

BIwTL: A Business Information Warehouse Toolkit and Language for Warehousing Simplification and Automation

Bin He¹, Rui Wang², Ying Chen¹, Ana Lelescu¹, and James Rhodes¹
IBM Almaden Research Center¹, IBM China Research Laboratory²
binhe@us.ibm.edu, wangrcrl@cn.ibm.com, {yingchen, lelescu, jjrhodes}@us.ibm.com

ABSTRACT

Rapidly leveraging information analytics technologies to mine the mounting information in structured and unstructured forms, derive business insights and improve decision making is becoming increasingly critical to today's business successes. One of the key enablers of the analytics technologies is an Information Warehouse Management System (IWMS) that processes different types and forms of information, builds, and maintains the information warehouse (IW) effectively. Although traditional multi-dimensional data warehousing techniques, coupled with the well-known ETL processes (Extract, Transform, Load) may meet some of the requirements in an IWMS, in general, they fall short on several major aspects: 1. They often lack comprehensive support for both structured and unstructured data processing; 2. they are database-centric and require detailed database and warehouse knowledge to perform IWMS tasks, and hence they are tedious and time-consuming to operate and learn; 3. they are often inflexible and insufficient in coping with a wide variety of on-going IW maintenance tasks, such as adding new dimensions and handling regular and lengthy data updates with potential failures and errors.

To cope with such issues, this paper describes an IWMS, called BIwTL (Business Information Warehouse Toolkit and Language), that automates and simplifies IWMS tasks by devising a high-level declarative information warehousing language, GIWL, and building the runtime system components for such a language. BIwTL hides system details, *e.g.*, databases, full text indexers, and data warehouse models, from users by automatically generating appropriate runtime scripts and executing them based on the GIWL language specification. Moreover, BIwTL supports structured and unstructured information processing by embedding flexible data extraction and transformation capabilities, while ensuring high performance processing for large datasets. In addition, this paper systematically studied the core tasks around information warehousing and identified five key areas. In particular, we describe our technologies in three areas, *i.e.*, constructing an IW, data loading, and maintaining an IW. We have implemented such technologies in BIwTL 1.0 and validated it in real world environments with a number

of customers. Our experience suggests that BIwTL is light-weight, simple, efficient, and flexible.

Categories and Subject Descriptors

H.2.7 [Database Management]: Database Administration—*Data warehouse and repository*; H.2.8 [Database Management]: Database Applications—*Data mining*

General Terms

Design, Languages

Keywords

Information warehouse, warehousing language, data mining

1. INTRODUCTION

Today's businesses increasingly rely on vast amount of information. Data warehousing is an essential step towards exploring, understanding and analyzing business information for better and faster decision making. According to IDC, data warehousing market is expected to grow from 9.6 billion in 2005 to 13.5 billion in 2009 at a CAGR of 9 [24]. Although data warehousing techniques, such as multi-dimensional data warehouse models and ETL processing, have been widely studied by academia and practiced in industry, they are increasingly inadequate in their functionalities due to several major changes in today's information dynamics:

Comprehensive support for structured and unstructured data:

Existing data warehousing techniques are mainly designed to handle structured data. However, a large fraction of the enterprise data is in unstructured or semi-structured formats, *e.g.*, call center problem tickets, customer complaints. It is vital for an IWMS to support both types of data.

Simple and high-level warehouse operations:

The prominent and rapid adoption of the information analytics technologies such as text mining and Business Intelligence (BI) tools mandates for simple and efficient IWMS systems that can quickly process various information sources and build information warehouses. However, today's data warehouse systems often are hard to use and require detailed database and data warehouse knowledge. Even with skilled staff, building data warehouses often takes multi-person weeks and months to complete.

With the emergence of unstructured data, knowledge and skills about full-text indexers and search engines also become necessary. Such requirements are often impractical to assume in real world situations based on our customer experiences. For instance, organizations that want to use BI and text mining tools may not have strong database administrators (DBAs) for handling complex ETL processing. Clearly, a simpler and more efficient approach is needed.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD'07, June 11–14, 2007, Beijing, China.

Copyright 2007 ACM 978-1-59593-686-8/07/0006 ...\$5.00.

We believe that an IWMS that supports high-level operations for users to specify *what* they want to do, automatically figures out the details of *how-tos*, and handles all the operations accordingly could be a promising approach. Such an approach can significantly improve the system usability, robustness and performance and allows users without SQL and database expertise to perform IW tasks without significant training. This paper presents such an approach.

On-going IW maintenance operations: Today's ETL solutions mainly focus on only one aspect of the information warehouse management – building the IW. It is often assumed that once the warehouse is fully constructed and the data is completely loaded from the given set of data sources, the project is completed. That is, no subsequent modifications to the data warehouse is needed. Unfortunately, this assumption is no longer valid in today's information-driven age. In practice, information is constantly changing. It is common for users to frequently adjust the information warehouses by adding new dimensions to an existing IW or modifying and loading new data on a regular basis. All such operations must ensure the overall IW integrity and consistency.

Furthermore, due to the on-going data updating and loads, it becomes extremely important for the IWMS system to cope with various forms of system failures and potential error conditions. Today's warehousing systems have little or no support for fast failure recovery or error correction. Neither do they support on-going IW schema changes well. The common practice is to simply rebuild the data warehouses from scratch each time when the changes are required or perform lengthy and error-prone manual IW operations. Clearly, this is not acceptable.

This paper takes a holistic and systematic approach towards understanding the general IWMS requirements for BI and information analytics purposes and devising an IWMS system to fill the gaps identified above. In particular, the key contributions of this paper lies in the following:

- We systematically studied the information warehousing activities and identified five major information warehouse management activities which guides the design and architecture of a comprehensive IWMS system. Those five categories of activities include building IW, loading data into IW, maintaining IW, data cleansing, and querying.
- We presented a high level declarative language based approach to simplify and automate the ETL processing, and demonstrated how to use such language to drive three of the IW key activities.
- We presented a working light-weight IWMS system, BIwTL which leverages the language and its runtime components to address the core issues in three IWMS tasks: building IW, loading data, and maintaining IW. The BIwTL 1.0 system has been bundled with the IBM Business Insights Workbench [22] and deployed to a number of customers.

To our knowledge, this is the first-of-the-kind effort in a systematic study of the requirements, issues, and solutions for an IWMS system in today's information world. In the rest of the paper, we first compare our work with the related areas in Section 2. We then highlight the five types of IW activities and describe the key issues to be addressed in each of those areas in Section 3. We also present the high level language based approach in addressing the key issues using the GIWL language designed for BIwTL. Sections 4, 5, and 6 describe the GIWL language syntax and the system components for three aspects of the IWMS system respectively, *i.e.*, warehouse construction, data loading and warehouse data maintenance. The

overall system framework of BIwTL is presented in Section 7 and the real world case studies are presented in Section 8. Finally, we conclude our work in Section 9.

2. RELATED WORK

Our work is tightly related to the traditional data warehousing work. However, it differs from them in several aspects: First, traditional multi-dimensional data warehouse is mainly designed for OLAP applications on structured data alone. It also assumes that skilled DBAs are available for most of the data warehousing tasks. The main research effort has been on data warehouse schema model design [8, 27], OLAP operations [28, 6, 14, 15], OLAP indexing [13, 36], OLAP query optimization [11, 5, 10, 37], view selection [18, 12, 35, 7, 39], and view maintenance [40, 33, 31, 20, 41, 19]. Although there are some initial effort on developing a high-level data mining language for warehouse data, called DMQL [16], the major focus is on warehouse schema design and data mining operations for structured data.

BIwTL, however, is designed for both text mining as well as BI applications. It integrates both structured and unstructured data, and is much simpler, more efficient, and easy-to-use IW toolkit than those that exist today in the marketplace. For instance, existing ETL products, such as Kalido [25], Sunopsis [34], IBM DB2 Data Warehouse Edition [23], IBM WebSphere Information Integrator (formally Ascential) [21], Microsoft BI Accelerator [29] are often too complex to use or cannot handle structured and unstructured data in the same system. Also, no incremental data maintenance operations are supported in these products.

Moreover, to our knowledge, none of the existing research and development in data warehousing has taken a holistic view of all major warehousing activities or designed a comprehensive IWMS system that can cope with today's IW requirements. BIwTL is the first attempt in this direction while leveraging existing data warehousing, RDBMS, and full-text indexer technologies whenever appropriate.

In the development of BIwTL, we also observe many research opportunities. Some of them are existing research topics but may bring new challenging issues to solve. Others are novel research problems that have not been extensively studied in the literature. We believe that the development of an IWMS is not only useful for facilitating industrial development of warehousing products, but also for bringing many novel and challenging issues to the research community to study.

3. A BIG PICTURE

3.1 IWMS Core Activities

In an attempt to understand the requirements of the next generation IWMS system, we took a holistic look at the key activities around information warehousing. Our study based on the past few years of experience with customers suggests that today, the major information warehousing activities are centered around five areas (as shown in Figure 1):

Building warehouse: The first task involved in information warehousing is to construct an IW by designing the right data warehouse model, defining the appropriate schemas, *e.g.*, tables and dimensions that constitute the IW, and implementing the schema in the designated IW. Today, such tasks are time-consuming and complex. The next generation IWMS must allow warehouse administrators to build an IW in an easy, quick and flexible fashion without burdening them with the details of data warehouse models and database specific operations.

Loading data into warehouse: Once an IW is defined, the next step is typically to load data into the warehouse. Loading data is tightly related to the well-known ETL (*i.e.*, extract, transform, load) process in that proper data fields need to be extracted, transformed and loaded into the target IW. Due to the requirements originating from unstructured data, the new IWMS must support *flexible* data extraction for various sources with different formats, such as XML files, *sophisticated* data transformations, and *efficient* loading mechanisms for large amount of data. None of the traditional ETL tools can handle the level of flexibility, complexity, and efficiency imposed by semi-structured data and unstructured data today.

Maintaining warehouse data: Because of the potential change in information sources and the need for different types of analysis, an IWMS must allow flexible changes to IW while automatically maintaining the data consistency and integrity, *e.g.*, data and the indices in RDBMS and full-text indices. Furthermore, an IWMS must be able to cope with various failure and error situations without requiring users to rebuild IW all the time. Such areas have received very little research in the past.

Data cleansing: An IWMS should support actions to clean up data already loaded in the warehouse or before the data is loaded. Such cleansing includes cleansing for structured, semi-structured and unstructured data, *e.g.*, changing and merging dimension values, coping with invalid XML files, handling different character encodings, etc. Although data cleansing techniques exist in various contexts, they have not been systematically studied in an I environment.

Querying warehouse data: Once the IW is built, it will be used to serve various forms of queries. An IWMS must support efficient processing of IW queries, *e.g.*, OLAP operations, as well as text search and retrieval operations. Thus far, work has been done in isolation on supporting OLAP queries on structured information in an IW. The combination of structured and unstructured queries require some new innovation.

3.2 BIwTL– The next generation IWMS

As a step towards developing the next generation IWMS system that encompasses capabilities to facilitate all five core IW activities, we built a light-weight IWMS called BIwTL (Business Information warehouse Toolkit and Language). BIwTL closes the major gaps as identified in Section 1. In this paper, we describe the key solutions embedded in BIwTL to support the first three types of IW activities, *i.e.*, building warehouse, loading data and maintaining data. These three tasks are essential to enable the data cleansing and querying capabilities.

Underlying BIwTL is a declarative high level language based approach which is instrumental to simplify and automate the IW tasks. In particular, we designed a general and declarative IW language, GIWL, for all IW tasks. With GIWL, users only specify *what they want* rather than *how to execute* the operations. The analogy is that of SQL language [3] and its relationship to relational database management systems (RDBMS). With SQL, users only need to define what they want in the query result rather than how to execute the query. The runtime system handles the query automatically.

Similarly, GIWL language can be used to declaratively define various IW tasks. The runtime system, *i.e.*, BIwTL, provides inherent support to carry out operations automatically without human intervention. BIwTL manages both structured and unstructured data. BIwTL uses databases to store and index structured data, and file systems or databases for unstructured data. To support high-performance and complex keyword queries, BIwTL integrates with

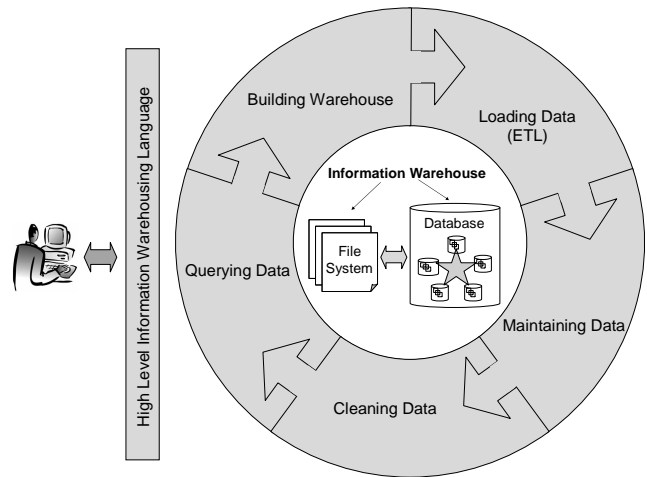


Figure 1: Information warehouse management systems.

full text indexers (*e.g.*, Juru [1], Lucence [2]) to automatically build indices for unstructured data. BIwTL maintains the data and index consistency between databases and file systems at all times.

4. IW CONSTRUCTION

To illustrate our GIWL based approach and how it is applied to all IW tasks, we focus on key IW activity areas one at a time. In this section, we discuss the issues we face in IW construction and our solutions using the GIWL language and runtime components.

One of the major issues with today’s IW tools is that they are often system-dependent (*e.g.*, DB2, Oracle, SQL Server, etc.). They require detailed RDBMS, IW modeling, and text indexing knowledge. Moreover, they often require highly skilled IW administrators to perform tedious, time-consuming, and error-prone customization when utilized in different customer environments. Such customizations are necessary mainly due to idiosyncrasy of different RDBMSs, file systems, and full-text indexers. Even though there are standards such as SQL [3] for RDBMS and POSIX for file systems, such standards often contain many non-standard variations, even for simple operations, which require customization at customer sites. The following examples in Example 1 illustrate some variations in different RDBMSs. Such customization tasks create barriers for timely IW construction and require significant training even with proficient DBAs.

Example 1: 1) Databases may use different keywords for defining the same data type. For instance, for the integer type, DB2 uses “INTEGER” and MySQL uses “INT”; for the LOB (Large Object) data type, DB2 uses “CLOB (size)” and MySQL uses “TINYBLOB”, “BLOB”, “MEDIUMBLOB”, and “LONGBLOB”.

2) Databases may have different ways to drop an index. For instance, in DB2, Oracle and PostgreSQL, the syntax is “DROP INDEX index_name”. SQL Server, however, uses “DROP INDEX table_name.index_name”. MySQL uses “ALTER TABLE table_name DROP INDEX index_name”.

3) Databases may have different ways to define automatically incremental attributes. For instance, in DB2, the syntax is “NOT NULL GENERATED ALWAYS AS IDENTITY”. In MySQL, the syntax is “NOT NULL AUTO_INCREMENT”. In SQL Server, the syntax is “NOT NULL IDENTITY”. Other databases also have their own syntax.

4) Since creating (and dropping) databases are not supported in JDBC and ODBC, to create (and drop) a database, one has to use database-specific command line utilities that are often different

from one database to another. For instance, DB2 has its command line utility “db2”, MySQL has “mysql”, and SQL Server has “sqlcmd”. They all have a very different set of options to execute. ■

To exacerbate the situation, full-text indexers are far less standardized than file systems and RDBMSs. Even building text indices may require users to link in very different libraries (e.g., Juru versus Lucene). In order to integrate structured and unstructured data in a single IW, IW designers must not only be a highly skilled DBA, a data warehouse modeling guru, but also a file system and full-text indexer expert. Clearly, this can create significant roadblocks for the adoption of IW, text mining, and BI technologies.

To alleviate such pains, BIwTL utilizes a high level declarative language GIWL to clearly separate the high-level IW design and low-level IW implementations. With BIwTL, IW designers design an IW by creating a language script that contains a set of declarative statements defined in GIWL. The language script indicates which database and full text indexer the user wishes to use and what tasks needs to be carried out. BIwTL automatically generates low-level database-specific and full-text indexer-specific commands at runtime based on the script and executes them without human intervention. Such an approach removes the customization burden from users and enables overall IW task simplification and automation. The following section describes the detailed language specification for IW construction.

4.1 Language Specification

The language specification for constructing an IW include two levels: IW level and dimension level. The IW level commands define the language syntax at the granularity of entire information warehouses, such as creating, using, and dropping an IW. The dimension level commands define the language syntax at the granularity of facts and dimensions, such as creating, altering, and dropping a dimension. The concepts of facts and dimensions are well known in data warehousing. We do not repeat in this paper (see [8, 27] for references). In the rest of section, we use the lower-case words to indicate the command attributes that require user inputs and upper-case words to designate the command names.

4.1.1 IW Level Commands

To create an IW, e.g., iw_name in a database, e.g., db_name, use:

```
CREATE INFORMATION WAREHOUSE iw_name
  IN DATABASE db_name;
```

A single db_name database may hold multiple information warehouses.

To list all information warehouses on a database server, use:

```
SHOW INFORMATION WAREHOUSES;
```

To delete a specific IW, e.g., db_name . iw_name, use:

```
DROP INFORMATION WAREHOUSE db_name.iw_name;
```

To use an existing IW, use:

```
USE INFORMATION WAREHOUSE db_name.iw_name;
```

4.1.2 Dimension Level Commands

To set the name of the fact table, use:

```
SET FACT_TABLE = fact_table_name;
```

To create a dimension, e.g., dimension_name, use:

```
CREATE DIMENSION dimension_name FIELDS(
  attribute_def, ..., attribute_def)
  MODEL USING [MEASURE | STAR | SNOWFLAKE
  | NEWFACT | FILE] ON [ FACT | dim_name];
```

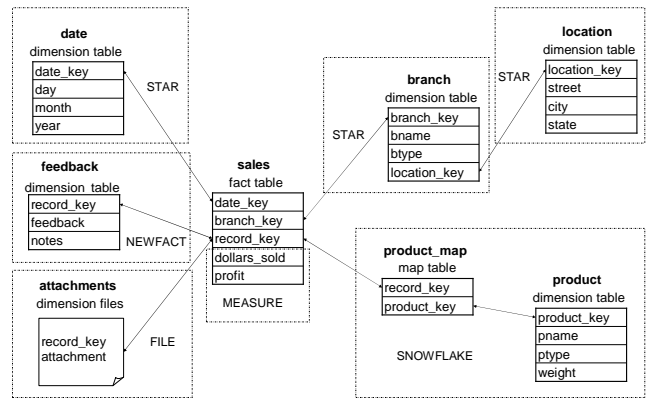


Figure 2: An IW Example sale in BIwTL.

```
attribute_def: attr_name TYPE [DATETIME
  | INT | BIGINT | FLOAT
  | VARCHAR (size) | CLOB (size) | ...]
  INDEX [NO | DB | TEXT]
```

Each dimension consists of a set of attributes. Each attribute is defined by the triplet: attribute name, e.g., attr_name, data type and index type. Data types include all standard types such as INT, BIGINT, FLOAT, DATETIME, VARCHAR, and CLOB. Index type can be an index in database or index in full text indexer, or no index. In addition, each dimension requires a schema model to be defined by the user. In our current implementation, the schema model can be one of the five: MEASURE, STAR, SNOWFLAKE, NEWFACT and FILE. A dimension can be associated with either the fact table or another dimension, which can be set using the “ON” keyword.

Since most of the above models are well-known concepts in the data warehousing world and have conventional meanings [17], we do not discuss them in details in this paper. However, in our work, in order to support integration of structured and unstructured data in one IWMS system, i.e., BIwTL, we slightly modified their meanings as explained below. Such modifications do not imply fundamental limitations of the approach. Instead they were designed to help simplify information warehousing tasks. Given the tradeoffs of different models and usage patterns, we believe that there is not a single model that would fit all situations. Hence, supporting different models provides users with flexibility while still hiding the low-level system-specific details during model construction.

1. MEASURE: A dimension with the MEASURE model means that the attributes in this dimension will be put in the fact table. No new tables are created for this type of dimension. MEASURE model is only meaningful when associating with the fact table. Hence, a dimension with the MEASURE model should always use “ON FACT” in the creation statement.
2. STAR: A dimension with the STAR model means that the primary key of this dimension is a foreign key of the fact table or another dimension, depending on the setting of the “ON” keyword. Therefore, a traditional snowflake model can be constructed by using a sequence of STAR models in BIwTL, as shown in an example later.
3. SNOWFLAKE: A dimension, D , with the SNOWFLAKE model means this dimension is linked to the fact table, F , or another dimension, D' through a map table, or named bridge table in [26]. The map table describes the $n:n$ relationship between D and F or D and D' .

```

CREATE INFORMATION WAREHOUSE sale
  IN DATABASE company;

USE INFORMATION WAREHOUSE company.sale;

SET FACT_TABLE = sales;

CREATE DIMENSION amount FIELDS (
  dollars_sold TYPE FLOAT INDEX DB,
  profit TYPE FLOAT INDEX DB)
  MODEL USING MEASURE ON FACT;

CREATE DIMENSION date FIELDS (
  day TYPE INT INDEX DB,
  month TYPE INT INDEX DB,
  year TYPE INT INDEX DB)
  MODEL USING STAR ON FACT;

CREATE DIMENSION branch FIELDS (
  bname TYPE VARCHAR (500) INDEX DB,
  btype TYPE VARCHAR (100) INDEX DB)
  MODEL USING STAR ON FACT;

CREATE DIMENSION location FIELDS (
  street TYPE VARCHAR (200) INDEX DB,
  city TYPE VARCHAR (50) INDEX DB,
  state TYPE VARCHAR (2) INDEX DB)
  MODEL USING STAR ON branch;

CREATE DIMENSION product FIELDS (
  pname TYPE VARCHAR (200) INDEX DB,
  ptype TYPE VARCHAR (50) INDEX DB,
  weight TYPE INT INDEX DB)
  MODEL USING STAR ON FACT;

CREATE DIMENSION feedback FIELDS (
  feedback TYPE CLOB (100K) INDEX TEXT,
  notes TYPE CLOB (100K) INDEX NO)
  MODEL USING STAR ON FACT;

CREATE DIMENSION attachments FIELDS (
  attachment TYPE CLOB (50M) INDEX TEXT)
  MODEL USING FILE ON FACT;

```

Figure 3: GIWL script for building the IW *sale*.

- NEWFACT: A dimension with the NEWFACT model means that the primary key of this dimension is also the primary key of the fact table or another dimension it links to. The NEWFACT model is needed for efficient incremental IW construction, such as adding new measure attributes or new attributes into existing dimensions after the IW has been constructed and data has been loaded. This model is used to accommodate the RDBMS limitation on adding new columns for already constructed tables.
- FILE: A dimension with the FILE model is the most unique model. It means that the dimension data are stored in file systems instead of databases. Such a model is designed to handle unstructured data. Although most databases also include text indexing and storage capabilities, our experience suggests that such support in RDBMSs is often premature and unscalable to large amount of unstructured data. On the other hand, file systems traditionally are more friendly to full-text indexers and are designed to store unstructured data. In the FILE model, unstructured files are directly associated with the primary key of the fact table.

Example 2: Figure 2 illustrates a sample IW *sale* built by BIwTL. It contains different dimensions using all five models described above. Figure 3 shows the GIWL language script used to generate the IW *sale*. With such a script, BIwTL automatically generates database table and index designs, including all database and indexer specific details.

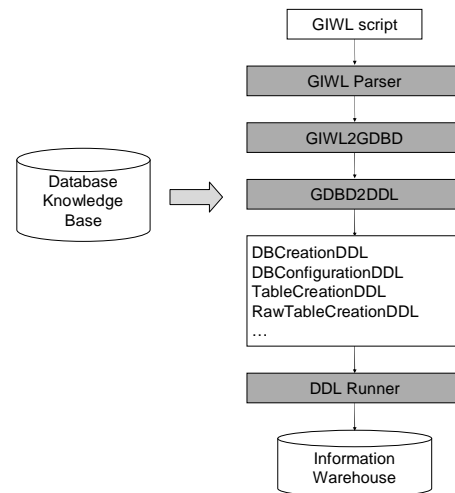


Figure 4: The system architecture for IW construction in BIwTL.

To construct an IW such as *sale*, the following steps are taken:

- one first uses IW level commands to indicate the intention of creation of the warehouse *sale* as shown in Figure 3.
- he/she sets the name of the fact table to be *sales* and creates a set of dimensions. In the *sale* example, user created a dimension *amount* with two measure attributes *dollars_sold* and *profit* in the fact table *sales*. Both attributes need indexes in databases. He/she also created a dimension *date* with the STAR model associated with *sales*.
- a dimension *branch* with the STAR model on *sales* is created.
- a dimension *location* with the STAR model on *branch* is created. Note that the *location* dimension is traditionally considered as a snowflake model in the IW design. But in BIwTL, it is viewed as a STAR model for its directly linked dimension.
- user also created a dimension *product* with the SNOWFLAKE model on *sales*. As a result, a table called *product_map* is generated to connect the *product* table and the *sales* table.
- a dimension *feedback* with the NEWFACT model on *sales* is also created. From Figure 3, we can see that, although the attribute *feedback* is stored in a database, its index can be built using a full text indexer.
- we create a dimension *attachments* with the FILE model which uses file systems to store the attachment texts. ■

To list all dimensions in an IW, use:

```
SHOW DIMENSIONS;
```

To delete a dimension in an IW, use:

```
DROP DIMENSION dimension_name;
```

4.2 System Architecture and Implementation

Given a GIWL script for building an IW, Figure 4 shows the high-level system architecture of BIwTL to process the script: The input script is first parsed by the GIWL Parser. The parser output goes through the GIWL2GDBD module which generates database independent designs for the tables and indices according to the script. GDBD2DDL then translates the general database designs into database-specific DDL statements for the user-selected target

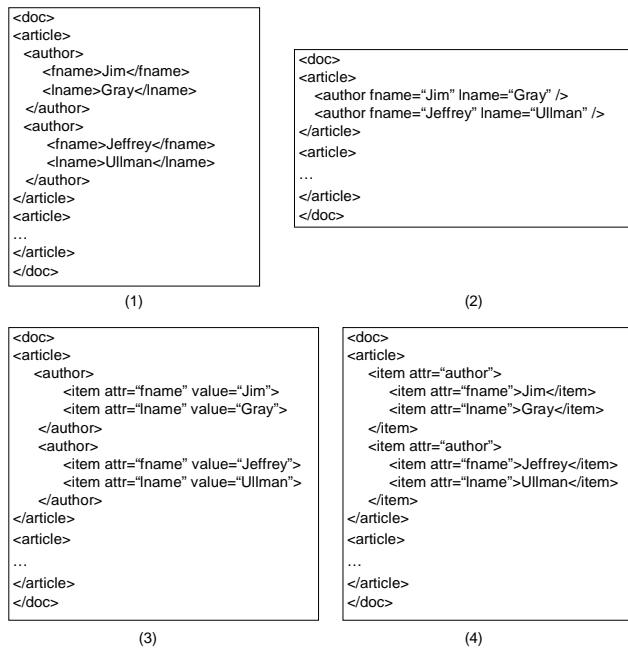


Figure 5: An example to show that the same data can be organized into different XML formats.

database by leveraging a database knowledge base. The knowledge base contains database-specific parameters and settings that must be incorporated at runtime so that the DDL Runner can simply execute the database-specific DDL scripts to construct the IW.

Clearly, such a framework is extensible and can support different databases while hiding the RDBMS details. Besides the fact and dimension tables, BIwTL also maintains several internal system tables to record the IW design. In particular, we record all the information about warehouses (*i.e.*, warehouse names, their storage databases, their fact tables, dimension tables, dimension models, dimension attributes, attribute types and index types) in the internal system tables. In the IW construction phase, file system storage and full text indexers are not involved. They will play a role in the data loading stage.

5. DATA LOADING

After the IW is constructed, data loading can begin. Data loading typically requires users to first specify how the source data is mapped to the target IW schema before the data is loaded according to the mappings. Source-to-target mapping entails two tasks: data extraction (*i.e.*, extracting pieces of data of interest from the source data) and data transformation (*i.e.*, transforming the extracted data into appropriate formats and/or values for the target IW). BIwTL currently is implemented to work with any XML-formatted sources. We selected XML because XML is common. It also represents a combination of structured and unstructured data. That is, it typically contains both structured fields as well as unstructured texts. It also has complex schema model. We believe if BIwTL handles XML well, extensions to other sources will be simple. Below, we discuss the requirements for data extraction, transformation, and loading respectively.

Data Extraction: Typically, to support analytics applications, only a subset of source data fields need to be populated into the target IW. This suggests that the data extraction mechanism must be flexible to allow accurate extraction of the desired fields from source data. For XML data, this is especially challenging due to its highly

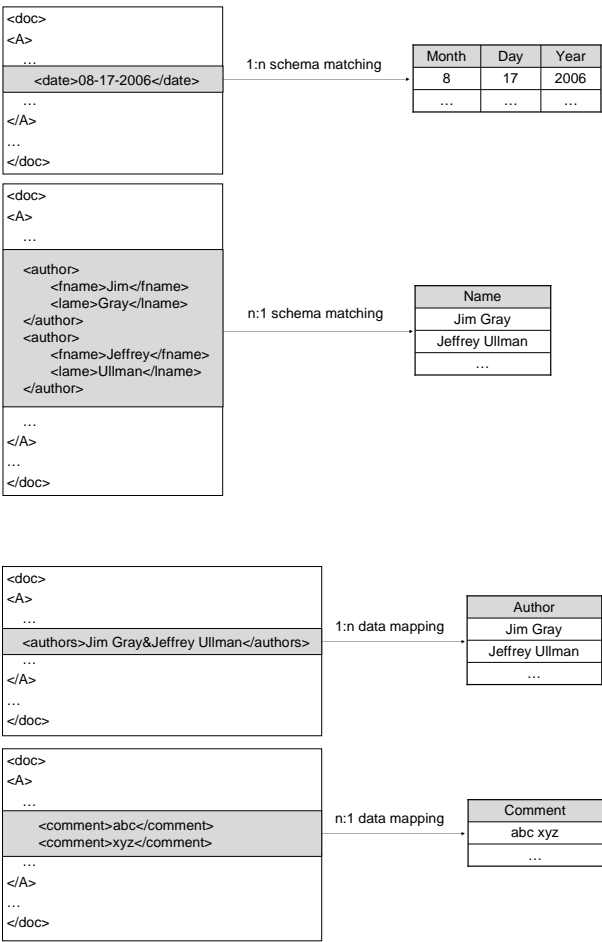


Figure 7: An example of complex data mapping.

flexible formats. For instance, the exactly same data content can be organized very differently in XML format as shown in Figure 5.

All four segments, although having very different DTDs (Document Type Definitions), contain the same data content (*i.e.*, the first and last names of two authors). Yet, some of them put data as text values (*i.e.*, segments 1 and 4) while others as attribute values (*i.e.*, segments 2 and 3). Some of them put last name and first name as tag names (*i.e.*, segment 1), while others as attribute names (*i.e.*, segment 2) or attribute values (*i.e.*, segments 3 and 4). An IW solution must be able to cope with such situations. We will show some real world examples in Section 8.

To resolve the issues above, BIwTL uses XPath [4] to locate the desired data fields. For the examples in Figure 5, the xpath to locate the article nodes is “/doc/article”. Assuming that users would like to retrieve fields for each article, then given an article node context, the xpath to locate all the authors in segments 1, 2, 3 is “./author”, while the xpath to locate all the authors in segment 4 is “./item[@attr=‘author’]”. Given an author node context, the xpaths to locate the value of the first name in these four segments are “./fname/text()”, “self::node()/@fname”, “./item[@attr=‘fname’]/@value”, and “./item[@attr=‘fname’]/text()” respectively.

Such record and attribute location capabilities can be modified and extended to support locating records and attributes in RDBMS or delimited files as well. We do not discuss these topics in this paper.

Data Transformation: The extracted data sometimes requires transformation before it can be inserted into the target IW, *e.g.*, trans-

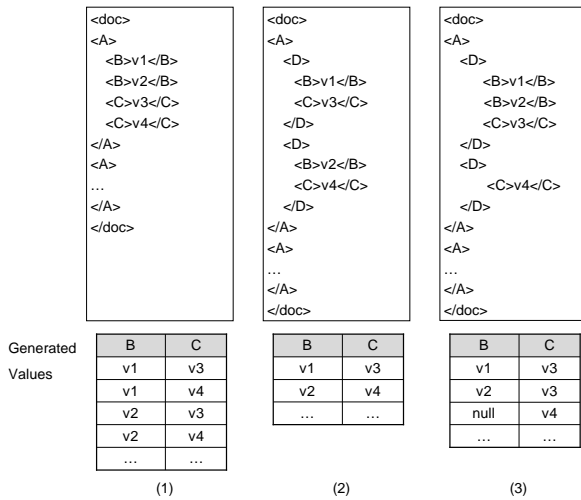


Figure 8: An example of data relationships.

forming a date value from “yyyy-mm-dd” format to “mm/dd/yyyy”, or selecting a substring from the data value. BIWTL supports such transformations by providing a common set of transformation functions as well as user defined functions.

In addition to *data format* transformations, an ETL solution should also provide two common, but more complex transformations, *i.e.*, *source-to-target complex schema matching* and *complex data value mapping* as shown in Figure 6 and Figure 7. Figure 6 shows an example of source-to-target complex schema matching. The first scenario is 1:n schema matching, which means one attribute is matched into multiple attributes in a dimension, *e.g.*, from date to month, day, and year. The second is n:1 schema matching, which matches multiple attributes into one attribute in a dimension, *e.g.*, combining last name and first name into a name attribute.

Figure 6 shows an example of the source-to-target complex data value mapping. The first scenario is a 1:n data value mapping, which splits the source data value into multiple data values in a dimension in the target IW, *e.g.*, from “Jim Gray&Jeffrey Ullman” to “Jim Gray” and “Jeffrey Ullman”. The second is a n:1 data value mapping, which combines multiple data values into one value in a target dimension.

Data Loading: Once desired data fields are extracted and transformed, data loading can begin. Data loading must ensure proper maintenance of the *data relationships*. That is, if two attributes A_1 and A_2 have certain dependencies in the source data, such dependencies must be maintained in the target IW as well. Similarly, if two attributes A_1 and A_2 are independent in the source data, they should be so in the target IW. Figure 8 shows different data relationships for three seemingly similar XML segments as examples. We assume that users would like to create a dimension with two columns, *B* and *C*, in all three cases. In segment 1, the values of tags *B* and *C* are independent. In segment 2, *B* and *C* are grouped together at all times. Segment 3 also has *B* and *C* grouped together, but within each group, their values are independent. Such groupings indicate that the data loading results for the dimension table that contains *B* and *C* will be different as shown in Figure 8.

5.1 Language Specification

To handle both structured and unstructured data in file systems and databases, we designed a set of high level GIWL commands for data loading tasks while hiding the extraction, transformation and loading details from users. The language specification for data loading contains two main categories: source mapping commands

and data loading commands. The source mapping commands define the language syntax for data extraction and transformation. The data loading commands are for data loading.

5.1.1 Source Mapping Commands

To create a mapping profile, *e.g.*, `profile_name`, use:

```
CREATE MATCHING PROFILE profile_name;
A mapping profile contains a set of mappings.
```

To set the path of a record in a profile, use:

```
SET RECORD_PATH = "xpath:a_xpath"
IN profile_name;
```

A record is a unit of entity that is to be inserted into the target IW, such as a document and all of its associated metadata fields. It corresponds to a record in the fact table of the IW. This command sets the xpath to the record nodes.

To add mappings for a dimension into a profile, use:

```
CREATE MATCHING IN profile_name
FOR DIMENSION dimension_name FIELDS(
mapping_def, ..., mapping_def);
```

```
mapping_def: attr_name FROM
attr_path, ..., attr_path
FUNCTION a_func
```

```
attr_path: "xpath:a_xpath"
```

This command creates all the mappings from the source data to the target attributes of a dimension. `attr_name` specifies the attribute name in the dimension. The source data fields are expressed using a set of xpaths. The data transformation is defined by the “FUNCTION” keyword. A function is defined in a nested way, *i.e.*, a function can contain other functions.

To list all the profiles, use:

```
SHOW MATCHING PROFILES;
```

To list all the mappings in a profile, use:

```
SHOW MATCHINGS IN profile_name;
```

To delete a profile, use:

```
DROP MATCHING PROFILE profile_name;
```

To remove mappings for a dimension from a profile, use:

```
DROP MATCHING FROM profile_name
FOR DIMENSION dimension_name;
```

5.1.2 Data Loading Commands

To load a source data file, use:

```
LOAD FILE file_name
USING PROFILE profile_name;
```

To load a directory of files, use:

```
LOAD DIR dir_name
USING PROFILE profile_name;
```

Example 3: Figure 9 illustrates a data loading example using an IW, *example*. Assuming that the XML data contains all the attributes used in Figures 6 and 7, we first create a mapping profile *mth* and set the xpath of all the records to be “/doc/A” which locates all document records to be loaded. We then create a set of mappings. There are three sample mappings to the *value* dimension: *value1* contains an xpath, *value2* contains a data transformation function *Substring*, and *value3* contains nested functions. The question mark in the function represents the values extracted from

```

USE DATA WAREHOUSE demo.example;

CREATE MATCHING PROFILE mth;

SET RECORD_PATH = "xpath:/doc/A" IN mth;

/* simple matchings */
CREATE MATCHING IN mth FOR DIMENSION value FIELDS (
  value1 FROM "xpath:/value1/text()",
  value2 FROM "xpath:/value2/text()" FUNCTION Substring(?, 3, 9),
  value3 FROM "xpath:/value3/text()" FUNCTION ToUpperCase(Substring(?, 5));

/* 1-to-n complex schema matching */
CREATE MATCHING IN mth FOR DIMENSION date FIELDS (
  month FROM "xpath:/date/text()" FUNCTION SplitSelect(?, "-", 1),
  day FROM "xpath:/date/text()" FUNCTION SplitSelect(?, "-", 2),
  year FROM "xpath:/date/text()" FUNCTION SplitSelect(?, "-", 3);

/* n-to-1 complex schema matching */
CREATE MATCHING IN mth FOR DIMENSION name FIELDS (
  name FROM "xpath:/author/fname/text()", "xpath:/author/lname/text()"
  FUNCTION Concat(?1, "?" ?2));

/* 1-to-n complex data matching */
CREATE MATCHING IN mth FOR DIMENSION author FIELDS (
  author FROM "xpath:/authors/text()" FUNCTION Split(?, "&");

/* n-to-1 complex data matching */
CREATE MATCHING IN mth FOR DIMENSION comment FIELDS (
  comment FROM "xpath:/comment/text()" FUNCTION Merge(?, " ");

```

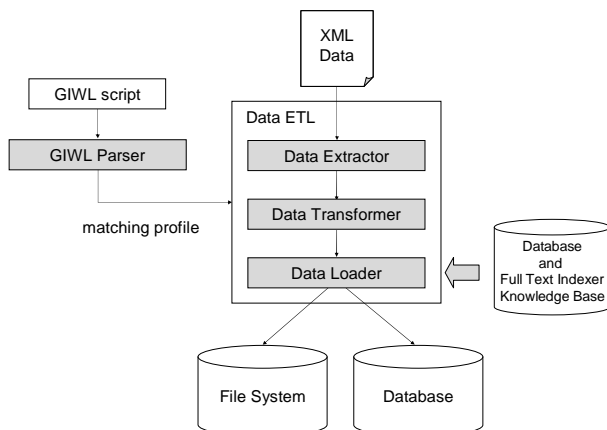


Figure 10: The architecture for data loading in BIwTL.

the xpath.

We also showed a dimension with 1:n schema mapping for the *date* dimension. In this example, a 1:n schema mapping is decomposed into *n* individual simple mappings, where each mapping utilizes the function *SplitSelect* to split and select the desired portions of data. The reverse, *i.e.*, a dimension with an *n:1* schema mapping, is illustrated using the *name* dimension, where multiple xpaths are used to specify the desired data, *i.e.*, the first name and last name and the function *Concat* concatenates them for the target field in the dimension. “?1” denotes the extracted data from the first xpath and “?2” the second. Similar to schema matching, we showed two data mapping examples, *i.e.*, 1:n data mapping for the *author* dimension and *n:1* data mapping for the *comment* dimension. They used functions *Split* and *Merge* to split the data value into multiple data values and merge multiple data values into one value respectively. Finally, the *LOAD* command loads the XML data file data.xml according to the mapping profile *mth*. ■

5.2 System Architecture and Implementation

Figure 10 shows the high-level system architecture of the key

data loading components in BIwTL. The input language script, such as the example above, is parsed by the GIWL Parser. The parsed output is fed into the Data ETL component. The Data Extractor module extracts the desired data using an XPath Parser for XML sources, for example. The Data Transformer module then transforms the extracted data into the target formats according to the mapping profile. Finally, the Data Loader module loads the data into appropriate target IW tables and file system locations. It also builds the appropriate indexes. Similar to the IW construction phase, the Data Loader first utilizes generic database and full text indexing APIs as much as possible and then customizes the commands into database and full text indexer-specific ones according to the knowledge base inputs.

BIwTL leveraged the similar approach to Clio [30, 38] to maintain the data relationships. Besides, our data loader is able to handle some more tricky real-world cases, as we will discuss in Section 8. We do not describe the technical details of the data loader in this paper. Its content deserves a separate technical paper.

To ensure fast data loading for large amount of data, BIwTL also embeds a set of performance enhancement techniques, such as stream based XPath parser, bulk loading into staging tables coupled with bulk population from the staging tables to the target IW tables. Such techniques are critical as today’s IW projects may be loading significant amount of data on a regular basis.

6. IW MAINTENANCE

Due to the fast growing nature of today’s information contents, IWMSs must provide sufficient IW maintenance capabilities which are typically missing from today’s ETL solutions. In this paper, we describe two key types of maintenance capabilities, *i.e.*, the ability to recover from data loading failures and errors efficiently. Without such capabilities, failures or errors occurred in a lengthy data load typically result in complete IW reloads. Even if DBAs would manually clean up the failed loads, there is often a danger of breaking the overall IW data integrity and consistency in such manual processes.

Failure Recovery describes a situation where a data loading may fail due to unexpected system errors, *e.g.*, machine or network crash, and an IWMS must properly clean up the incomplete records from the IW and be able to resume from where it was left off without breaking data integrity and consistency. To cope with such situations, two logical actions should take place: *Abort* and *Resume*. Similar to the transaction rollback concepts in databases, *abort* intends to roll the IW state back to a consistent state before the failure point. While *resume* allows users to resume the failed data loads from that last consistent state without reloading everything from scratch. No known RDBMS has readily available support for such failure recovery operations. Yet it extremely important for an IW environment. Note that abort and resume also is useful when users intentionally wish to abort a data load.

Error Correction describes a situation where the data may be loaded successfully but users may find errors in some loaded data content in the IW and hence wish to revert some portion of the data load and reload them using corrected contents. To allow flexible error corrections for the loaded data, BIwTL supports undo and redo operations. Such operations can also be used to facilitate data modifications.

Similar to other IW activities, BIwTL contains a set of high level GIWL commands to handle both failure recovery and error correction cases. The system internal will automatically recover from the failed and erroneous situations as described below.

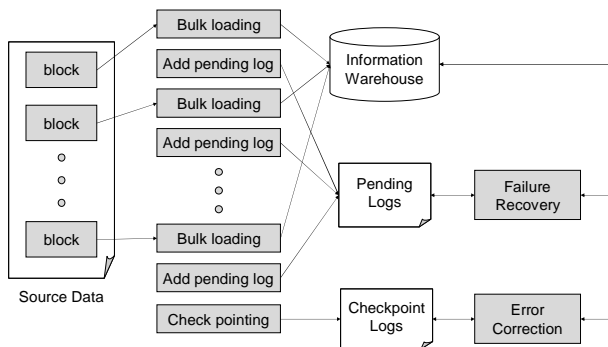


Figure 11: The system architecture for data maintenance in BIwTL.

6.1 Language Specification

6.1.1 Failure Recovery Commands

To abort a failed data loading of a source file, e.g., `src_name`, use:

```
ABORT LOADING src_name;
Here src_name is the name of source file being loaded.
```

To list all aborted loads, use:

```
SHOW PENDING LOADINGS;
Loading resumption is done implicitly when users issue the LOAD
command as shown in Section 5.1. BIwTL automatically resumes
the loading from the last consistent break point.
```

6.1.2 Error Correction Commands

To list all loaded source files in the loading order, use:

```
SHOW LOADED SOURCES;
```

To undo the load of a source file, e.g., `src_name`, use:

```
UNDO src_name;
This command will undo the loading of the source src_name.
```

To undo loadings to a particular source, use:

```
UNDO UNTIL src_name;
This command will undo all the loaded sources until the source
src_name. (The source src_name is also included in the undo
action.)
```

To undo all the loaded source files, use:

```
UNDO ALL;
This command will empty the IW.
```

To list all the sources can be redone, use:

```
SHOW REDOABLE SOURCES;
This command will list all the sources that have no changes after
undone and thus are redoable.
```

To redo loadings to a particular source file, use:

```
REDO src_name;
This command will redo the loading of the source src_name.
```

To redo loadings to a particular source, use:

```
REDO UNTIL src_name;
This command will redo loadings of all the sources until the
source src_name. (The source src_name is also included in the
redo action.)
```

6.2 System Architecture and Implementation

Figure 11 shows the high-level system architecture and key components in BIwTL for handling IW maintenance activities. To sup-

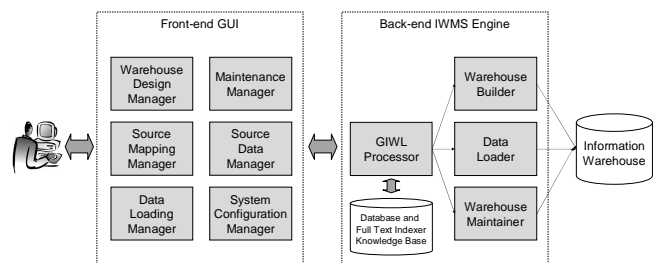


Figure 12: The overall system architecture of BIwTL.

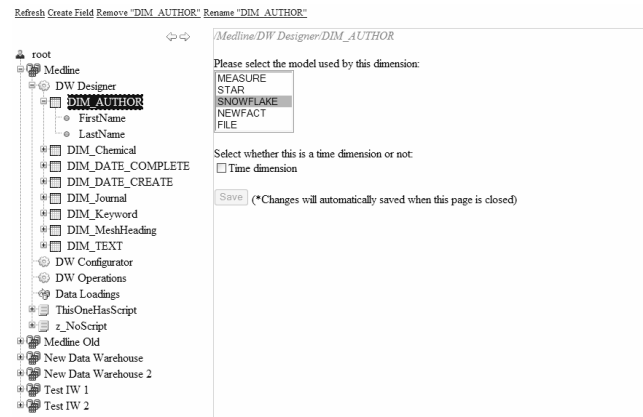


Figure 13: A snapshot for the front end GUI: Designing data warehouse dimension.

port proper IW maintenance, BIwTL utilizes a set of pending logs and checkpoint logs in BIwTL. Pending logs track the intermediate status in during data loading. They are used for failure recovery. Checkpoint logs record the IW status when data loads are completed. They are used for error correction.

For a given source file, BIwTL may split it up into blocks if the file is very large. This is because today's RDBMS often has limitations on how much data can be bulk loaded each time, and BIwTL utilizes the existing RDBMS bulk loading capability for fast data loads. Before the source file is fully loaded, its status is set as pending. BIwTL bulk-loads one block at a time. After loading each block, BIwTL records the number of tuples loaded in the pending logs. If the loading is interrupted, the failure recovery module uses the pending logs to abort or resume the loading.

After a source file is fully loaded, BIwTL records the a new checkpoint into the checkpoint logs. The error correction module will use the checkpoint logs to do undo and redo actions. To support undo and redo operations, we use a version scheme to track different versions of the data instead of physically deleting the records from the IW. The scheme is worthy of a separate technical paper, hence we do not discuss the details in this paper.

7. SYSTEM FRAMEWORK & USER INTERFACE

The overall BIwTL system architecture contains the front-end GUI and the back-end IWMS Engine as shown in Figure 12. The back-end IWMS Engine includes a command line based GIWL processor and its associated runtime modules, *i.e.*, the Warehouse Builder for constructing the IW, the Data Loader for loading data, and the Warehouse Maintainer for IW maintenance. We skip the details of those modules as we have discussed them previously. This section mainly focuses on the front-end GUI.

The front-end is a Web based graphic user interface (GUI) for the back-end IWMS Engine. It contains six modules: Warehouse De-

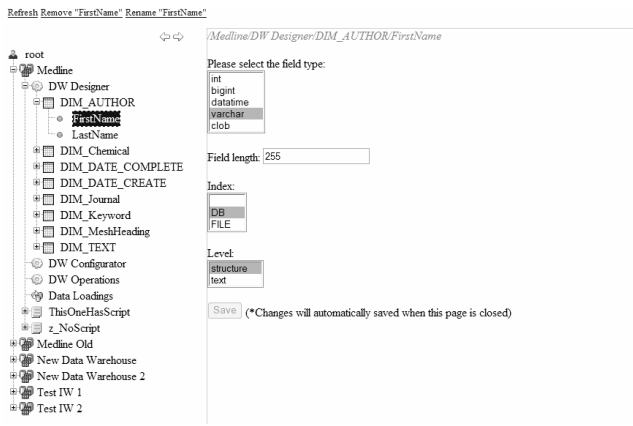


Figure 14: A snapshot for the front end GUI: Designing data warehouse attribute.

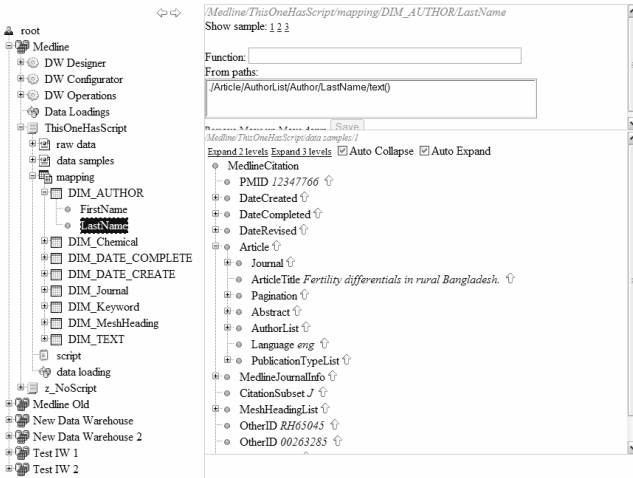


Figure 15: A snapshot for the front end GUI: Specifying source mapping.

sign Manager, Source Mapping Manager, Data Loading Manager, Maintenance Manager, Source Data Manager, and System Configuration Manager. The first four modules generate GIWL commands for the IW building, schema-mapping, data loading, and IW maintenance tasks. Such a front-end allows users to work with the BIWTL without even knowing the details of the GIWL language. A more intuitive GUI front-end will automatically generate the appropriate GIWL scripts. The generated commands are passed to the back-end IWMS Engine for execution. The Source Data Manager manages the source data. The System Configuration Manager manages and configures the IW settings.

Figure 13 shows a screen snapshot for the front-end GUI, which organizes warehouses in a tree structure the left panel shows. Under each IW, there are DW designer, configuration, a set of mapping profiles. Figure 13 shows the screen for the dimension design. When a user chooses the dimension *DIM_Author*, the right panel displays the schema model of this dimension. The top portion of the screen shows the list of operations available for this dimension, such as create an attribute and remove this dimension.

Figure 14 shows the screen snapshot of the interface for the attribute design. When a user chooses the attribute *FirstName* in the dimension *DIM_Author*, the right panel will display the attribute type, data length and index information. The top portion of the screen shows the available operations for this attribute such as remove and rename the attribute.

```
<Journal>
<ISSN IssnType="Print">0044-278X</ISSN>
<JournalIssue CitedMedium="Print">
  <volumn>90</volumn>
  <PubDate>
    <Year>1930</Year>
  </PubDate>
</JournalIssue>
<Title>Zeitschrift fur Geburtshilfe und Gynakologie.</Title>
...
</Journal>
```

(1)

```
<cdr:document>
  <cdr:item name="form">TCitation</cdr:item>
  <cdr:item name="daysincycle">5</cdr:item>
  <cdr:item name="approvedinrepid">
    <cdr:text>8025633890057D098</cdr:text>
  </cdr:item>
  <cdr:item name="yearmonthfirstapproved">
    <cdr:text>200304</cdr:text>
  </cdr:item>
  ...
</cdr:document>
```

(2)

Figure 16: Sample real data for different extraction cases.

Figure 15 shows the screen snapshot for specifying the mapping for the attribute *LastName* in dimension *DIM_Author* under one mapping profile. The right panel shows the input XML schema structure with some sample data. A user can choose the data of interests by browsing the structure and data. The front-end interface will derive corresponding xpaths automatically.

8. CASE STUDY

BIWTL is a working system and has been utilized to manage many real data sources in our customer engagements. It was validated against large data sets that contains tens of millions of records as well as data sets that have large fraction of unstructured data. Furthermore, many of the sources are complex and require a wide variety of data extraction and transformation capabilities. BIWTL is a critical component of an end-to-end information analytics solution our team has developed over the past few years, called Business Insights Workbench [22, 9, 32] (BIW). BIW integrates the structured and unstructured information analytics in one platform. In this section, we present a set of sample XML data from real world data sources and show how BIWTL copes with such cases. Because of the reason of data privacy, we may change the data contents for some sample cases, which will not affect the quality of illustration.

Data Extraction Complexity: Figure 16 illustrates a situation where different extraction cases need to be handled. Sample 1 is a common case, where each field has its own tag name. Sample 2, however, uses the same tag name *cdr:item* for different fields. Such fields can only be differentiated by the attribute names, e.g., *form*, *daysincycle*. Using XPath, BIWTL can easily handle such complicated cases.

Data Transformation Flexibility: BIWTL has flexible supports for various data transformations, Figure 17 shows several real world cases. In sample 1, project partners' names needs to be extracted, e.g., Tom Lee from CN=Tom Lee/OU=US/OU=PDF/O=WEM. Figure 18 shows, that by using the nested function *SplitSelect* (*SplitSelect*(?, "/", 1), "=", 2), we can easily extract the names. Sample 2 is another nested function case, where dates are extracted from the data, e.g., getting 20040901 from 20040901T113518,00+0530 and transforming it into day 1, month 9, and year 2004. Figure 18 shows the corresponding nested function we used for such transformation. For instance, to

```

<cdr:item name="projpartner">
  <cdr:textlist>
    <cdr:text>CN=Tom Lee/OU=US/OU=PRF/O=WEM</cdr:text>
    <cdr:text>CN=John Smith/OU=US/OU=PRF/O=WEM</cdr:text>
    <cdr:text>CN=James Anderson/OU=US/OU=PRF/O=WEM</cdr:text>
  </cdr:textlist>
</cdr:item>
(1)

<cdr:item name="dimodified">
  <cdr:date>20040901T113518,00+0530</cdr:date>
</cdr:item>
(2)

<cdr:item name="projmanager">
  <cdr:textlist>
    <cdr:text>CN=Tom Lee/OU=US/OU=PRF/O=WEM</cdr:text>
    <cdr:text>CN=Paul Wang/OU=US/OU=PRF/O=WEM</cdr:text>
    <cdr:text>CN=Alice King/OU=US/OU=PRF/O=WEM</cdr:text>
  </cdr:textlist>
</cdr:item>
<cdr:item name="docauthor">
  <cdr:text>CN=Paul Wang/OU=US/OU=PRF/O=WEM</cdr:text>
</cdr:item>
(3)

<cdr:item name="projdesc">
  <XHTML:p class="id8">
    <XHTML:span style="color:blue;font-size:8pt;line-eight:1em;
      font-family:Arial;">Scoping, timing, resources and activities by
      phase</XHTML:span>
  </XHTML:p >
  <XHTML:p class="id9">
    <XHTML:span style="color:blue;font-size:8pt;line-eight:1em;
      font-family:Arial;">E-business strategic assessment</XHTML:span>
  </XHTML:p >
  ...
</cdr:item>
(4)

```

Figure 17: Sample real data for different transformation cases

get year information, we use `DateConversion(SplitSelect(?, "T", 1), "yyyymmdd", "yyyy")`.

Sample 3 shows a case that combines two fields, `projmanager` and `docauthor`, into an `author` field. Figure 18 illustrates the language statements needed for such a case. It uses the `function Union` to combine multiple values and remove duplicate names. Sample 4 shows a case of merging multiple data values into *i.e.*, merging all the data values in `XHTML:span` into one text value. Figure 18 shows how we used the `Merge` function for a purpose.

Data Loading: Figure 19 illustrates several complex data loading cases. In sample case 1, users wish to combine two fields `DescriptorName` and `QualifierName` in the same dimension. Although they are grouped by the tag `MeshHeading`, within each group, the tag `QualifierName` can have zero to more than one occurrences. The correct data loading must handle the grouping of these two fields as well as the variation on the number of `QualifierName` within a group, as shown in Figure 20.

In sample case 2, users wish to group fields by an implicit ordering rather than an explicit XML tag. That is, each file record should contain the three adjacent fields `file_id`, `filename`, and `attachmentfile_text`. BIWTL can discover such implicit groups and generates correct loading result as shown in Figure 20. To our knowledge, no existing work and tools can automatically handle this implicit ordering based attribute grouping. The scheme is worthy of a separate technical paper, hence we do not discuss the details in this paper.

9. CONCLUSION

This paper presented a high level, declarative language, *i.e.*, GIWL, and its runtime system components, embedded in BIWTL for IW simplification and automation. The GIWL language specification provides high level abstractions for users to manage the IWs. It allows the decoupling of IW designs from lower level system oper-

```

/* matching for sample case 1 */
CREATE MATCHING IN mth FOR DIMENSION prj_partner FIELDS (
  name FROM "xpath:./bcs:item[@name='projpartner']/bcs:text/text()"
  FUNCTION SplitSelect(SplitSelect(?, "T", 1), "=", 2));

/* matching for sample case 2 */
CREATE MATCHING IN mth FOR DIMENSION modify_date FIELDS (
  year FROM "xpath:./bcs:item[@name='dimodified']/bcs:date/text()"
  FUNCTION DateConversion(SplitSelect(?, "T", 1), "yyyymmdd", "yyyy"),
  month FROM "xpath:./bcs:item[@name='dimodified']/bcs:date/text()"
  FUNCTION DateConversion(SplitSelect(?, "T", 1), "yyyymmdd", "mm"),
  day FROM "xpath:./bcs:item[@name='dimodified']/bcs:date/text()"
  FUNCTION DateConversion(SplitSelect(?, "T", 1), "yyyymmdd", "dd"));

/* matching for sample case 3 */
CREATE MATCHING IN mth FOR DIMENSION author FIELDS (
  name FROM "xpath:./bcs:item[@name='docauthor']/bcs:text/text()",
  "xpath:./bcs:item[@name='projmanager']/bcs:text/text()"
  FUNCTION Union(SplitSelect(SplitSelect(?, "T", 1), "=", 2),
    SplitSelect(SplitSelect(?, "T", 1), "=", 2)));

/* matching for sample case 4 */

<MeshHeadingList>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Diagnosis</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Tuberculosis</DescriptorName>
    <QualifierName MajorTopicYN="Y">diagnosis</QualifierName>
    <QualifierName MajorTopicYN="Y">epidemiology</QualifierName>
  </MeshHeading>
  ...
</MeshHeadingList>
(1)

<cdr:files>
  <cdr:file_id>ENE19_1_1</cdr:file_id>
  <cdr:filename>gdais-executive-summary-1-28-02-final-v9.ppt</cdr:filename>
  <cdr:attachmentfile_text>(large unstructured text)</cdr:attachmentfile_text>
  <cdr:file_id>ENE4_1_1</cdr:file_id>
  <cdr:filename>dwed-and-leveraging-citation---solution-based.ppt</cdr:filename>
  <cdr:attachmentfile_text>(large unstructured text)</cdr:attachmentfile_text>
  ...
</cdr:files>
(2)

```

Figure 19: Sample real world data for different loading cases.

ations, such as database operations, data mapping semantics, data loading semantics, and full-text indexer operations. With GIWL, users only need to specify what tasks they want to do. System internals will automatically carry out and complete the tasks. In our experience, two-hour training to normal users who have no RDBMS backgrounds is sufficient for carrying out most of the IW tasks. With a graphical front-end GUI to generate the GIWL scripts, it is even simpler for users to learn and use such IWMSs.

Such language abstraction also allows easy integration of unstructured data such as integration of full-text indexers and file systems while maintaining the integrity and consistency at all times. This is true even in the face of failure and error conditions. With the checkpointing and versioning capabilities, BIWTL allows fast and consistent failure recovery and error corrections.

To conclude, this paper discussed on our initial effort in developing BIWTL, with a particular focus on issues in warehouse construction, data loading and IW maintenance. We propose a declarative information warehousing language GIWL to decouple high-level user operations from lower-level system execution details. We have implemented BIWTL 1.0 and evaluated it over a large number of real world data sources. Our experience shows that BIWTL is light weight, automatic and easy to use.

DescriptorName	QualifierName
Diagnosis	null
Tuberculosis	diagnosis
Tuberculosis	epidemiology
...	...

(1)

file_id	filename	attachmentfile_text
ENE19_1_1	gdais-executive-summary-1-28-02-final-v9.ppt	(large text)
ENE4_1_1	dewd-and-leveraging-citation---solution-based.ppt	(large text)
...

(2)

Figure 20: Loading results for sample real world data.

10. REFERENCES

- [1] Juru. <http://www.haifa.ibm.com/km/ir/juru/index.html>.
- [2] Lucene. <http://lucene.apache.org>.
- [3] SQL. <http://en.wikipedia.org/wiki/SQL>.
- [4] XPath. <http://www.w3.org/TR/xpath>.
- [5] S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *VLDB Conference*, pages 506–521, 1996.
- [6] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. In *ICDE Conference*, pages 232–243, 1997.
- [7] E. Baralis, S. Paraboschi, and E. Teniente. Materialized views selection in a multidimensional database. In *VLDB Conference*, pages 156–165, 1997.
- [8] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *SIGMOD Record*, 26(1):65–74, 1997.
- [9] W. F. Cody, J. T. Kreulen, V. Krishna, and W. S. Spangler. The integration of business intelligence and knowledge management. In *IBM Systems Journal*, 2002.
- [10] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing iceberg queries efficiently. In *VLDB*, pages 299–310 Conference, 1998.
- [11] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data warehousing environments. In *VLDB Conference*, pages 358–369, 1995.
- [12] H. Gupta. Selection of views to materialize in a data warehouse. In *ICDT Conference*, pages 98–112, 1997.
- [13] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index selection for OLAP. In *ICDE Conference*, pages 208–219, 1997.
- [14] M. Gyssens and L. V. S. Lakshmanan. A foundation for multi-dimensional databases. In *VLDB Conference*, pages 106–115, 1997.
- [15] M. S. Hacid, P. Marcel, and C. Rigotti. A rule based data manipulation language for OLAP systems. In *Proc. of the 5th Intl. Conf. on Deductive and Object-Oriented Databases*, 1997.
- [16] J. Han, Y. Fu, W. Wang, K. Kopferski, and O. Zaiane. DMQL: A data mining query language for relational databases. In *SIGMOD'96 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'96)*, 1996.
- [17] J. Han and M. Kamber. *Data Mining: Concept and Techniques*. Morgan Kaufmann, 2000.
- [18] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. pages 205–216, 1996.
- [19] C. A. Hurtado, A. O. Mendelzon, and A. A. Vaisman. Maintaining data cubes under dimension updates. In *ICDE Conference*, pages 346–355, 1999.
- [20] N. Huyn. Multiple-view self-maintenance in data warehousing environments. In *VLDB Conference*, pages 26–35, 1997.
- [21] IBM. Ascential. <http://ibm.ascential.com>.
- [22] IBM. Business Insights Workbench. <http://www.almaden.ibm.com/asr/projects/biw>.
- [23] IBM. DB2 Data Warehouse Edition. <http://www-306.ibm.com/software/data/db2/dwe>.
- [24] IDC. Worldwide data warehousing tools 2005-2009 forecast. 2005.
- [25] Kalido. Enterprise Data Warehousing. <http://www.kalido.com>.
- [26] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. Wiley Publishing, Inc, 2002.
- [27] W. Lehner, J. Albrecht, and H. Wedekind. Normal forms for multidimensional databases. In *Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, 1998.
- [28] L. Libkin, R. Machlin, and L. Wong. A query language for multidimensional arrays: design, implementation, and optimization techniques. pages 228–239, 1996.
- [29] Microsoft. BI Accelerator. <http://www.microsoft.com/sql/prodinfo/previousversions/ssabi/default.aspx>.
- [30] R. J. Miller, L. M. Haas, and M. A. Hernández. Schema mapping as query discovery. In *VLDB Conference*, 2000.
- [31] D. Quass, A. Gupta, I. S. Mumick, and J. Widom. Making views self-maintainable for data warehousing. In *Proceedings of the 5th International Conference on Parallel and Distributed Information Systems*, pages 158–169, 1996.
- [32] W. S. Spangler and J. T. Kreulen. Machines in the conversation: Detecting themes and trends in informal communication streams. In *IBM Systems Journal*, 2006.
- [33] M. Staudt and M. Jarke. Incremental maintenance of externally materialized views. In *VLDB Conference*, pages 75–86, 1996.
- [34] Sunopsis. Data Conductor. <http://www.sunopsis.com>.
- [35] D. Theodoratos and T. K. Sellis. Data warehouse configuration. In *VLDB Conference*, pages 126–135, 1997.
- [36] M.-C. Wu and A. P. Buchmann. Encoded bitmap indexing for data warehouses. In *ICDE Conference*, pages 220–230, 1998.
- [37] D. Xin, J. Han, X. Li, and B. W. Wah. Star-cubing: Computing iceberg cubes by top-down and bottom-up integration. In *VLDB Conference*, pages 476–487, 2003.
- [38] L. L. Yan, R. J. Miller, L. M. Haas, and R. Fagin. Data-driven understanding and refinement of schema mappings. In *SIGMOD Conference*, 2001.
- [39] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *VLDB Conference*, pages 136–145, 1997.
- [40] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View maintenance in a warehousing environment. In *SIGMOD Conference*, pages 316–327, 1995.
- [41] Y. Zhuge, H. Garcia-Molina, and J. L. Wiener. Multiple view consistency for data warehousing. In *ICDE Conference*, pages 289–300, 1997.