></field>
      <field name="value_column_id_name">price_unit</field>
      <field name="table_name">sale_order_line</field>
      <field name="agregator">sum</field>
</record>
```

Part VI

Part 6 : Using CommandLine Interface (CLI)

CHAPTER

SIXTEEN

COMMAND LINE INTERFACE

16.1 Introduction to the cli

The CLI - Command Line Interface

Command Line Interface will be the equivalent of psql, but it will be used for MDX queries on the cube. As

```
Welcome to OpenObject BI , the interactive terminal.
Communication: XML-RPC.
Type: \? for help with MDX commands
     \e for execute the MDX query
     \d for quit
BI-terp=#
```

Currently we can run use cli for testing different queries and it can also be used for running automated test.

The basic syntax for testing query is: Syntax

./tinybi.py -d <<Database Name>> -H localhost -U <<User Name> -W <<Password>> -p <<Port Number>> -s <<Schema Name>> -c <<MDX Query>>

One can check all these options by typing

$python tinybi.py –help

Usage: tinybi.py [options]

Options:

```
--version             show program's version number and exit
-h, --help            show this help message and exit


General options:
  -c COMMAND, --command=COMMAND
                      The query to execute
  -s SCHEMA, --schema=SCHEMA
                      The schema to use for the query


Connection options:
  -d DATABASE, --database=DATABASE
                      Database name
  -H HOSTNAME, --hostname=HOSTNAME
                      Server hostname
  -U USERNAME, --username=USERNAME
```

```
                    Username
-W PASSWORD, --password=PASSWORD
                    Password
-p PORT, --port=PORT
                    Server port
```

Example

./tinybi.py -d “terp” -H localhost -U admin -W admin -p 8069 -s “tinyerp” -c “select {[user].[all]} on rows, {[measures].[credit_limit],[measures].[count]} on columns from res_partner”

This will give output on CLI as :

	credit_limit	count
All user	[66700.0]	[21.0]

CHAPTER

SEVENTEEN

RUNNING AUTOMATED TEST CASES

We can check all the queries on *CLI* by making a file with queries to be tested with the syntax given in *CLI*

For example we can create *test_query* file with the content like

---

```
DATABASE='terp'

./tinybi.py -d ${DATABASE} -H localhost -U admin -W admin -p 8069 -s "tinyerp" -c "select <br />{[us

./tinybi.py -d ${DATABASE} -H localhost -U admin -W admin -p 8069 -s "tinyerp" -c "select <br /> {[m
```

The file can be run on command prompt and it will give output for queries and error if its not able to run it on

The output of this file can be viewed CubeCliExample test_query

CHAPTER

EIGHTEEN

REPORTS

The report generated for the *test_query* file is:

```
                credit_limit       count
All user        [66700.0]          [21.0]


                 All user
                                    Root                Administrator       Demo User
credit_limit     [66700.0]                              [2700.0]            [49000.0]
count            [21.0]                                 [2.0]               [6.0]
```

CHAPTER

NINETEEN

WEBSERVICE INTERFACE

Another component of OpenObject - BI, enables application access to dimensional data using Web Services. This facility will allow applications to issue XML-based queries across a Web-based connection using XML-RPC. This support will be based more on XML query languages rather than on OLAP language.

By providing a flexible, platform-neutral way for rendering diverse data types, XML has become a standard for exchanging information across heterogeneous applications. Web services, a set of XML based protocols for finding and communicating between loosely-coupled, internet callable application "services" have therefore become the preferred mechanism for integrating heterogeneous applications and enabling service architectures.

Emphasising on XML and Web services for data exchange and integration provides significant IT benefits including flexibility, interoperability and reach.

Part VII

Part 7 : Using Cube Browser

CHAPTER

TWENTY

CUBE BROWSER

Cube Browser of Open Object-BI Platforme, aims at allowing user to design his own Reports through a clean and effective Web-based environment.

It provides user with a friendly layer to create MDX Query on his previously designed OLAP Cubes with just few mouse operations, and retrieve the desired information in a comprehensible tabular layout. He can further re-format the layout, to drill to fetch further fine information and easily analyze it to understand business currents and causes.

And if they don't require the full power of query capabilities, users can simply explore information in existing reports—formatting and interacting with data to meet his reporting needs.

20.1 Goal behind Cube Browser

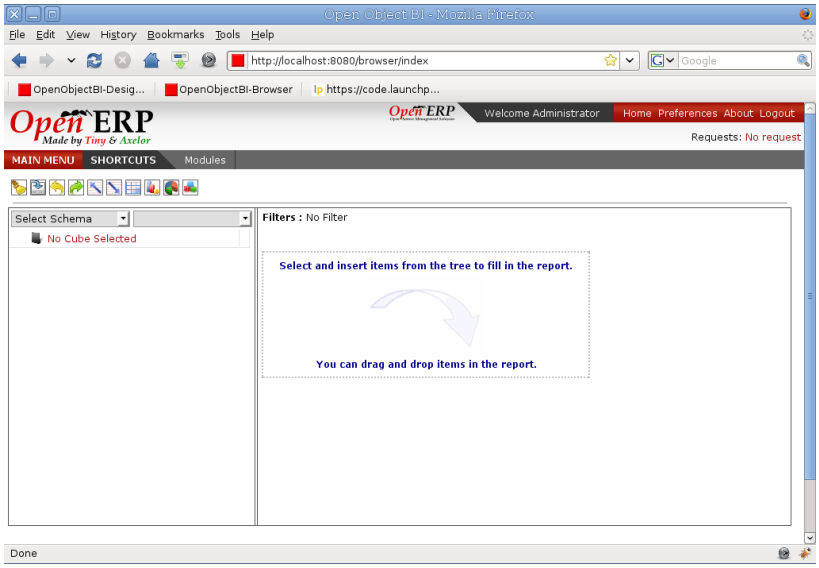
Provide user with a thin client to make reporting far more efficient. Let user use full power of MDXAlchemy engine. Wrapping the process of designing queries into a lot easier steps like clicks, drags and drops. Keeping the interface as simple yet powerful reporting tool.

CHAPTER

TWENTYONE

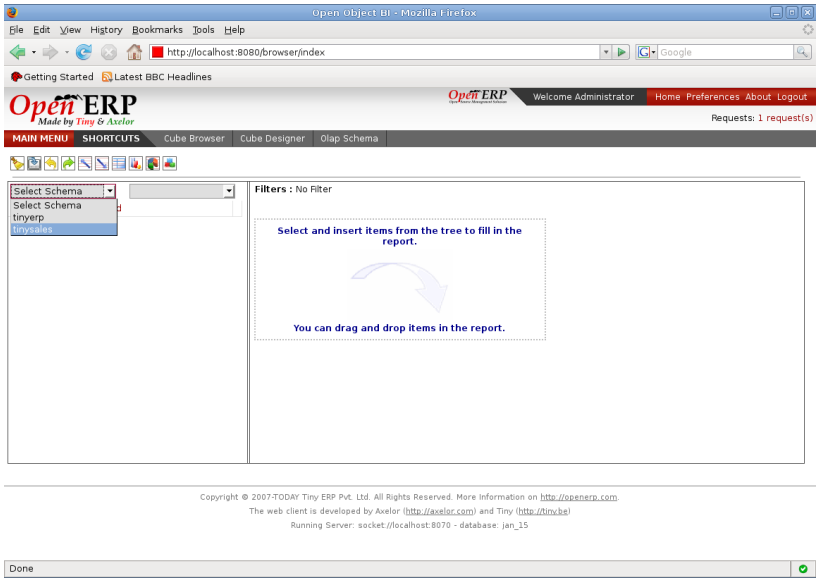
LOADING THE CUBE

Cube Browser will provide the selection of the schema and cube. It come handy with toolbar to perform some common operation in one click

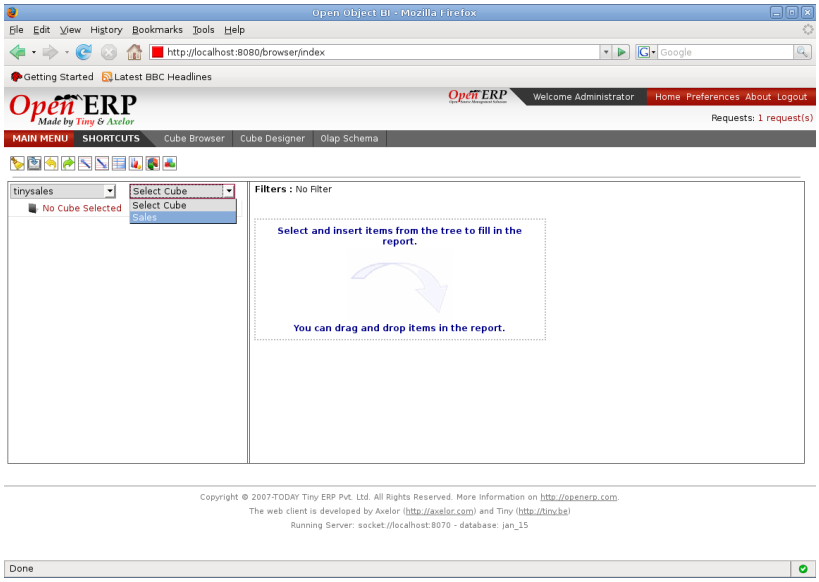


With the help of toolbar we can undo redo the queries, we can switch between the graph and grid view, we can exapand all or collapse all elements.

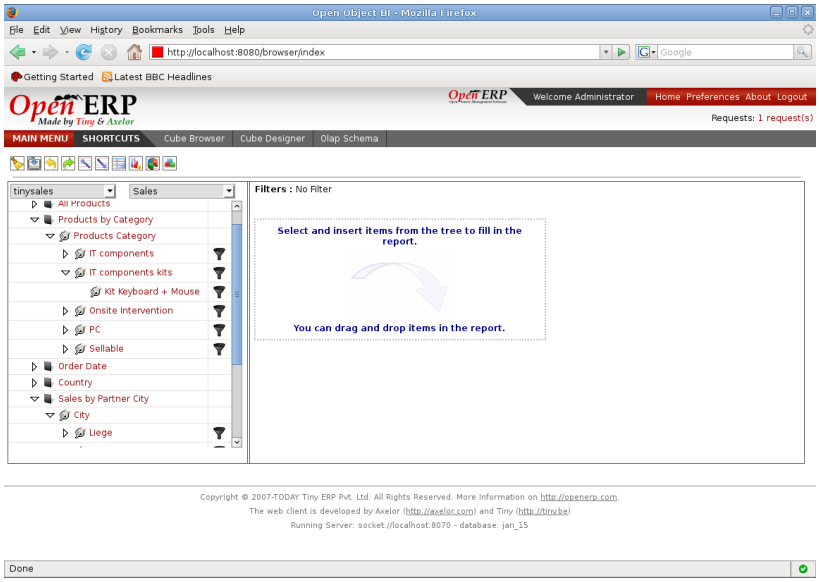
Schema combo box will show all the schema made.



Once we select the schema next combo box will show all the cube that belong to selected schema.



Once we select the cube, the cube is loaded in the tree form in the space below. We can expand each node and it will dispplays to the depth of the level configured.


Open Object BI - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/browser/index
Getting Started
Latest BBC Headlines
Open ERP
Made by Tiny & Axelor
Welcome Administrator
Home Preferences About Logout
Requests: 1 request(s)
MAIN MENU
SHORTCUTS
Cube Browser
Cube Designer
Olap Schema
tinysales
Sales
All Products
Products by Category
Products Category
IT components
IT components kits
Kit Keyboard + Mouse
Onsite Intervention
PC
Sellable
Order Date
Country
Sales by Partner City
City
Liege
Filters : No Filter
Select and insert items from the tree to fill in the report.
You can drag and drop items in the report.
Copyright © 2007-TODAY Tiny ERP Pvt. Ltd. All Rights Reserved. More Information on http://openerp.com.
The web client is developed by Axelor (http://axelor.com) and Tiny (http://tiny.be)
Running Server: socket://localhost:8070 - database: jan_15
Done

CHAPTER

TWENTYTWO

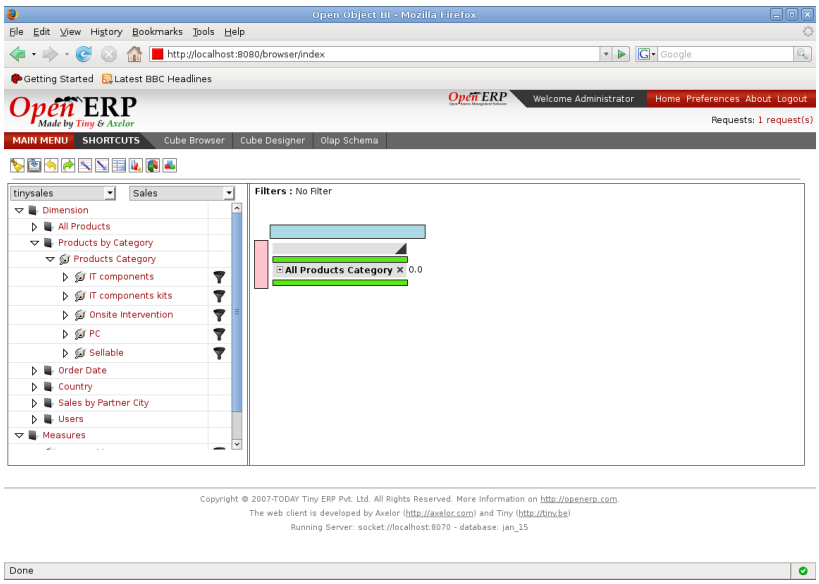
CREATING YOUR REPORT WITH DRAG AND DROP

Cube Browser uses basic drag-and-drop operations to add data to a report. Measures represent categories of stored values; Dimensions represent categories of OLAP information

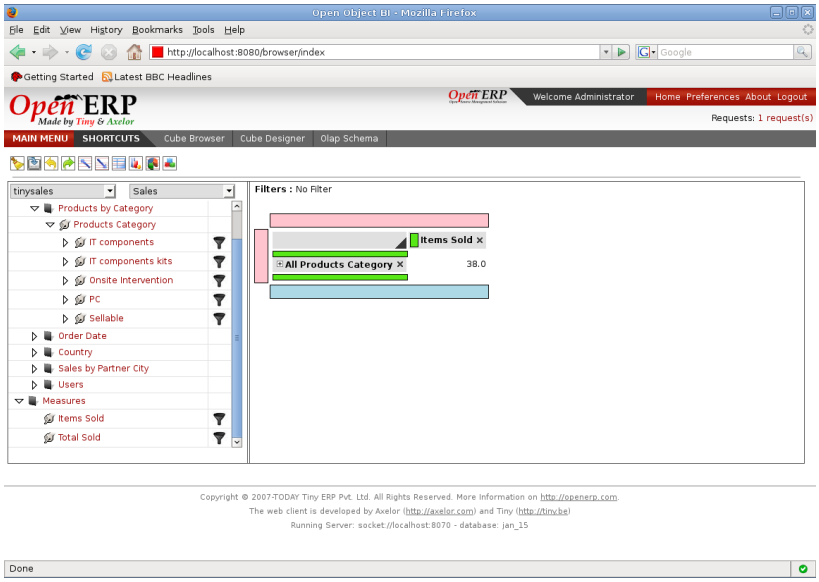
All UI controls update their contents automatically, and the resulting query is displayed on the OLAP Grid.

Queries can be created by Dragging a member on the on the Droppable Area marked with the box for the drop zone.

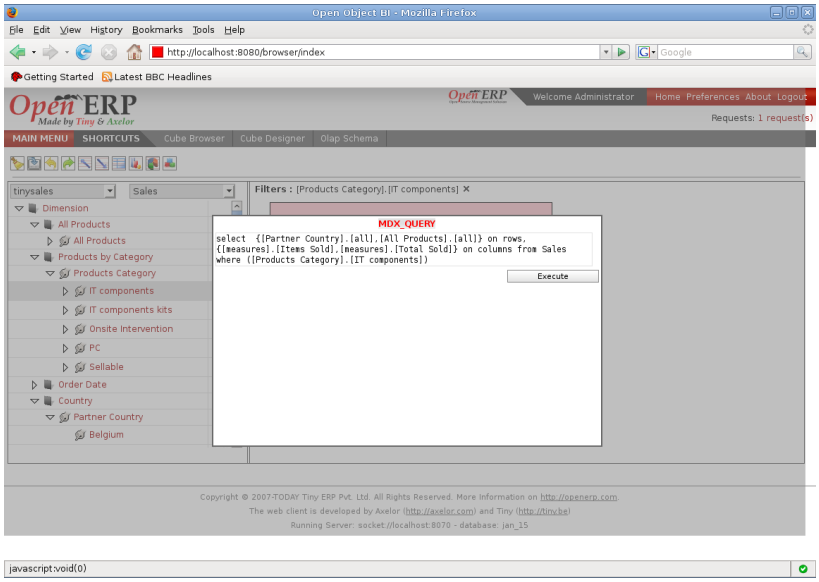
Lets drop the All Product Category on the drop zone refered as grid after ward. Immediately as the member is dropped resultant query is formed and can be viewed using the toolbar. Query made is executed giving the first output on the Grid. The first drop is always on the rows.



Second Axis can be added by dropping a member on blue zone. The output can be seen immediately. Each user action does a change in query, execute and shows the result. The cross or delete button beside each item in grid allows to delete the elements from the query and the resultant grid. On top we can see it any filters applied on the query or not.



We can see the query by selecting on the MDX button on the toolbar. This will open the pop up to show the mdx query for the current grid.



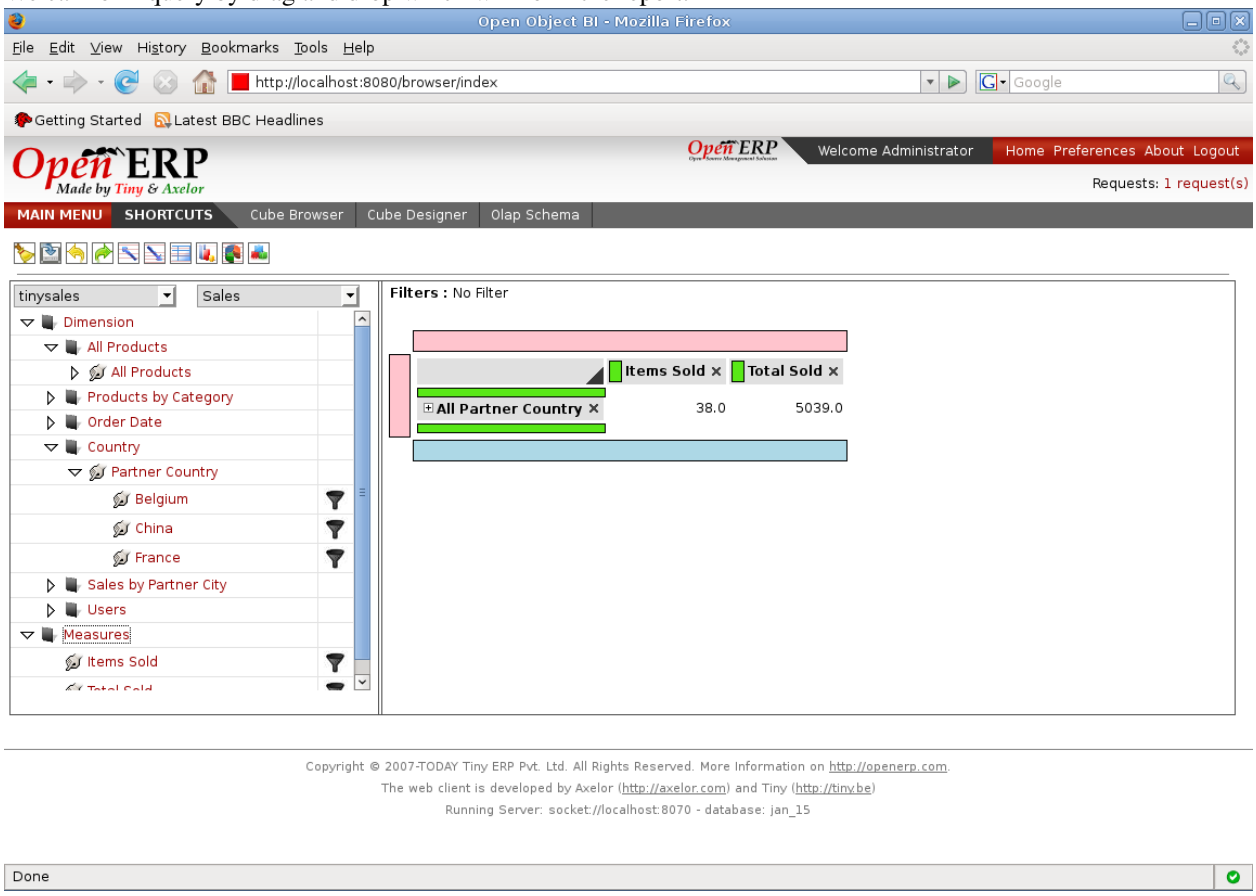
CHAPTER

TWENTYTHREE

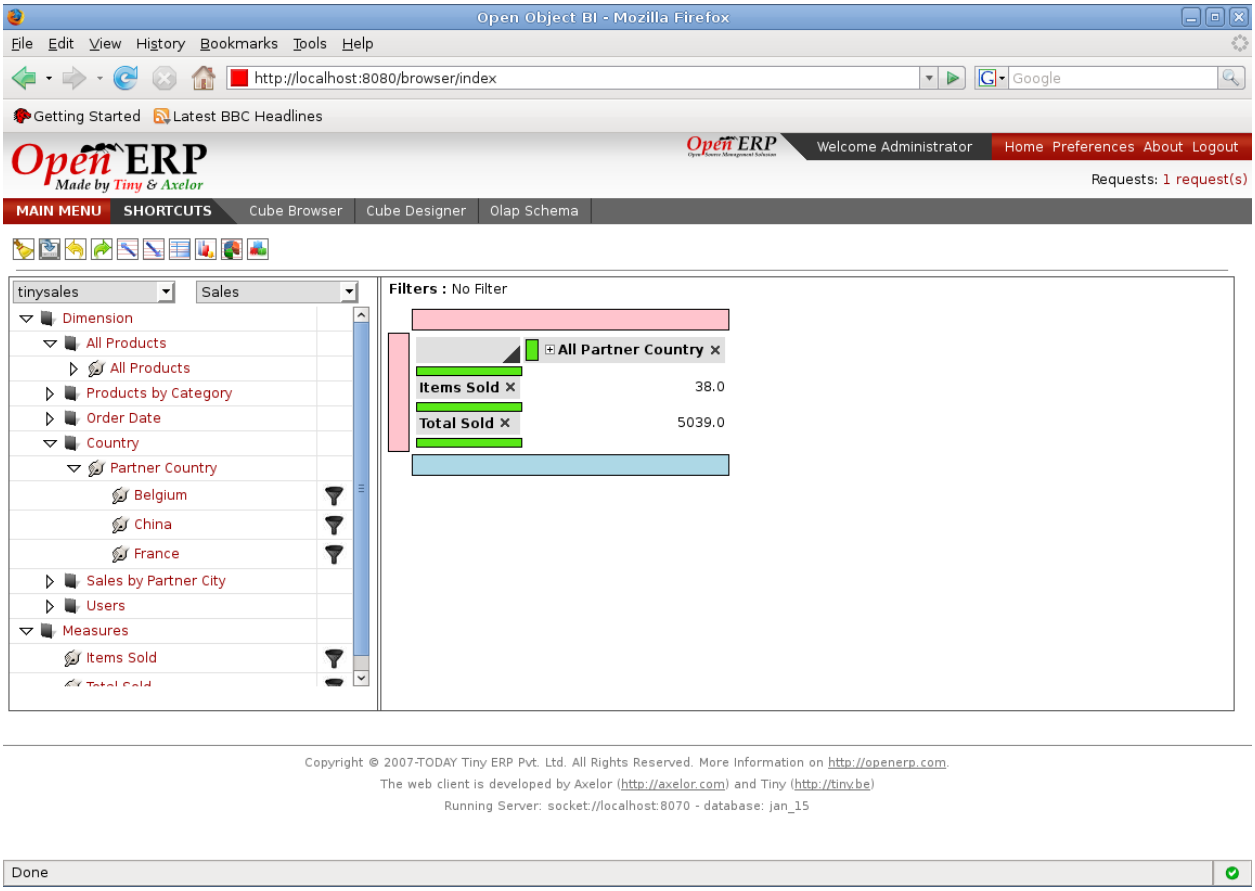
SWAPPING ROWS AND COLUMNS

Once the report is generated in the cube browser we can swap it so the rows moves in to the columns area and column moves in to the rows area. This will make swapping in query part too.

We can form query by drag and drop which will form the report.



Once the report is generated, we can swap which will form the report which is shown below



Now after swapping, we can move on with drag and drop to form queries and generate report.

CHAPTER

TWENTYFOUR

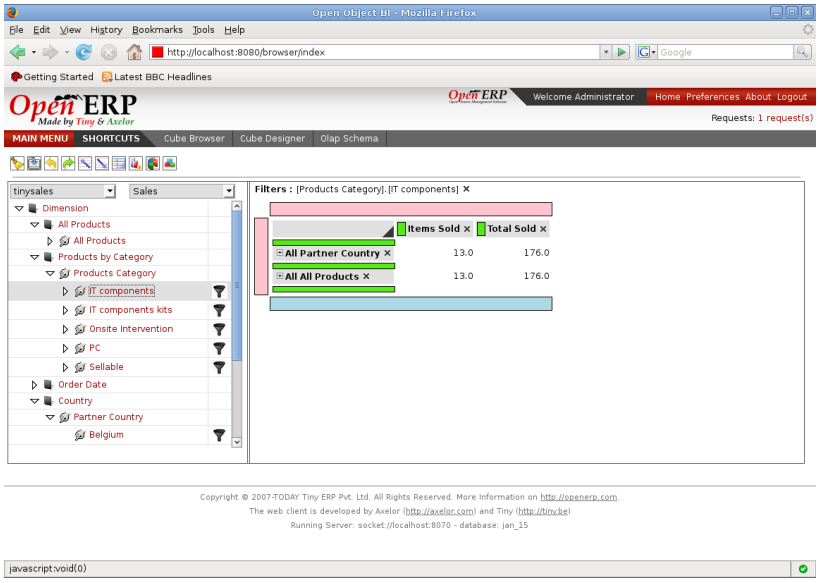
ADDING SLICER

A Slicer is filtering on the data fetched by a MDX Query.

It can be added by clicking on filter images on right side on the members.

Filtering gives user a powerful tool for slicing the multidimensional data for organizing and analyzing in more detail.

This filtering of the data allows user to get the required information to the lowest level and analysed the same so to get the best use of his cube's multidimensional data.



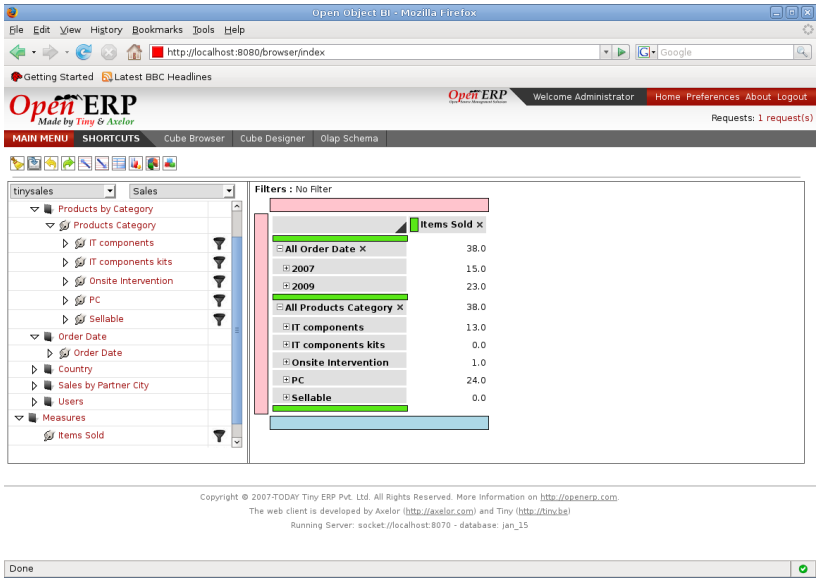
CHAPTER

TWENTYFIVE

DRILL UP/DOWN

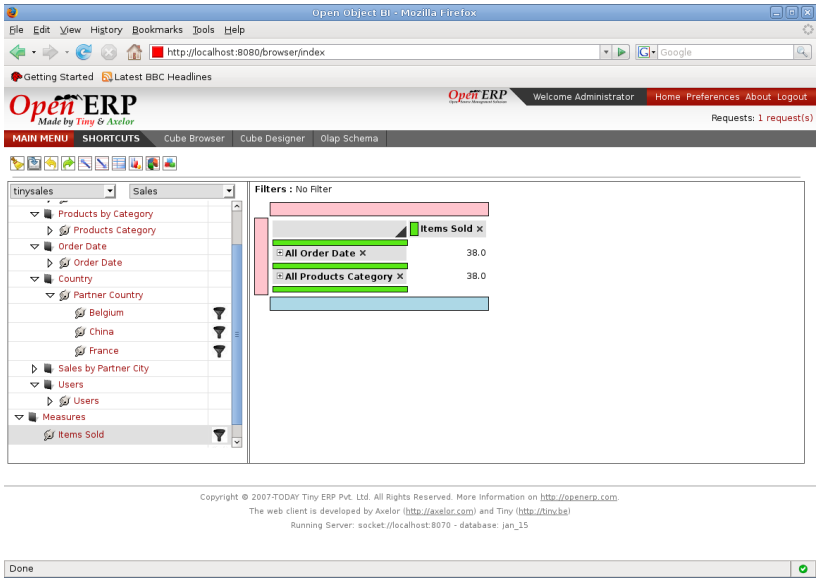
25.1 Drill Down on a Member

Drilling down is the process of expanding a member to its child member for a purpose of broader analysis on data. User can locate the member he wants to drill down. Just a click on that member will expand the parent member to display its child members.



25.2 Drill Up on a Member

Drill up causes a member to wrap up all this children if displayed. Just a click on an expanded parent member will wrap all its child members.


Open Object BI - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/browser/index
Getting Started
Latest BBC Headlines
Open ERP
Made by Tiny & Axelor
Welcome Administrator
Home Preferences About Logout
Requests: 1 request(s)
MAIN MENU
SHORTCUTS
Cube Browser
Cube Designer
Olap Schema
tinysales
Sales
Products by Category
Products Category
Order Date
Order Date
Country
Partner Country
Belgium
China
France
Sales by Partner City
Users
Users
Measures
Items Sold
Filters : No Filter
Items Sold
All Order Date
38.0
All Products Category
38.0
Copyright © 2007-TODAY Tiny ERP Pvt. Ltd. All Rights Reserved. More Information on http://openerp.com.
The web client is developed by Axelor (http://axelor.com) and Tiny (http://tiny.be)
Running Server: socket://localhost:8070 - database: jan_15
Done

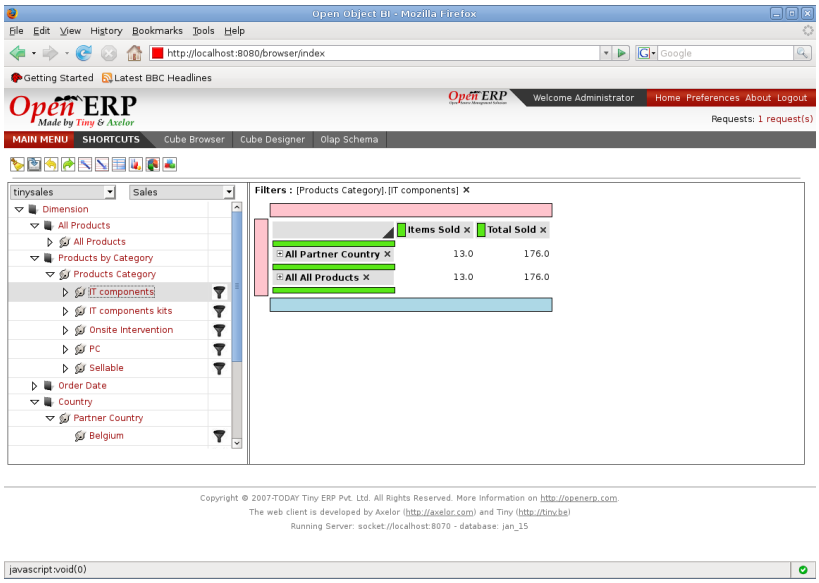
CHAPTER

TWENTYSIX

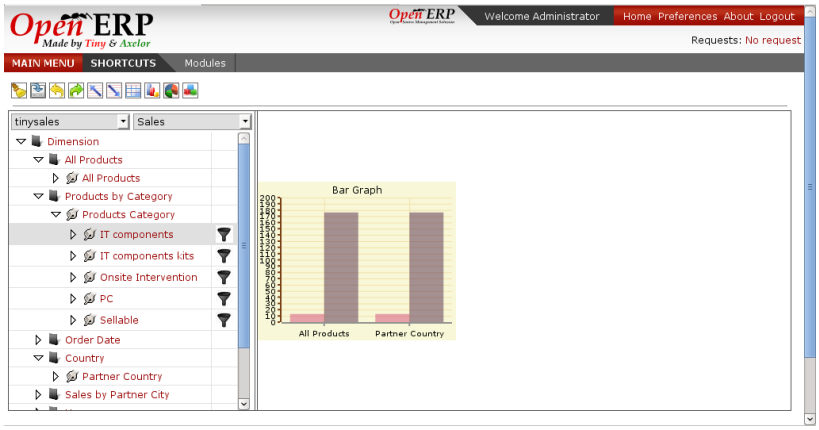
GRAPHS

Graphs presnts the grid output on the graphs or graphical way by plotting on the axis.

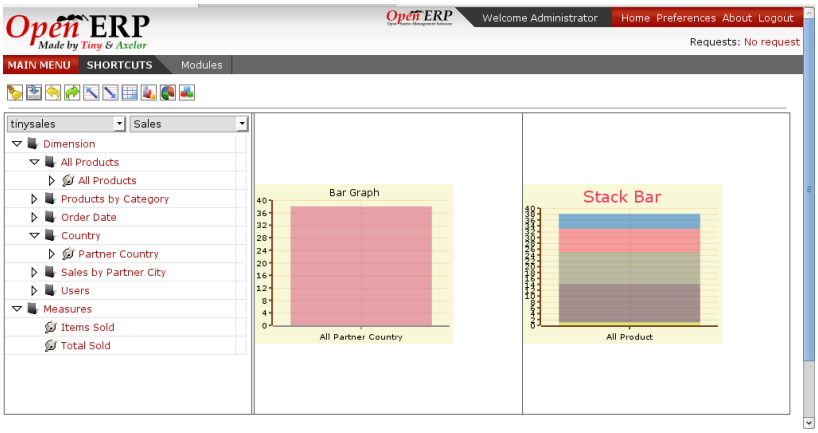
Lets take the case:



For the above grid when we will generate graph it will be:



When we have childrens expanded for the one element on the axis and the other is not drilled. Their will be two graph to show the same


Open ERP
Made by Tiny & Axelor
Welcome Administrator
Home Preferences About Logout
Requests: No request
MAIN MENU
SHORTCUTS
Modules
tinysales
Sales
Dimension
All Products
All Products
Products by Category
Order Date
Country
Partner Country
Sales by Partner City
Users
Measures
Items Sold
Total Sold
Bar Graph
All Partner Country
Stack Bar
All Product

Part VIII

Part 8 : Using Cube Designer

CHAPTER

TWENTYSEVEN

GOAL OF CUBE DESIGNER

The goal is to develop a User Friendly Cube Designer for Open Object - BI that allows a user to define and / or modify an OLAP cube definition starting from any database. (Oracle, MySQL, PostgreSQL). This has to be user friendly so that a end-user can define his own cube on his own database without any development knowledge.

27.1 Basic features

The cube designer of the OpenObject – BI Solutions helps the User to Create New cubes and Modify the existing cubes. First the User checks the connection with its database and if he is connected after that only the user can create cubes in two ways #Wizard Flow #Generic Flow.

27.1.1 Wizard Flow

In the wizard flow a wizard will guide the user through the entire process of cube creation. Navigation can be done through Next and Previous button.

Note: *Clicking on the "Save" button on every form causes the data to be written in the database and simultaneously the "Next" button is activated and the user is navigated to the next form. Next button will not be activated until the data is "Saved".*

27.1.2 Generic Flow

The cube can be modified / created by the user in a normal way.

Modify / Create A Schema

The user specifies the desired schema name. He selects the desired database or creates it with the help of [new]. He specifies the schema description. He saves it.

Modify / Create A Fact Table

User Makes a perticuler Type for Fact table He select desired database or Schema for perticular Fact Table and also create by its own [new] button.

Modify / Create A Database

User specifies the "General Parameters" He specifies the "Connection Parameters" that specify which database is used for the connection with which port number. He tests the connection for error and the "Connection URL" is generated. On connection string being correct the new database is created.

Modify / Create A Cube

The user provides desired cube name along with the fact tables and schema name. The user can select the already created fact tables via a drop down box or can create a new fact table by clicking on [new]. Same goes for schema too. The dimensions and measures at this point will be empty as they are not created as of now.

Modify / Create A Dimension

The user provides with the dimension name. The cube name that was provided by him will appear in the drop down box. He can select a cube name from the list else he can create a new cube by clicking on [new]. Hierarchies are absent.

Modify / Create A Hierarchies

The user provides the hierarchy name. The dimension name will come in the dropdown box. User can create a new dimension by clicking on [new]. User will provide with the hierarchy field name,sequence,hierarchy type,all member and default member fields. User will give the fact table by selecting it from a drop down box or by creating a new fact table altogether by clicking on [new]

Modify / Create A Levels

The user has to specify the level name,column name,column id,level class,table name,sequence and hierarchy. Hierarchy will appear in the drop down box. He can create a new hierarchy by clicking on [new].

Note: *Clicking on the "Save" button on every form causes the data to be written in the database. Double Click on row opens modification window of respective record.*

Modify / Create A Measures

The user provides with the Measure name. The cube name that was provided by him will appear in the drop down box. He can select a cube name from the list else he can create a new cube by clicking on the [new]. It defines the all calculation / aggregation with fact column name. Here all calculation / aggregation are interdepended with the fields of fact column name, aggregator, data type and format of string.

CHAPTER

TWENTYEIGHT

DATABASE INTROSPECTION

CHAPTER

TWENTYNINE

DEFINING CUBE

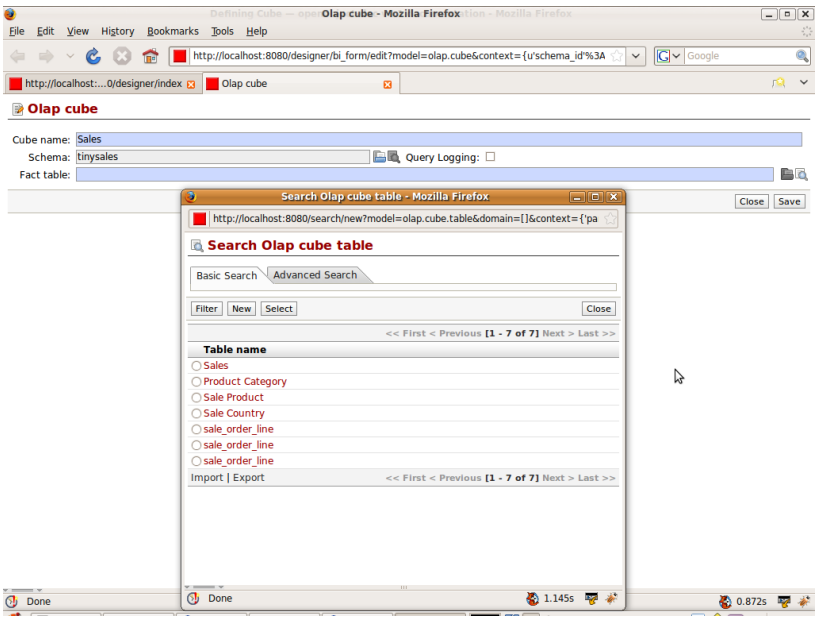
Cube as we discussed in *The Cube*

For making the cube we need

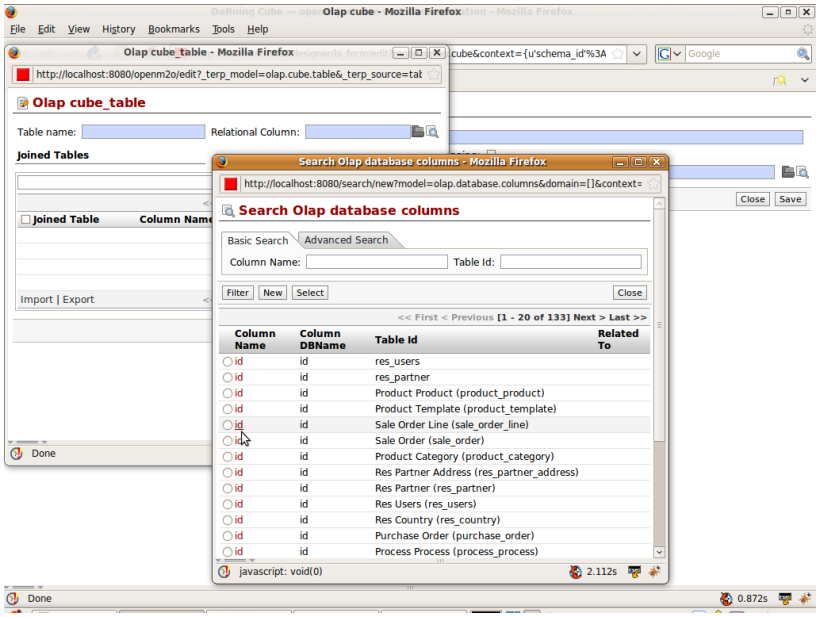
1. Cube Name : Meaningfull name for the cube
2. *Schema*: *Configuring Schema*
3. Query Logging : It will decides whether the query will be logged or not.
4. Fact Table : A table that contains the facts (measures) and the link to the other tables that in turn qualifies for the dimensions. The fact table can be a join of two or more tables.

For example : To analysing the sales data we will take join of sale_order and sale_order_line as a fact table.

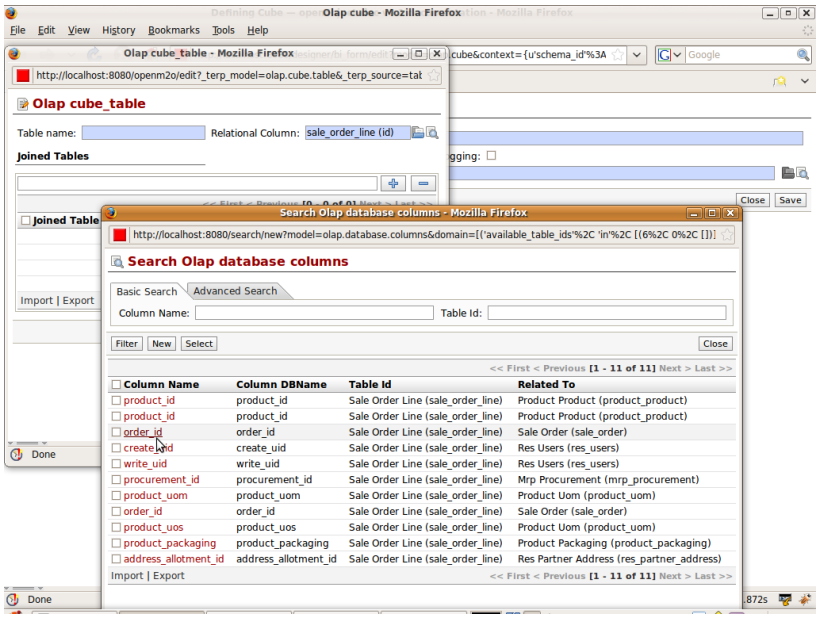
So we start making the cube. The screen shows the new cube window. The schema name will be filled by default depending on the schema we are browsing in the tree. The initial winodw will show the relational column for fact table which will open the search box of all the fact table created so far.



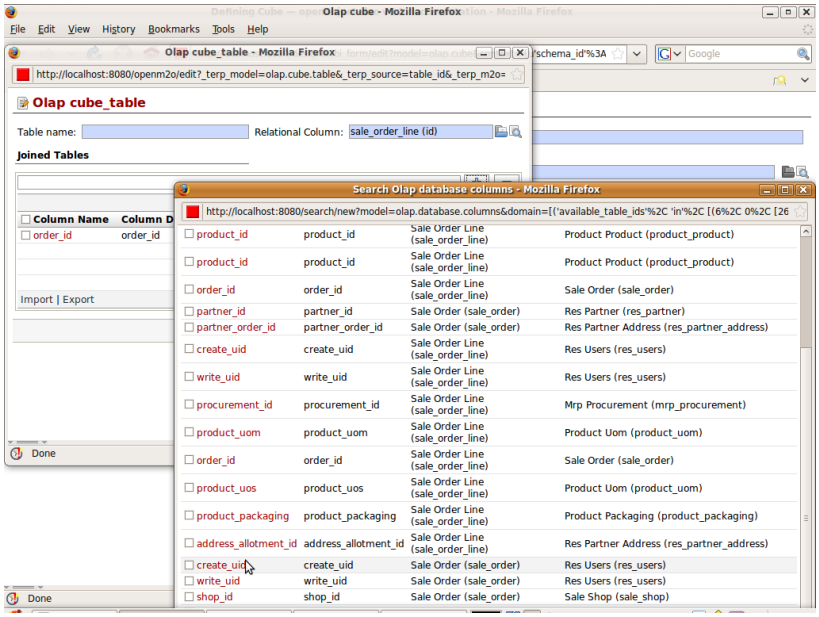
We go for creating the new fact table. In the relational column it will show all the primary keys for all the table loaded in the introspection. We select sale_order_line table with primary key id. Once we are done with the selection we will go for joining the table.



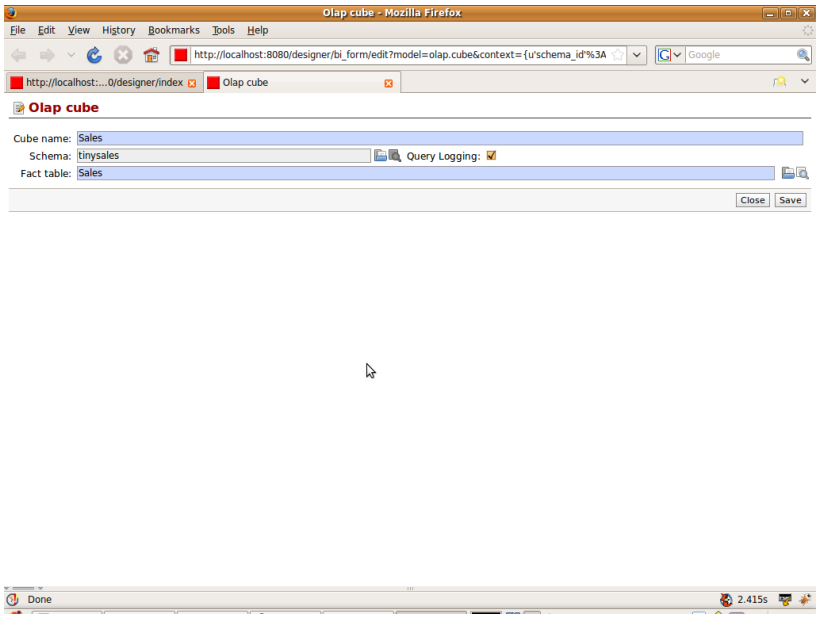
Join Table will open all the tables that are referenced from the sale_order_line, according to the table name from the relational column. We select order_id as it is the field related to the sale_order. Hence, we made the join of the sale_order_line and sale_order



If we want more table to be joined we can do that by adding more on Join Tables. Now the search will be on all the reference from sale_order and sale_order_line



Finally the cube page will look like

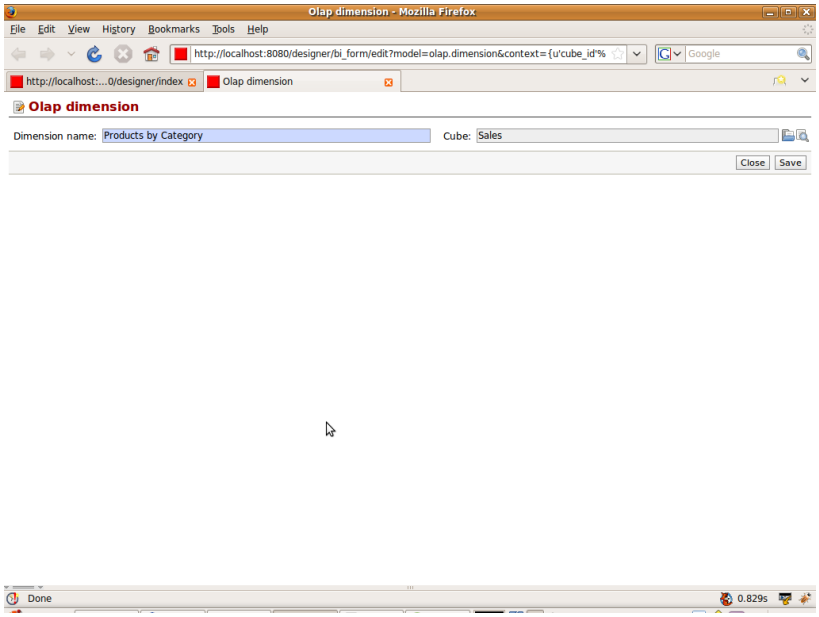


CHAPTER

THIRTY

DEFINING DIMENSION

Dimension need to name to be identified. Cube will be filled by default according to the schema opened.



CHAPTER

THIRTYONE

DEFINING HIERARCHY

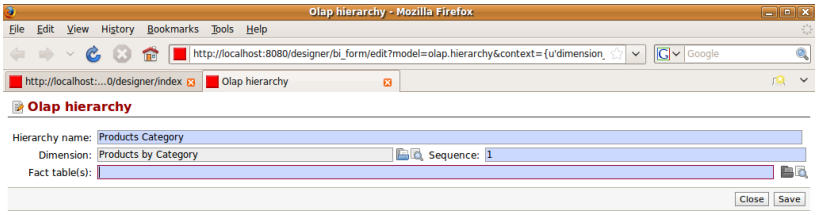
Hierarchy is the arrangements of objects, peoples etc .. in a ranked or some series. The hierarchy are the way of arranging the dimensions.

It need the fact table.

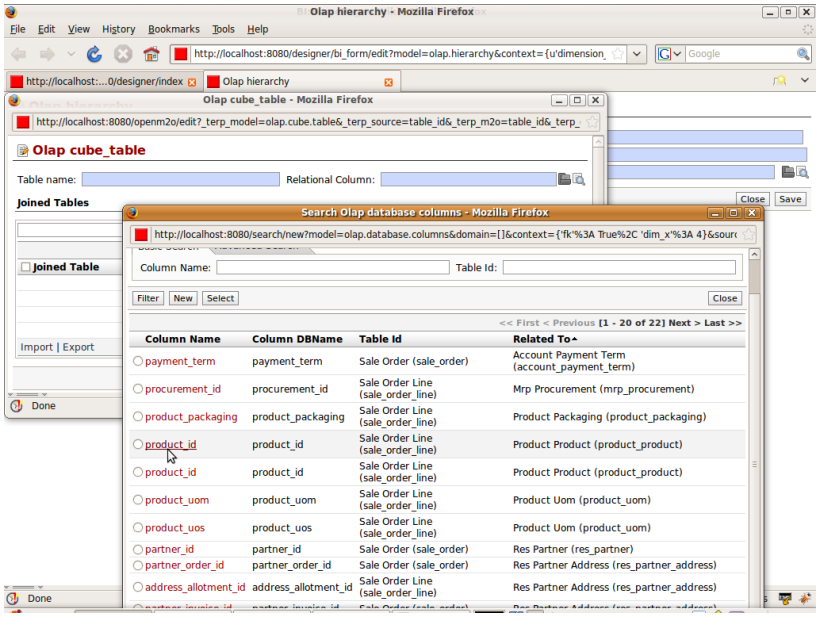
Lets take the Example of Products.

We want our sales cube to work on the products. Means we want to see the products sold. We have divided the products in to the category. So we will make the Product Hierarchy to display products by categories.

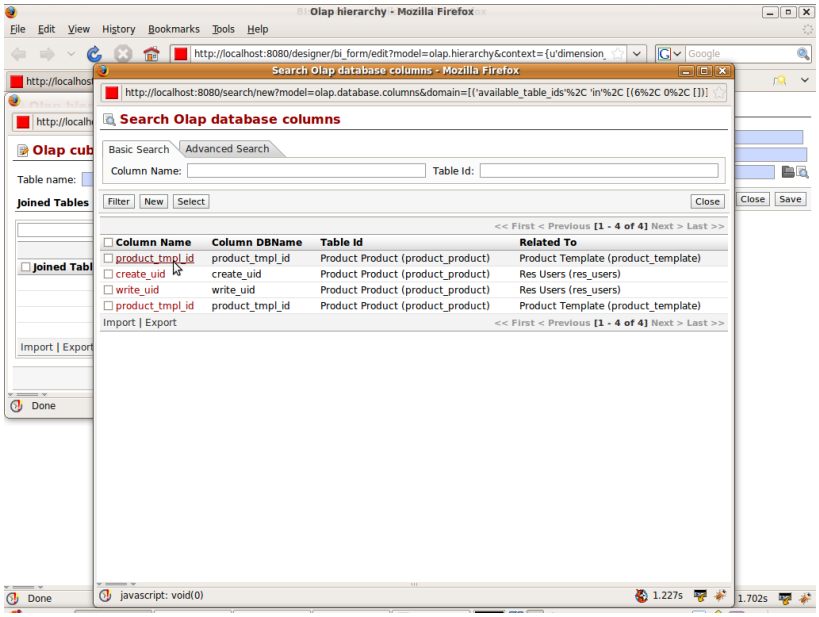
Lets see the new hierarchy. According to the dimension some values are set by default like hierarchy name and dimension.



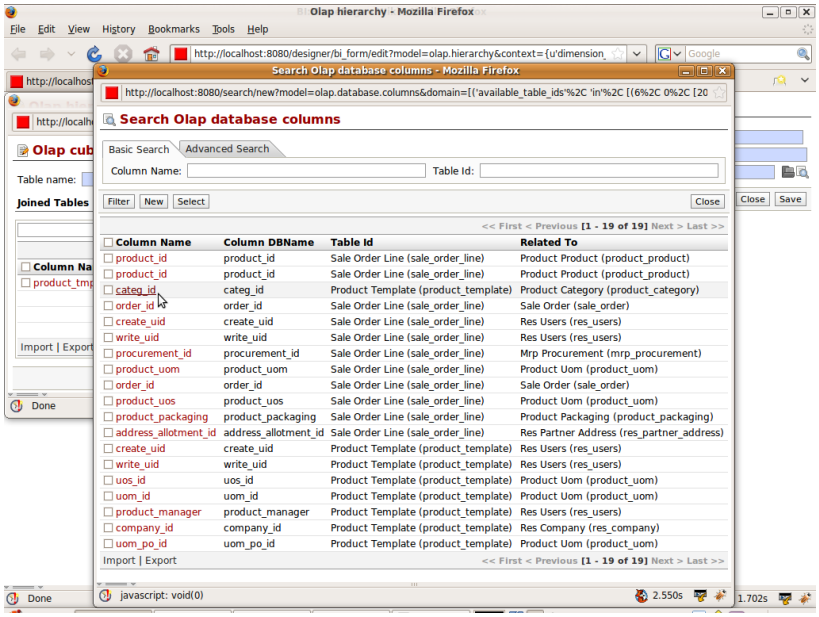
We now move to define the fact table for the hierarchy. Relational column will show the all fields of the sale_order_line and sale_order. As these are the fact tables for the cube. We select product_id from sale_order_line which is related to product_prodcut



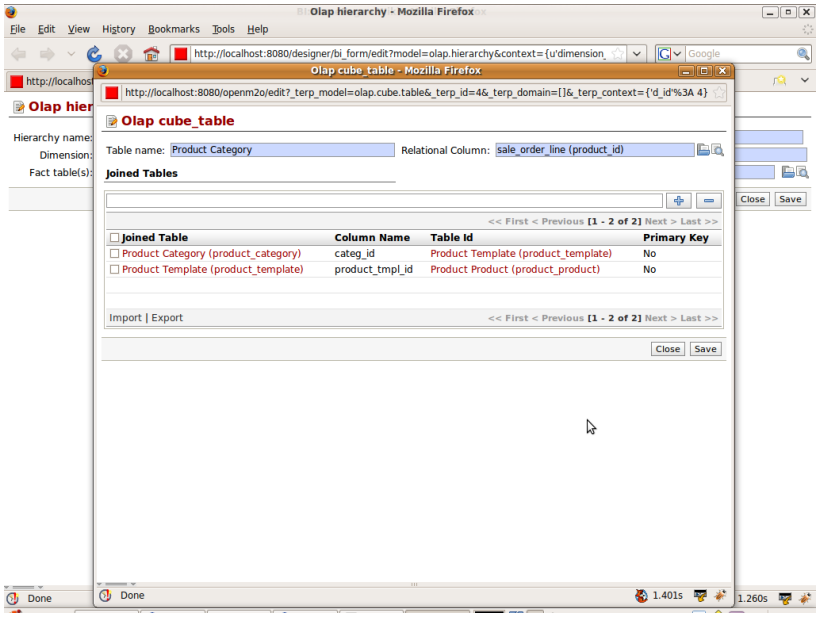
We want to get the product name and the category both. We know the product_category will give category and product_template will give the name. Now the list is filtered accordingly for adding the join tables



After selecting the product_tmpl_id we move to select the category table.



So final fact table for the Product Category will be



CHAPTER

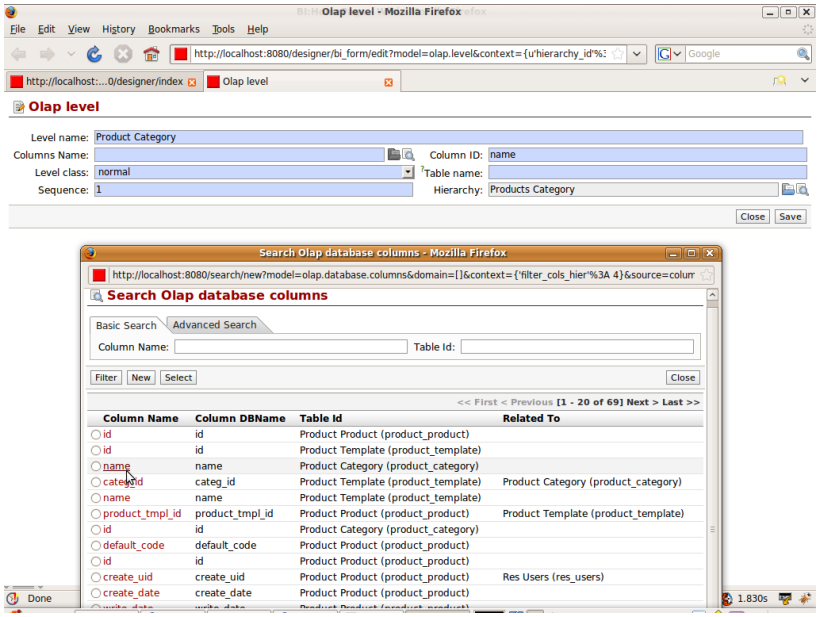
THIRTYTWO

DEFINING LEVEL

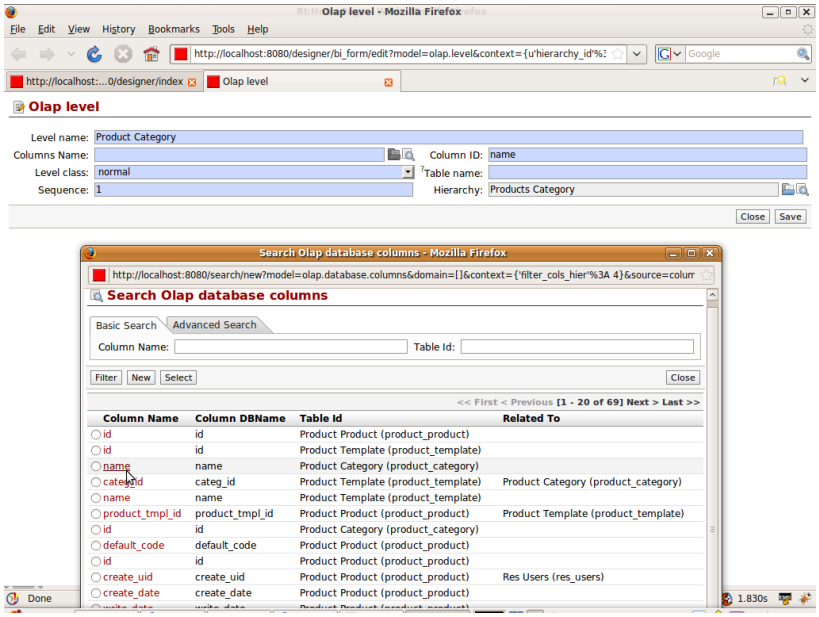
Level It specifies the actual data to be displayed. It specifies the depth for the hierarchy. Now taking the same example for the Prodcuts. We need it to be two level depth.

1. Product Category
2. Product Name

Lets start with making the levels. Details like hierarchy name, level name filled by default according the schema we are in. We need to specify the column to be used for filling the level. We open the column name and it will show all the fields from tables defined in the hierarchy. We select name from the product category



We want more level for displaying the name of the products. In the column name we will select the column name from the product_template. The main thing is to change the sequence to 2. This will show the prodcuts category wise on the browser.


Olap level - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/designer/bi_form/edit?model=olap.level&context={u'hierarchy_id'%3
http://localhost:...0/designer/index
Olap level
Olap level
Level name: Product Category
Columns Name:
Column ID: name
Level class: normal
Table name:
Sequence: 1
Hierarchy: Products Category
Close Save
Search Olap database columns - Mozilla Firefox
http://localhost:8080/search/new?model=olap.database.columns&domain=[]&context={'filter_cols_hier'%3A 4}&source=colum
Search Olap database columns
Basic Search Advanced Search
Column Name: Table Id:
Filter New Select Close
<< First < Previous [1 - 20 of 69] Next > Last >>
Column Name Column DBName Table Id Related To
id id Product Product (product_product)
id id Product Template (product_template)
name name Product Category (product_category)
categ_id categ_id Product Template (product_template) Product Category (product_category)
name name Product Template (product_template)
product_tmpl_id product_tmpl_id Product Product (product_product) Product Template (product_template)
id id Product Category (product_category)
default_code default_code Product Product (product_product)
id id Product Product (product_product)
create_uid create_uid Product Product (product_product) Res Users (res_users)
create_date create_date Product Product (product_product)
Done
1.830s

CHAPTER

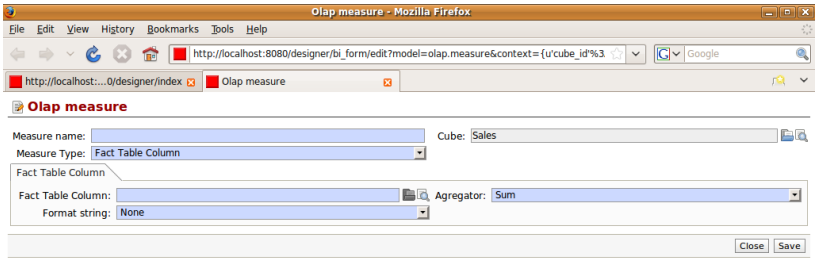
THIRTYTHREE

DEFINING MEASURE

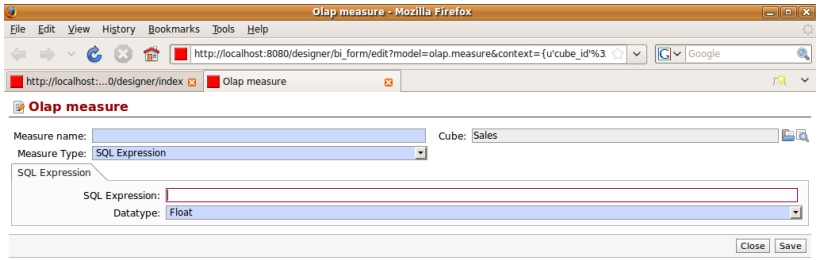
Measure are the fact or quantitative values. It comes from the fact table configured in the cube.

We will make the measure for the same example. Measure type specifies whether it will be column base or sql expression based.

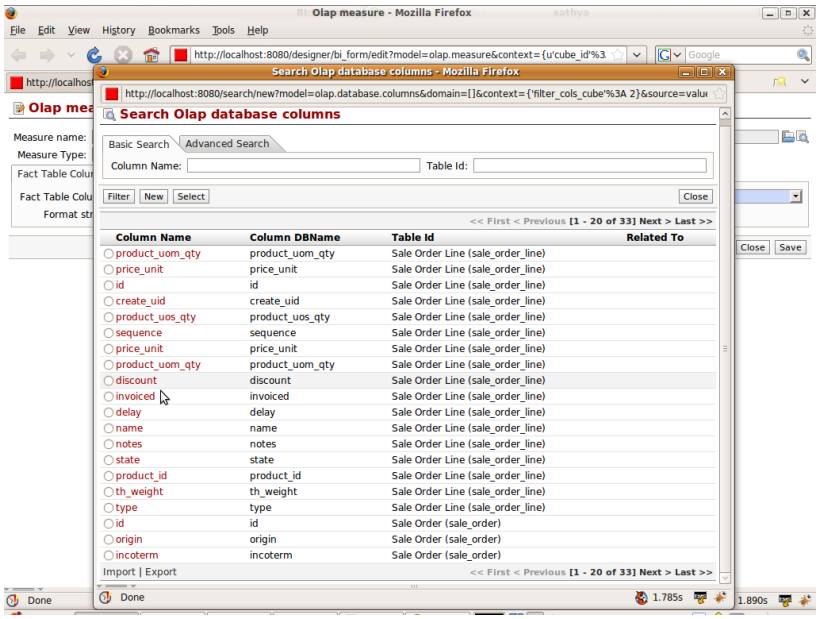
Column Based:



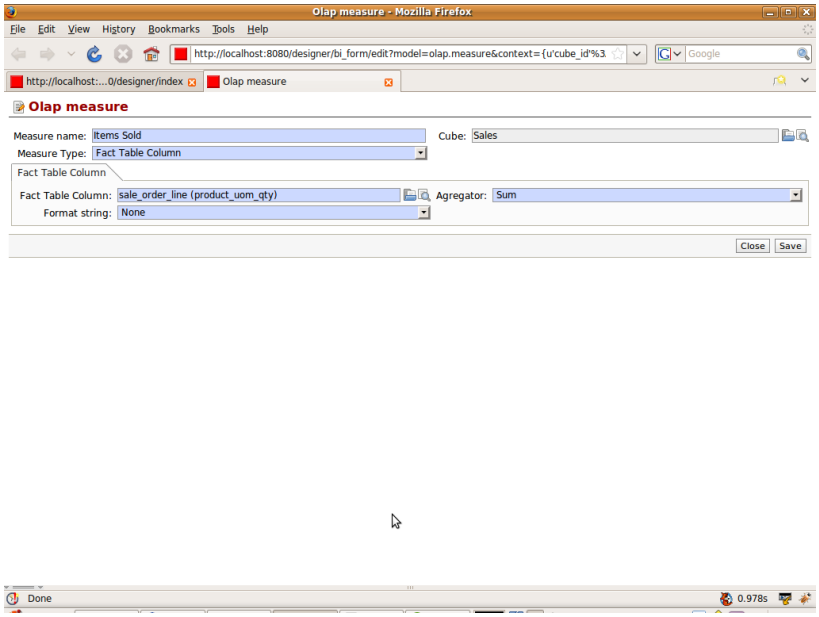
SQL Expression Based:



In the column based measure we will see select the fact table column from the columns of the fact table define in the cube i.e. sale_order and sale_order_line.



So finally measure will lool like:


Olap measure - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/designer/bi_form/edit?model=olap.measure&context={u'cube_id'%3
Google
http://localhost:...0/designer/index
Olap measure
Olap measure
Measure name: Items Sold
Cube: Sales
Measure Type: Fact Table Column
Fact Table Column
Fact Table Column: sale_order_line (product_uom_qty)
Agregator: Sum
Format string: None
Close
Save
Done
0.978s