IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4859

Supporting Object and Relational Data in a Data Warehouse

Ramesh Suribhatla, Les Miller
Dept. of Computer Science
Iowa State University
Ames, Iowa 50011
lmiller@iastate.edu

M. Mehdi Owrang O.
Dept. of Computer Science, Audio Technology, and Physics
American University
4400 Massachusetts Avenue, N.W., Washington, D.C. 20016
owrang@american.edu

ABSTRACT

Data warehouses with proven capabilities to deal with the massive amounts of information have already won the acceptance of many large organizations. This acceptance now is having users look for support of more complex data rather than just supporting data from legacy file systems and relational databases. While the incorporation of the object database technology has been one choice, the Object/Relational Database Management Systems (ORDBMSs), a blend of the Relational and Object technologies aimed at reaping the best of the both, is the preferred option. This paper presents a model for an ORDBMS based data warehouse. The model provides an infrastructure for the user to work with data represented in the form of classes, analyze it using tools, and also query the database at an abstract level. The model has been tested with a prototype using Oracle 8 in the backend and Java to build the analysis tools and user interface.

1. INTRODUCTION

Data warehouses provide the database arranged for ease and speed of search and retrieval of the data, in a way especially useful for business analysis purposes [1-6]. Such a database/databank can be accessed and analyzed using several tools like query engines, data mining algorithms, information visualization tools, report generators and the like [4]. The end-users are readily supplied with all the required business information on their desktops so that more responsive, faster and better decisions than ever before are taken.

To provide fast, efficient and reliable access to huge quantities of data for the millions of business units we often find the Relational DBMS (RDBMS) in use. However, in many application domains, like Computer Aided Design and Modeling (CAD/CAM), multimedia repositories, and document management, complex data types must be handled. As the amount of data grows, the many features offered by a DBMS for data management — for example, reduced application development time, concurrency control and recovery, indexing support, and query capabilities — become increasingly attractive, and ultimately, necessary. In addition, for developing data warehouse and On-Line Analytical Processing (OLAP) systems, the dominant relational database reaches its limitation. For example, the conventional star schema model of data warehouse has some limitations due to the nature of the relational model. First, this model cannot represent the semantics and operations of multi-dimensional data adequately. It is very difficult to address the problems of view design efficiently because of the hidden semantics. Second, in defining the higher level of summary data which require multiple complex aggregations, SQL queries do not portray the intuition needed to facilitate building and supporting efficient execution of complex queries on complex data.

While the incorporation of the object database technology has been one choice[4], the Object/Relational Database Management Systems (ORDBMSs), a blend of the Relational and Object technologies aimed at reaping the best of the both, is the preferred option. It enables the users to define additional kinds of data — specifying the structure of the data and also the ways of operating on it — and use these types within the relational model [7]. It also facilitates the storage of the structured business data in its natural form and the applications for its retrieval too. By use of the object-oriented programming techniques ORDBMSs can also work efficiently with the applications so developed. Their support for user-defined datatypes further makes it easier for application developers to work with complex data like images, audio, video etc.

In developing complex systems such as data warehouses, the ORDBMS is becoming very attractive. Such integration is referred to as Object Relational Data Warehouses (ORDW). The semantics of data and queries can be explicitly captured, represented, and utilized based on objects, leading to more efficiencies as well as capabilities. There has been several researches related to developing and manipulating ORDW [8-11]. In [8], authors propose their ORDW architecture with new metadata layer and describe the design and implementation of a new kind of metadata to bridge the gap between the object-oriented environment and the relational database. Various metadata classes are defined and their roles in the O-R data warehouse are discussed.

In [9], authors present a methodology for efficient query processing in an ORDW environment by defining and incorporating the Associated Horizontal Class Partitioning (AHCP) techniques over the ORDW schema. The proposed scheme starts with a given set of data warehouse queries and creates a near-optimal AHCP scheme for the queries. Then, the proposed scheme selects the AHCP fragments as materialized views to facilitate efficient evaluation of these queries.

OLAP queries in data warehousing systems are essentially complex queries involving multiple dimensions and their specialization. In [10], authors discuss the need to incorporate additional semantics and provide the ORDW environment to explicitly capture, represent, and utilize the query and data semantics based on is-a and class composition hierarchies. They define a query-driven indexing approach based on Structural Join Index Hierarchy (SJIH) mechanism, specifically by using a hill-climbing heuristic algorithm, for efficient OLAP query processing.

In [11], authors propose the Object-Relational View design for the data warehouse. Using the object-oriented approach, it is possible to explicitly represent the semantics and reuse view (class) definitions based on the is-a hierarchy and the class composition hierarchies, resulting in a more efficient view mechanism.

Current literature lack discussion of the design of a class-based data warehouse model that provides an infrastructure for the user to work with data represented in the form of classes, analyze it using tools (i.e., data mining, statistical analysis tools), and query the database at abstract level. This paper presents a class-based model for an ORDBMS based data warehouse. It supports objects, by providing multi-dimensional tuples with facility for their embedment with other objects as well. As such, the user herein needs only basic knowledge of Java to create classes for the objects. A graphical user interface is also developed to facilitate interaction with the data warehouse, such that a user can list, add, delete and update the classes and the analysis tools also. These tools can be used on the classes in the databases to extract information, in addition to the queries. The model has been tested with a prototype using Oracle 8 in the backend and Java to build the analysis tools and the user interface.

The next section briefly looks at the issues in data warehousing. Section 3 looks at the design criteria and Section 4 looks at the system model. Section 5 looks at the prototype. Finally, we present our conclusions in Section 6.

2. DATA WAREHOUSE

According to W.H. Inmon (known as the father of data warehousing), a data warehouse [1-6] is a subject-oriented, integrated, time-variant, nonvolatile collection of data that is used primarily in organizational decision-making — where

- **Subject-oriented** means that all relevant data about a subject is gathered and stored as a single set in a useful format.
- **Integrated** refers to data being stored in a globally accepted fashion.
- **Time-variant** data represents long-term data.
- **Non-volatile** means the data warehouse is read-only.

3. DESIGN CRITERIA AND MODEL

Our data warehouse model has been designed to provide an infrastructure for a user to work with the data represented in the form of classes, analyze it using the tools and also query the database. The design and implementation of such a model should meet the following criteria:

Data and Classes

- The user views the data in the form of objects of a class and interacts with them.
- The classes support nested classes. And when a class is updated, all the other classes nesting that class would also be updated.
- The precision of the data is maintained, while storing, retrieving and updating the data from the data warehouse.
- The methods in a class can be called at the runtime. This allows methods, which are particular to a class, to be executed on the objects.
- The system contains kernel features for adding, deleting and updating classes.
- It also lists the classes and describes them by specifying its fields, constructors and methods.

Tools

- Tools are created, to analyze the data by running them on the data obtained by querying a class or a join of classes — with or without condition statements.
- The system kernel must provide features to add and delete a tool from the data warehouse.

Query

- An interface needs to be provided to build a query by selecting the fields from one or more classes — and by specifying the condition statements, if needed.

User-friendliness

- A user-friendly GUI should be provided to make the data warehouse usable.

In the next subsection we look at a conceptual view of our prototype.

3.1 A Conceptual View of the Data Warehouse Model

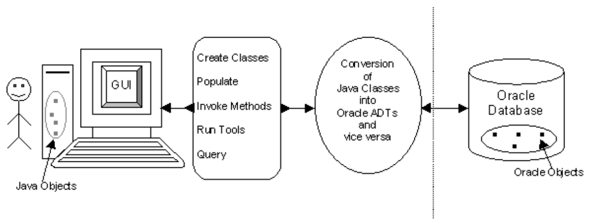
The block diagram in Figure 1 showcases the various features and components of this environment.

3.2 Meeting the Design Criteria

Data and Classes

- Using the fields specified in the Java source file, an ADT is created in the Oracle schema representing the Java class, where the fields in the Java class are represented by their corresponding attributes in the ADT.
- Apart from the Oracle datatypes, the ADTs can be built using other ADTs that are already created in the schema, thus supporting the nested objects feature.

Figure 1. Conceptual view of the model.



- When a method in a class is called at the runtime on an object, the method is executed by instantiating an object with the data retrieved from the database at the runtime by querying and then calling the method on that object.

Tools

The analysis tools are executed by passing the query results, obtained by executing the query, to a constructor of the tool. The source code of the tool has a set of overloaded constructors required to suit the various parameter lists.

Query

Similarly, the queries created by the user are sent to the database, executed and the results obtained are directly sent on to the GUI.

The next section overviews the current prototype.

4. PROTOTYPE

This section briefly overviews the implementation of the various kernel features of the data warehouse developed as a Java application that runs on a stand-alone system. It is built using the classes that add the kernel features of 'Classes', 'Tools', 'Query' and 'Help'.

4.1 Classes

Classes are one of the most important features in this application, as the user is to interact with the data in the form of Java classes. However, these classes are transformed into Oracle ADTs and stored in the Oracle database.

A utility Java class called 'Classes_Info.java' is implemented to interact with the Classes_Info_Table, to get and set the information needed for the kernel tools. The methods in this class make use of JDBC to connect to the Oracle database and get the list of classes existing; add or remove a class to the list of classes existing; get the list of classes that nest a given class; add or remove a class to the list of classes that nest a given class; and get the list classes that are populated.

4.2.1 List the Classes

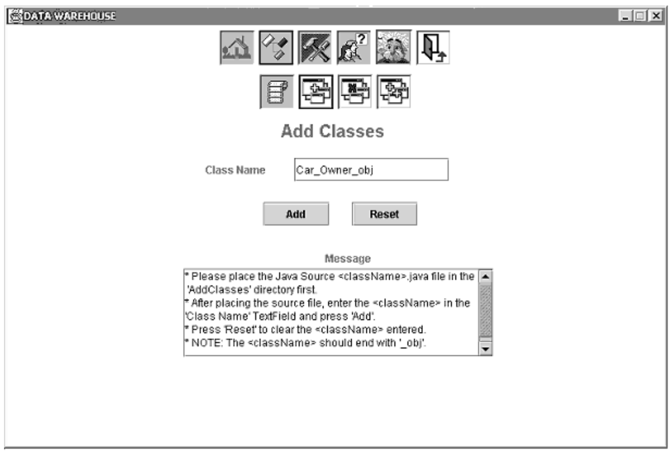
This kernel feature lists the classes available in the data warehouse. The list of classes is obtained from the 'Classes_Info_Table' and displayed in the form of a list in the GUI. When a user selects a class, the description of the class is displayed.

4.2.2 Add a Class

Implemented in the 'AddClassPanel.java' file, this kernel feature is useful for adding a class to the data warehouse. It is implemented in the following manner:

- The user places the Java source file in the 'AddClasses' directory and specifies the name of the class that is being added in the text field as shown in the Figure 2.
- Then it is compiled to make sure that there are no syntax errors. If a class nests other user-defined classes, we make sure that these nested classes are already in the warehouse before it is added.
- After meeting the necessary requirements, a class file is generated and placed in the home directory of the warehouse. Also, a source file is copied as a back up.

Figure 2. Adding a class to the data warehouse.



Class Name: Car_Owner_obj

```
* Please place the Java Source <className>.java file in the
 'AddClasses' directory first.
* After placing the source file, enter the <className> in the
'Class Name' TextField and press 'Add'.
* Press 'Reset' to clear the <className> entered.
* NOTE: The <className> should end with '_obj'.
```

- An Oracle ADT is created using the fields in the source file. A typemap is used to convert the datatype of a field in Java to the datatype of an attribute in Oracle.
- During the creation of this ADT, it is checked if this class uses any other existing classes. This list also helps to prevent deleting a class when classes that depend on this class exist in the schema.

4.2.3 Populate a Class

After creating a class, to populate a class <className>, a batch file containing Java application program called 'Populate' is executed from the command prompt, by typing 'populate className'. This creates a table in the schema with 'className_table' as its name and populates it by reading the data from the 'className.txt' file. The table is created only with one attribute, the ADT that is representing the className.

The prototype kernel also provides support for deleting and updating existing classes.

4.3 Tools

The tools are run on the classes to analyze them by extracting the data from the Oracle database by querying. Generic tools like Average_tool, Count_tool and Walk_tool have been implemented in the system as examples.

A utility Java class called 'ToolsInfo.java' is implemented to interact with the Tools_Info_Table. The methods in this class use JDBC to connect to the Oracle database to get the list of tools existing, and to add and remove a tool from the list of existing tools.

4.3.1 Running the Tools

This feature included in the query panel is used to run the tools on the classes and analyze the data. The user selects the needed class, the fields and a tool to be run on it. An SQL query, on a class or join of classes with or without condition statements, is generated at the runtime and executed to get the data from the database.

The results of a query are obtained in the form of a collection/array of Java datatypes by calling JDBC methods on the result set and converting the results from Oracle back to Java format.

The analysis tools consist of overloaded constructors accepting array(s) of primitive datatype or String as input parameters. The corresponding constructor is instantiated with the results in the form of an array obtained by querying as explained above.

4.4 Querying

Querying is the core and most important feature of the data warehouse developed in this paper. It combines the conventional query interface with the analysis tools and method invocation features making it possible for the user to interact with the system more efficiently, providing a user-friendly interface. The interface consists of two panels 'Query Form' and 'Query Results'. Using the Query Form, a user can build an SQL query by selecting the required class(es), the fields in the class and by specifying the conditions, which can be combined with AND' / 'OR'. The Query Results panel is used to display the results. Further the results obtained by querying can be analyzed either by running the analysis tools on them or by executing a method on the objects of a class or a combining the two.

In the Query Form, the 'List Of Populated Classes' consists only of the classes that are populated. The fields in the class are represented in the form of a tree in the 'List Of Fields' allowing the user to select the fields in the class and also the fields in the nested classes. The user can select multiple fields from a class by clicking on the fields with the control key 'Ctrl' pressed on the keyboard or can select all the fields in the class by selecting the Class name, which is the root of the tree. The user can also deselect a field by clicking on it with the control key pressed. The list of fields selected is shown in the 'Query' box, prefixed by 'SELECT', a keyword in this implementation indicating the fields selected (having the same terminology as in the standard SQL).

Similarly, the user can select multiple classes by using the control key from the list of classes, to form a query involving a join of two or more classes. The classes selected are displayed in the 'Query' box prefixed by 'FROM'.

Conditional statements can be added to the query by typing in the conditions. These logical expressions can be combined by using keywords like 'AND' and 'OR'. The conditions added are prefixed by the keyword 'WHERE'. The query can be made more sophisticated by using the keywords in Oracle like 'LIKE', 'NOT IN', 'NULL', and 'NOT NULL', but for this user needs to have knowledge of Oracle. Apart from these, though there is no text field for adding other clauses, options like 'ORDER BY' can be added to the query in the 'Query' box. Also, a user familiar with SQL can type the query in the Query box, in the format that is being used in this project and following the keywords used, and execute it.

The user, after forming the required query, can click on the 'Run' button and execute the query or click on the 'Reset' button to start the querying from the beginning. The query formation procedure can be depicted as shown in Figure 3.

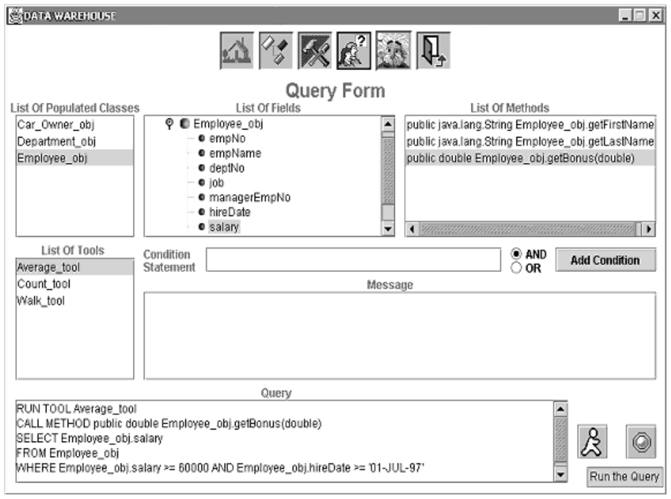
Before executing the query, it is parsed to change the required parts of it into a format that Oracle can understand. After parsing, JDBC is used in getting connected to the Oracle database and execute the queries.

The data warehouse allows the user to look at the query results or store them in a file.

4.4.1 Querying with Analysis Tools and Method Calls

A user can run an analysis tool on a query by selecting it from the tools available in the data warehouse (and only one tool can be selected

Figure 3. Query interface to query, run tools and call methods



List Of Populated Classes	List Of Fields	List Of Methods
Car_Owner_obj	Employee_obj	public java.lang.String Employee_obj.getFirstName
Department_obj	empNo	public java.lang.String Employee_obj.getLastName
Employee_obj	empName	public double Employee_obj.getBonus(double)
	deptNo	
	job	
	managerEmpNo	
	hireDate	
	salary	

List Of Tools: Average_tool, Count_tool, Walk_tool

```
RUN TOOL Average_tool
CALL METHOD public double Employee_obj.getBonus(double)
SELECT Employee_obj.salary
FROM Employee_obj
WHERE Employee_obj.salary >= 60000 AND Employee_obj.hireDate >= '01-JUL-97'
```

at a time). The tool runs on the query results and returns the analysis results.

Similarly, a method can be selected from the list of methods implemented in a class and invoked on the query results by passing the results as input arguments. Using these results, objects of the class are instantiated and the method is executed on them. Also, the analysis tools and the method calls can be used together, i.e. by first calling the method and then by running the tool on the values returned by the method call.

5. CONCLUSION

In this project, a data warehouse model has been developed and implemented using Oracle, version 8, supporting abstract datatypes. A successful attempt has been made to meet the criteria stated in this paper. It involves an integrated implementation of certain features like creating a Java perspective for the objects, which are actually stored in Oracle as ADTs; the nesting of objects (one into another); calling the methods of a class on its objects; running data analysis tools on the classes and querying. A user-friendly GUI is also provided which facilitates the use of the kernel features and work with the objects and the tools.

REFERENCES

1. Sun in Data Warehousing, Business Intelligence white papers, January 1997 http://www.sun.com/software/solutions/third-party/dw/whitepapers/med-WP.html

2. What is a Data Warehouse? W H Inmon http://www.cait.wustl.edu/cait/papers/prism/vol1_no1

3. Data Warehouse, Tatiana Goes Mendonca, December, 1997 http://www.ecst.csuchico.edu/~tatianam/csci374/dataware.html

4. A Data Warehouse Based on Materializing Object-Oriented Views, L.L. Miller, Yeping Zhou, Ying Lu, Sree Nilakanta, A.R. Hurson, submitted for publication.

5. An Introduction to Data Warehousing, Vivek R. Gupta, System Services, August 1997. http://www.system-services.com/dwintro.asp

6. Data Warehousing: An Overview, Gabrielle Gagnon http://www.zdnet.com/pcmag/pctech/content/18/05/ec1805.001.html

7. Oracle 8 Server Concepts http://nash.baruch.cuny.edu/oracle/server803/A54643_01/toc.htm

8. Metadata for Object-Relational Data Warehouse, Thanh N. Huynh, Oscar Mangisengi, and A. Min Tjoa, Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'2000), Stockholm, Sweden, 2000, PP. 3.1-3.9.

9. Efficient Query Processing with Associated Horizontal Class Partitioning in an Object Relational Data Warehousing Environment, Vivekanand Gopalkrishnan, Quing Li, and Kamalakar Karlapalem, Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'2000), Stockholm, Sweden, 2000, PP. 4.1-4.9.

10. Efficient Query Processing with Structural Join Indexing in an Object Relational Data Warehousing Environment, Proceedings of Information Resource Management Association International Conference (IRMA '00), Anchorage, Alaska, 2000, PP. 976-979.

11. Star/Snow-Flake Schema Driven Object Relational Data Warehouse Design and Query Processing Strategies, Vivekanand Gopalkrishnan, Quing Li, and Kamalakar Karlapalem, Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery (DaWak), LNCS 1676, Florence, Italy, 1999, PP. 11-22.