Logical Design of Data Warehouses from XML

Marko Banek, Zoran Skočir and Boris Vrdoljak
FER – University of Zagreb, Zagreb, Croatia
{marko.banek, zoran.skocir, boris.vrdoljak}@fer.hr

Abstract—**Data warehouse is a database that collects and integrates data from heterogeneous sources in order to support a decision making process. Data exchanged over the Internet and intranets has recently become an important data source, having XML as a standard format for exchange. The possibility of integrating available XML data into data warehouses plays an important role in providing enterprise managers with up-to-date and relevant information about their business domain. We have developed a methodology for data warehouse design from the source XML Schemas and conforming XML documents. As XML data is semi-structured, data warehouse design from XML brings many particular challenges. In this paper the final steps of deriving a conceptual multidimensional scheme are described, followed by the logical design, where a set of tables is created according to the derived conceptual scheme. A prototype tool has been developed to test and verify the proposed methodology.**

I. Introduction

Data warehousing system is a set of technologies and tools that enable decision-makers (managers and analysts) to acquire, integrate and flexibly analyze information coming from different sources. The central part of the system is a large database specialized for complex analysis of historical data, called a data warehouse. The process of building a data warehousing system includes analysis of the data sources, design of a warehouse model that can successfully integrate them and later the construction of the warehouse according to the proposed model. Decision-makers use OLAP (OnLine Analytical Processing) tools to put queries against the warehouse in a quick, intuitive and interactive way. OLAP tools use the multidimensional data model, which enables focusing on small pieces of data, generally a few numerical parameters, that are most interesting for the decision making process. Other data in the warehouse are organized hierarchically into several independent groups, called dimensions, and used to perform calculations with the few important parameters.

Data warehouses, owned by big enterprises and organizations, integrate data from heterogeneous sources: relational databases or other legacy database models, semi-structured data and different file formats. Recently, the World Wide Web, Web services and different information systems for exchanging data over the Internet and private networks have become an important data source.

The amount of semi-structured content in the information systems of big organizations has constantly and rapidly been increasing, particularly since the appearance of XML [12] as a format for notating semi-structured data. XML has become a standard de facto for data exchange over the Internet or other network. It is used in e-business: either for business-to-business (B2B) or business-to-customer (B2C) applications, as well as in the e-government projects. XML has also become a standard format for accessing Web services. The possibility of integrating available XML data into data warehouses plays a crucial role in providing enterprise managers with up-to-date and comprehensive information about their business domain. Therefore, the task of defining a methodology for integrating XML data into data warehouses has become inevitable.

In [11] we proposed a methodology for integrating XML data modeled by XML Schemas [13][14][15] into data warehouses. The warehouse design process starts directly from the source XML documents and their XML Schema and includes conceptual, logical, ETL (Extraction, Transformation and Loading) and physical design. Conceptual and logical design are aimed at constructing the warehouse, while the latter two steps refer to populating the warehouse with data and using it optimally. In [9] and [10] the greatest part of conceptual design was described and implemented in a Java-based prototype tool.

In this paper the final steps of the conceptual design and the whole logical design are explained, thus completing the construction of the warehouse. The conceptual model is implemented in a relational database as a star schema. Given the completed conceptual scheme, the prototype tool first creates the logical model automatically and then constructs tables in a database, according to the proposed star schema.

The paper is structured as follows. The multidimensional data model is described in Section II. The methodology for integrating XML data into data warehouses is briefly explained in Section III. In Section IV the principles of rearranging the conceptual scheme are explained and illustrated through several examples. The problem of choosing dimensions and measures and planning dimension hierarchies is described in Section V. Section VI shows how the conceptual scheme is automatically transformed into a star schema. Conclusions are drawn in Section VII.

II. Multidimensional Model

In order to make the data accessible to OLAP and reporting tools and enable efficient analysis of a large amount of data, a multidimensional data model is used in the warehouse. Basic components of the multidimensional model are: fact, measures, dimensions and hierarchies.

A *fact* is a focus of interest for the decision-making process. It typically corresponds to events occurring dynamically in the enterprise world (such as sales or orders, for example).

Measures are continuously valued attributes that describe the fact numerically. During the business analysis their values are used for mathematic calculations that primarily include summing.

Dimensions are mutually independent parameters that describe the business process fact. Every parameter has a discrete domain of possible values. Each fact record is a *primary event*, an occurrence of a fact, defined by an n-tuple of values taken from the domains of its *n* dimensions. Dimensions can be presented as axes of an n-dimensional coordinate system (Fig. 1). Every primary event is represented by a cube. Each cube contains values of measures for that fact record. Primary events correspond to the finest grain level.

The business process can be viewed at different levels of abstraction. For instance, the total purchase cost for "Fresh milk" supplied by "General Milkman" can be calculated daily or monthly. The daily output corresponds to the primary event. Getting the monthly output requires collecting and summing the daily values. The month level is a higher level of abstraction. Fact records (cubes in Fig. 1) are joined together into a larger cube, thus making the grain level coarser. The process of joining primary events is called *aggregation*. One attribute value at a higher level of abstraction joins several attribute values of the lower level. The attribute of the lower level functionally determines that of the higher level. For instance, "January 2005" unites the 31 date values and "the 9th of January 2005" determines the month "January 2005". Functional dependencies imply *hierarchies*, where the cardinality of the relationship is always many-to-one

Two or more levels at different level of abstraction form a hierarchy. We call attributes that express hierarchy levels *level keys*. The key of the lowest level (at the finest grain level), which functionally determines all other attributes, is called the *dimension key*. Hierarchies may also include *descriptive attributes*, which contain additional information about a level of the hierarchy. They are also connected to the level key by a to-one relationship, but unlike other dimension attributes, they cannot be used for aggregation.

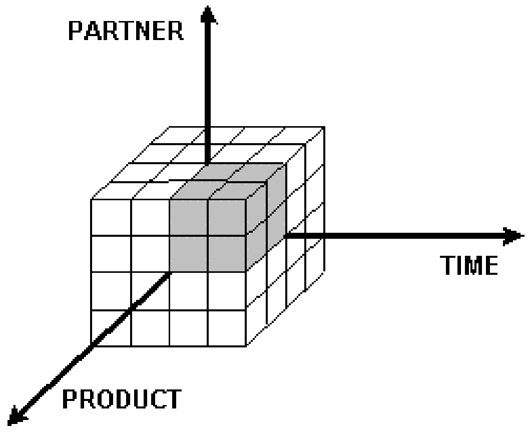


Figure 1. Dimensions and fact records

A measure is *additive* across a dimension if its values can be aggregated by summing, otherwise it is *non-additive*. The sum of the daily purchasing costs for all days of the month will always give the monthly cost, so cost is additive across time. Meanwhile, the sum of the daily account balances in the date warehouse of a bank is not a monthly account balance. An average value should be calculated instead. Bank account balance is not additive across time, but is additive across dimensions that describe types of account or branches of the bank. Such a dimension is called *semi-additive*. The more dimensions a measure is additive across, the more suitable and more often it can be used for calculations.

III. Methodology

In deriving the methodology for data warehouse design from XML sources [11], the methodology for creating a data warehouse from entity-relationship diagrams [1] was changed and adapted in order to address various issues emerging from the semi-structured nature of XML data. The methodology consists of the following steps:

1. preliminary work
   - analyzing the XML Schemas and the conforming XML documents
   - storing XML
2. design
   - conceptual design,
   - workload definition,
   - logical design,
   - ETL design,
   - physical design.

Another approach to designing a warehouse from XML sources is based on translating XML data into a relational scheme, either using DTD [8] or not [3]. Standard methods for designing a warehouse from relational databases are used afterwards. However, insufficient emphasis is given to determining to-one relationships, which express functional dependencies that form hierarchies. In [5] and [6] a technique for data warehouse design starting from DTDs is outlined. Although that approach bears some resemblance to ours, the unknown cardinalities are not verified against the source XML documents, but they are always arbitrarily assumed to be to-one.

Conceptual design consists of transforming the model of the source data into a multidimensional model, which represents the data in a warehouse. The conceptual multidimensional model does not depend on the model of the database used for storing data. Because of the semi-structured nature of XML data, the conceptual design represents the biggest challenge when developing the methodology. Two main issues arise: firstly, not all the needed information can be safely derived; secondly, there are different ways of representing the relationships in XML Schemas and each achieves different expressive power.

Conceptual design starts from the XML Schema that models the source XML documents. The Dimensional Fact Model [2] is adopted as the conceptual model. At the beginning, a *schema graph* (SG), which shows the structure of XML data, is created. XML elements and attributes declared in XML Schema correspond to vertices of the SG. Additional vertices in the SG are operators of cardinality, inherited from DTD, which appear between an element and its sub-element or attribute in case when the sub-element or attribute may appear one or more (operator

"+"), zero or more ("*") or zero or one times ("?"). If the cardinality is exactly one, no operator is inserted.

Relationships in XML Schema can be expressed in two ways: by joining sub-elements and attributes to an element or by *key*/*keyref* mechanism. When joining a sub-element or attribute to an element, XML Schema defines only cardinality of the relationship towards the sub-element or attribute (note that the vertices of the SG in Fig. 2 are connected by directed edges pointing towards the descendant). The cardinality in the other direction can be inferred by examining the content of the XML documents. The *key*/*keyref* mechanism is similar to the mechanism of primary and foreign key in relational databases. The *key*s and the referenced *keyref*s must belong to the same data type.

The algorithm for the semi-automated process of conceptual design was proposed in [4] and [9]. There are four basic steps of the algorithm:

1. Preprocessing the XML Schema
2. Creating and transforming the schema graph (SG)
3. Choosing facts
4. For each fact:
   - 4.1. building the dependency graph (DG) from the SG
   - 4.2. rearranging the DG
   - 4.3. defining dimensions and measures.

All steps except 4.2 and 4.3 were described into details and implemented in our prototype tool.

Steps 1, 2, 3 and 4.1 are performed semi-automatically (in some cases completely automatically) and a basic conceptual scheme that includes functional dependencies, called *dependency graph* (DG), is produced. After creating the SG automatically in steps 1 and 2, the designer of the warehouse chooses the fact among the vertices and relationships of the SG (step 3), using her/his knowledge on their semantic meaning. Step 4.1 is performed semi-automatically and a basic conceptual scheme that includes functional dependencies, called *dependency graph* (DG), is produced. During the process of creating the DG, relationships with cardinality to-one (which includes one and zero-or-one relationships) are recursively added into the DG, as they correspond to functional dependencies. In cases when the cardinality of a relationship cannot be read from the SG (i.e. from an XML element towards it parent) the content of XML documents must be examined and the designers knowledge on semantics is required. XML documents are examined using queries in XML Query language [16].

In this paper we focus to steps 4.2 and 4.3 of the proposed algorithm. After completing the step 4.3, the conceptual design of the warehouse is finished, with a conceptual scheme as a result. In the logical design, a set of tables is created in a database according to the derived conceptual scheme.

We have used a real-life example to verify the proposed methodology of integrating XML data into the data warehouse. The Open Applications Group (OAG) is a non-profit organization that supports e-business and electronic exchange of data. A major grocery store company in Croatia uses the Purchase Order document from OAG Integration Specification, version 7.2.1 (OAGIS 7.2.1, [17]). According to the XML Schema, each order document (the root element of the order is PROCESS) consists of a single header and one or many line items. The header contains the order ID (POID), order date (DATETIME and its sub-elements) and the business partner (i.e. supplier). Beside its ID (LINENUM), each line item contains data about one purchased product (UPC, DESCRIPTN) and the purchased quantity. The schema graph is shown in Fig. 2.

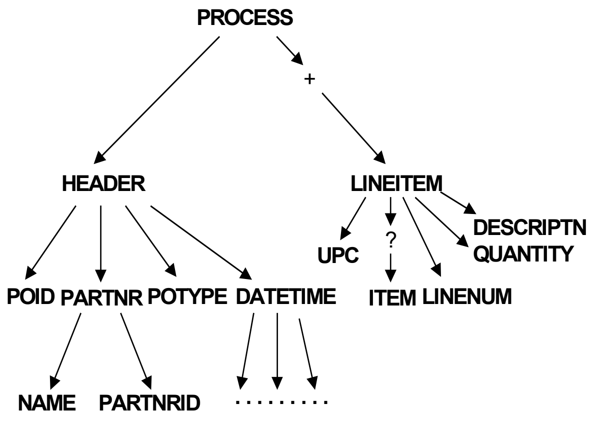


Figure 2. Schema graph for OAGIS 7.2.1. Purchase Order

The initial DG is shown in Fig. 3. The gray vertex, LINEITEM, is chosen as the fact and becomes the root of the DG.

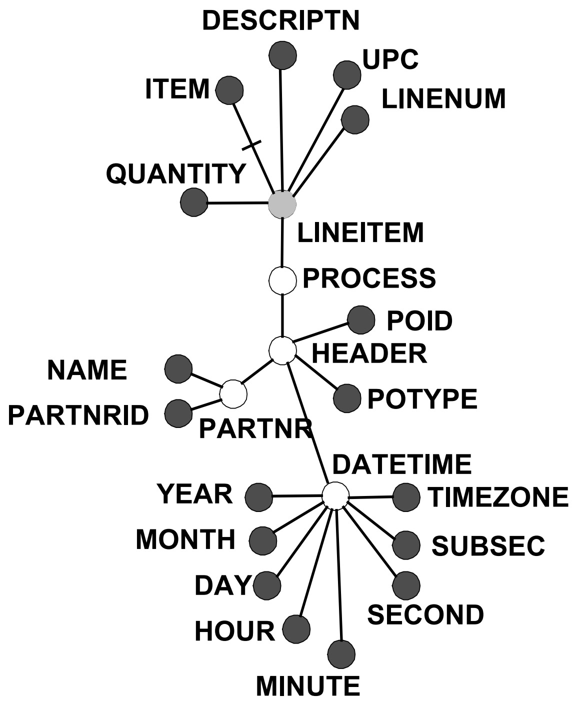


Figure 3. Dependency Graph

IV. Rearranging a Dependency Graph

The semi-automated algorithm creates the DG by recursively navigating relationships between vertices of the SG. Any time a to-one relationship is reached, it is added to the DG without checking its semantic meaning. Therefore, the data warehouse designer should check all vertices in the DG. The wrong conceptual scheme results in storing unnecessary information in the warehouse or, worse, loosing important ones. Creating a usable and efficient conceptual scheme of a DW, the designer usually has to:

- remove some existing vertices from the DG,
- add some new vertices to the SG,
- change the position of some existing vertices in the DG.

All described methods of rearranging the DG are implemented in the prototype tool.

A. *Removing Vertices*

Every vertex of the final conceptual scheme must conform to a database attribute of the logical scheme. All vertices that express no content to be stored to a database must be removed from the DG. In XML, the content may either be an attribute value or the text content of an element. Elements that have sub-elements and/or attributes but no text content only express the logical structure of a document. The XML Schema definition of such an element, PARTNR, is shown in Fig. 4. In our prototype tool vertices with no content are shown as white circles, while those with content can be seen as dark ones.

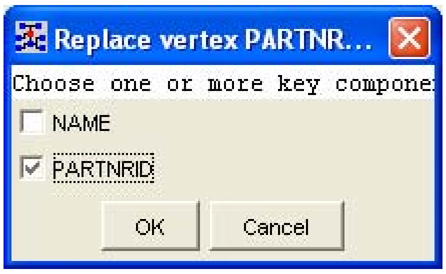
It is the warehouse designer's duty to remove all such vertices from the DG. This operation can be done in two ways: by replacing the vertex with one of its descendants or by simply grafting it.

```
<xsd:element name="PARTNR">
 <xsd:complexType>
  <xsd:sequence>
    <xsd:element name="PARTNRID"
               type="xsd:positiveInteger"/>
    <xsd:element name="NAME"
                type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

Figure 4. Defining an element with no content in XML Schema

Replacing a vertex ***v*** with its descendant ***w*** is possible if ***w*** functionally determines all other descendants of ***v***. Besides, ***w*** must have a text content. Consider vertex PARTNR in Fig. 3, which describes the supplier being dealt with when purchasing goods. Its child PARTNRID is the identity code of the company and functionally determines the other descendant of PARTNR. When replacing a vertex, the prototype tool asks which of its descendants will come in its place (Fig. 5). PARTNR is removed and, PARTNRID becomes child of HEADER. NAME becomes child of PARTNRID. This part of the DG can be seen in Fig. 6, as presented by the tool.

Grafting of a vertex is performed when none of its descendants determines the rest of them. In most cases the descendants are semantically independent and belong to different dimension hierarchies. When grafting a vertex ***v***, which has ***u*** as a parent, the entire sub-graph with root in ***v*** is connected directly to its parent ***u*** and ***v*** is eliminated. As a result, the aggregation level corresponding to ***v*** is lost. On the other hand, all descendant levels are maintained.



Choose one or more key compone

NAME [✓ PARTNRID]

Figure 5. Replacing an element with no content

The most interesting case is the one when the grafted vertex is child of the root (the fact vertex) and has more than one descendant. In such case, the number of dimensions in the conceptual scheme may increase. Consider vertices PROCESS and HEADER in Fig. 3. First, PROCESS is grafted and HEADER becomes child of the root vertex LINEITEM. After grafting HEADER, all its children: PARTNRID (note that we have already removed PARTNR), DATETIME, POID and POTYPE become dimension candidates. They are not only functionally but also semantically independent and some of them will become separate dimensions.

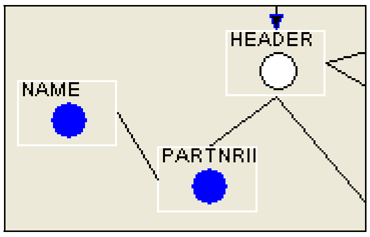


Figure 6. The element has been replaced

The designer may remove any of the remaining vertices that have text content if she/he considers them unnecessary for multidimensional analysis. For instance, the exact time an order has been created is described by DATETIME and its children. The finest grain level for the time dimension is day, so all vertices giving more detail about time (HOUR-SUBSEC, TIMEZONE) should be removed as uninteresting. LINENUM is also left out (Fig. 7) because it gives no useful information.

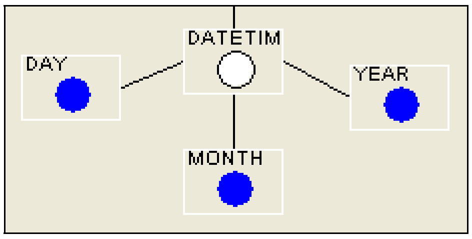


Figure 7. DATETIME with unnecessary vertices removed

B. *Adding New Vertices*

The source XML Schema often does not contain some important measures or dimensional attributes, which means that the designer should add them during the rearranging process. The DG in Fig 3. does not contain any vertex that represents price of the ordered goods. Prices are stored separately in an operational database of the grocery store company. The price vertex is added to the DG, suggesting that we are going to integrate data from different sources: operational (relational) databases and semi-structured (XML) data.

When adding new vertices to the DG, the designer must be aware that all of them must match some content. The content of a new vertex can be obtained by:

- using "foreign" content from other sources different from the XML documents matching the starting XML Schema,
- using content of other vertices of the DG,
- giving the vertices an "artificial" value.

Using "foreign" content from other data sources includes classical databases (the warehouse is designed for a large company that has one or more operational bases, either relational or even hierarchical), semi-structured data (other XML documents, either for business information exchange with partners or for internal use within the company) or Web services. We have already mentioned the PRICE vertex as an example of using "foreign" content. Web services may be used when the required values change frequently. Consider a European industry corporation that sells its products in the EU and the USA, earning both a euro and a dollar income. The business analysis requires all income to be comparable. The dollar prices must be expressed in euros, which requires the currency exchange rate for the day the products have been sold. Existing Web services give both temporary and historical exchange rates for different currencies. Web services are accessed and the calculated data returned via XML documents, so our warehouse design method for semi-structured data sources can be used to integrate the return data.

When the content of a vertex is obtained from other vertices, transforming functions must be created during the ETL design process. The main measure for the purchase order fact table is the amount of money paid for the purchased goods. Therefore a COST vertex is inserted in the DG. Its value is always the product of the unit price of the purchased product (PRICE) and the number of units (QUANTITY).

Numbers from an artificial sequence list are typically used if the inserted vertex is a dimension key. In that case it is not important to know the value of the vertex, but to distinguish the record it represents from other records. Dimension keys generally have integer values when implemented in a relational database.

V. Defining Dimensions and Measures

Defining dimensions and measures, which also includes formal specification of hierarchies, is the last step of the conceptual design.

A. *Planning dimensions in the dependency graph*

Data warehouse is a set of periodically taken snapshots of the relevant data in an enterprise information system. Therefore, every data warehouse must contain a time dimension, while the choice of other dimensions depends on the fact. The principles of rearranging a DG, described in Section IV, will be used when planning the time dimension. After removing the vertices that gave too many unnecessary details, the vertex DATETIME has three children attributes of integer datatype describing day of month (DAY, with values ranging between 1 and 31), month of year (MONTH) and YEAR. None of those attributes can be the key of the time dimension. The key must describe a date, the unique combination of the three mentioned attributes. A new TIMEKEY vertex is added to DATETIME, which is subsequently removed and replaced by TIMEKEY. TIMEKEY is an integer; when implementing the conceptual scheme as a star schema in a relational database, all dimension keys are artificially generated integers in order to save memory space. We put an integer key, PRODKEY, as the key of the product dimension instead of UPC. PARTNRID, which is an integer, is renamed to PARTKEY.

There will be four levels in the hierarchy: *day*, *month*, *quarter* and *year*. TIMEKEY is the key of the time dimension and of the day level. Key attributes of other levels will be MONTHKEY, QTRKEY and YEAR, which are functionally determined by TIMEKEY. The day, month and quarter level key also get additional vertices. The DATE vertex on the day level describes date instead of the integer TIMEKEY.

The rearranged conceptual scheme, as shown by the GUI of the prototype tool, is presented in Fig. 8.

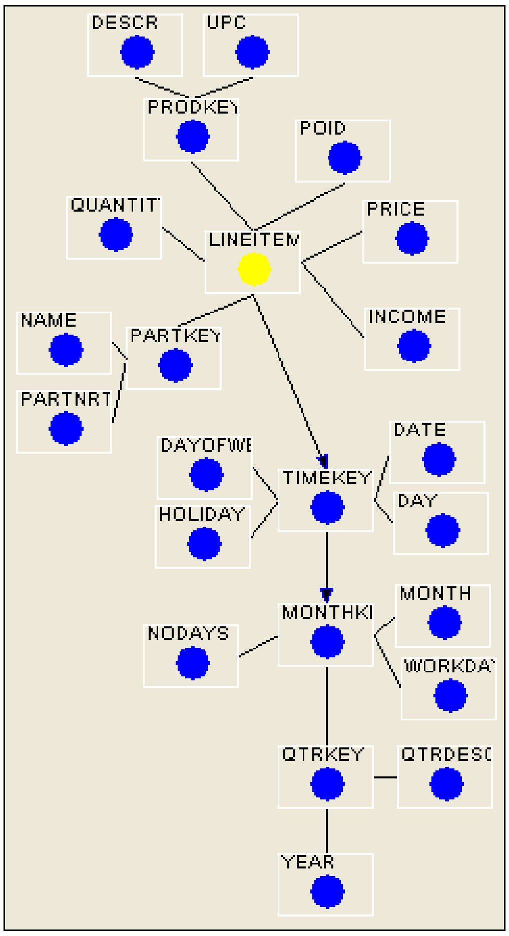


Figure 8. Rearranged DG in the prototype tool

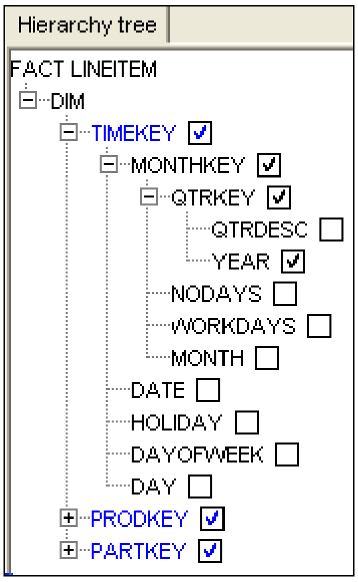
B. *Defining dimensions and measures in the prototype tool*

The prototype tool does not allow defining dimensions and measures if vertices representing elements without text content are still present in the current DG. We can define dimensions and measures after completing the rearranging process.

All children of the fact vertex of the DG either become dimension keys or measures. All vertices that have its own children (TIMEKEY, PRODKEY, PARTKEY) imply a hierarchy so they are set as dimension keys for the time, product and partner dimension, respectively. QUANTITY and COST are additive across all created dimensions. PRICE is not additive across time and partner. Viewing from the point of conceptual design, it could also be a descriptive attribute in product dimension. However, when implementing a star schema in a relational database, many authors (for instance [7]) suggest that parameters whose value often changes should be stated as measures. The value of the POID vertex identifies the purchase order. This is a discrete, non-additive constraint parameter, joining all line items that form the same purchase. This will be the fourth dimension of our conceptual model.

The tool automatically detects all children of the fact vertex that have descendants. Such vertices are automatically stated as dimensions. For each child of the fact without descendants the tool asks the designer whether it should be a dimension (like POID) or a measure (like PRICE, QUANTITY and COST).

The designer uses the GUI of the prototype tool to create the hierarchy metadata in dimensions. The dimension keys are stated as the keys of the finest grain level for all dimensions. The tool offers their children vertices to become the next level key. The process repeats recursively for the children of each last selected level key vertex. In Fig. 9, the checked vertices are selected as level keys.


Hierarchy tree

```
FACT LINEITEM
⊟ DIM
  ⊟ TIMEKEY ☑
    ⊟ MONTHKEY ☑
      ⊟ QTRKEY ☑
          QTRDESC ☐
          YEAR ☑
        NODAYS ☐
        WORKDAYS ☐
        MONTH ☐
      DATE ☐
      HOLIDAY ☐
      DAYOFWEEK ☐
      DAY ☐
  ⊞ PRODKEY ☑
  ⊞ PARTKEY ☑
```

Figure 9. Creating hierarchies

In our example, the product and partner dimension have only one hierarchical level, so aggregation cannot be done across those two dimensions.

VI. Creating the Logical Scheme

The conceptual scheme is implemented in a relational database. The star schema model is used for creating the logical scheme. It defines one central table, the fact table, describing the fact, and a set of dimensional tables. Each dimension table has a single-part primary key: its dimension key attribute. Generally, every of the *n* dimensions of the conceptual model correspond to one of the *n* dimensional tables. The fact table contains all measures of the fact and a multi-part key, each part referencing a dimensional table as foreign key.

In certain cases, the number of dimensional tables in the star schema may be smaller or larger than the number of dimensions in the conceptual model. The dimension POID has a single attribute. It would be useless and memory space consuming to create a separate dimension for POID because it would not reference any other attribute but itself. Single attribute dimensions are called degenerate dimensions in the star schema model.

It is the matter of data and their granularity whether a degenerate dimension remains a part of the fact table key or is left out, becoming a measure. If a partner can receive multiple orders for the same product in one day, as it happens in our example, POID must remain a part of the key.

If only one order from the same partner in one day were possible, or if only one of the orders might contain the same product, POID could become a measure. In that case POID would be left out of the fact table key. Such discrete measures are not additive across any dimension.

The PRICE attribute is stated as a measure, although it could possibly also be a descriptive attribute of the product dimension. Values of dimension attributes should change as rarely as possible. Any change of a single dimension record requires a new record to be inserted. As prices change often, the dimension would contain many records describing the same product. The speed of performing a query against a star schema depends on the number of joins that must be done. Smaller number of keys that have to be joined gives a better performance. On the other hand, the fact table requires far more storage space than all dimension tables together. Defining too many measures in a fact table would increase its size and deteriorate performance. Generally, attributes whose values often change should become measures, and those whose values change never or rarely should be left as dimensional attributes [7].

At the end, the logical scheme contains three dimensional tables (Fig. 10) and the corresponding conceptual scheme contains four dimensions.

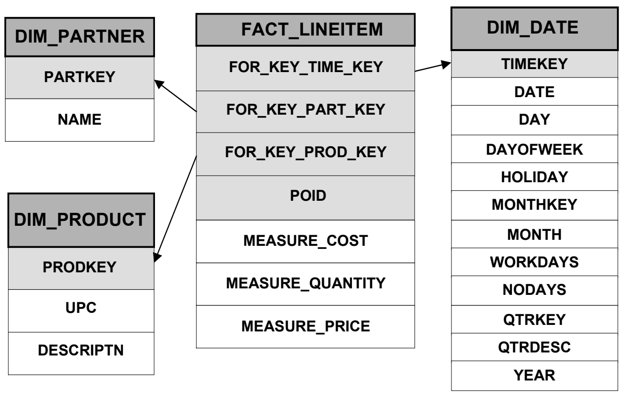


Figure 10. Logical scheme

Dimension keys of integer type with artificially generated values consume much less memory space than long character strings, making the fact table much smaller. This is the main reason why they are taken as keys. Besides, any dimension with an attribute whose value might change should have an artificially generated key. When the attribute value changes, a new record, with a new key value, is simply inserted. All dimension keys in the purchase order example are integers.

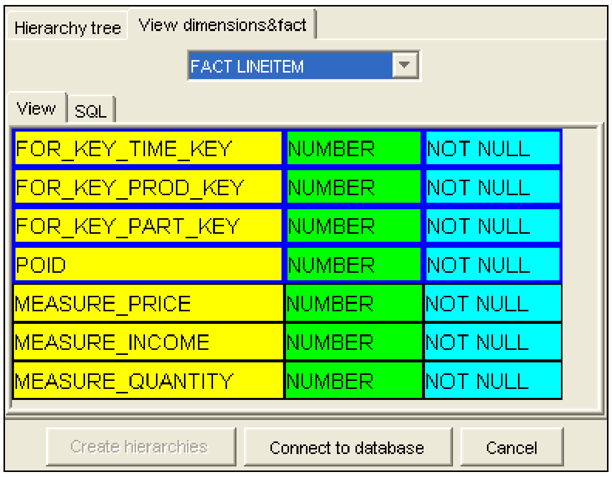
In addition to the simple rule that describes the creation of star schemas, some advanced design techniques may be needed for the correct translation of a fact scheme into a logical multidimensional scheme. Although unusual, sometimes it is useful to model many-to-many relationships. In this case a new table, called the *bridge table*, should be included into logical schema. The bridge table includes a *weight attribute*, whose values consist of the coefficients that represent weights of the many-to-many relationship for each couple of values participating in the relationship. In result, this technique gives consistent aggregations when many-to-many relationships are modeled.

After completing the conceptual design by defining dimensions and measures, the tool automatically produces the logical scheme based on the star schema. Tables are displayed in the GUI. SQL statements for creating the specified tables in a database are also produced.

The designer defines the database where the tables should be created. The tool connects to the database via JDBC (Java DataBase Connectivity) interface and executes the SQL statements. The temporary version of the tool can connect to an Oracle 9i or Oracle 10i database.

All dimensional tables and the fact can be seen in the GUI of the tool. The fact table for OAGIS 7.2.1 Purchase Order is shown in Fig. 11.

VII. Conclusion



FOR_KEY_TIME_KEY	NUMBER	NOT NULL
FOR_KEY_PROD_KEY	NUMBER	NOT NULL
FOR_KEY_PART_KEY	NUMBER	NOT NULL
POID	NUMBER	NOT NULL
MEASURE_PRICE	NUMBER	NOT NULL
MEASURE_INCOME	NUMBER	NOT NULL
MEASURE_QUANTITY	NUMBER	NOT NULL

Figure 11. Fact table as shown in the tool

This paper describes the accomplishment of data warehouse design and construction starting from the source XML Schemas and conforming XML documents. The final steps of conceptual design and the whole process of logical design are presented. The approach has been implemented in a prototype tool which helps the designer in designing faster and more accurately. All the phases of the conceptual and logical design are controlled and monitored by the designer through a graphical interface that allows some rearranging interventions too. In the last steps of the conceptual design, a semi-automatically created dependency graph is rearranged by the warehouse designer, and then dimensions and measures are defined. In the process of logical design, the derived conceptual schema is translated into a star schema. This step is performed automatically by the tool. When explaining the process of logical design, particular relevance is given to additivity of the measures and degenerate dimensions. An example of attributes that may either become measures or dimensional attributes is introduced and a solution of the problem is outlined. At the end of the design process, the prototype tool connects to a database and creates tables according to the proposed star schema.

References

[1] M. Golfarelli, D. Maio, and S. Rizzi, "Conceptual design of data warehouses from E/R schemes", *HICSS-31*, vol. VII, Kona, Hawaii, pp. 334-343, 1998.

[2] M. Golfarelli, D. Maio, S. Rizzi, "The dimensional fact model: a conceptual model for data warehouses", *International Journal of Cooperative Information Systems*, vol. 7, n. 2&3, pp. 215-247, 1998..

[3] D. Florescu and D. Kossman, "Storing and querying XML data using an RDBMS", *IEEE Data Engineering Bulletin*, vol. 22, n. 3, 1999

[4] M. Golfarelli, S. Rizzi, and B. Vrdoljak, "Data warehouse design from XML sources", *ACM International Workshop on Data Warehousing and OLAP (DOLAP01)*, Atlanta, pp. 40-47, 2001.

[5] M. Jensen, T. Møller, and T.B. Pedersen, "Specifying OLAP cubes on XML data", *Journal of Intelligent Information Systems*, 2001.

[6] M. Jensen, T. Møller, and T.B. Pedersen, "Converting XML data to UML diagrams for conceptual data integration", *International Workshop on Data Integration over the Web (DIWeb01)*, Interlaken, 2001.

[7] R. Kimball, *The Data Warehouse Toolkit*. John Wiley & Sons, 1996.

[8] J. Shanmugasundaram et al., "Relational databases for querying XML documents: Limitations and Opportunities", *25th VLDB Conference*, Edinburgh, 1999.

[9] B. Vrdoljak, M. Banek, S. Rizzi, "Automating conceptual design of web warehouses". Proceedings of the *International Conference on Telecommunications (ConTEL03)*, Zagreb, Croatia, 2003.

[10] B. Vrdoljak, M. Banek, S. Rizzi, "Designing web warehouses from XML schemas", *International Conference on Data Warehousing and Knowledge Discovery (DaWaK03)*, Prague, Czech Republic [Lecture notes in Computer Science (LNCS), Springer, 2003].

[11] B. Vrdoljak, M. Banek, Z. Skočir, " Methodology for integrating XML data into data warehouses", *International Convention MIPRO (MIPRO 2004)*, Opatija, Croatia, 2004.

[12] World Wide Web Consortium (W3C), "XML 1.0 Specification", 2004. http://www.w3.org/TR/2004/REC-xml-20040204

[13] World Wide Web Consortium (W3C), "XML Schema Part 1: Structures Second Edition", 2004.
http://www.w3.org/TR/2004/REC-xmlschema-1-20041028

[14] World Wide Web Consortium (W3C), "XML Schema Part 2: Datatypes Second Edition", 2004.
http://www.w3.org/TR/2004/REC-xmlschema-2-20041028

[15] World Wide Web Consortium (W3C), "XML Schema Part 0: Primer Second Edition", 2004.
http://www.w3.org/TR/2004/REC-xmlschema-0-20041028

[16] World Wide Web Consortium (W3C), "XQuery 1.0: An XML Query Language (Working Draft)", 2003
http://www.w3.org/TR/2003/WD-xquery-20031112

[17] Open Applications Group (OAG), "OAGIS Release 7.2.1", http://www.openapplications.org/downloads/oagidownloads.htm.