Communications of the IIMA

Volume 3 | Issue 2 Article 6

2003

Handling Unstructured Data Type in DB2 and Oracle

Alexander P. Pons
Computer Information Systems, University of Miami

Hassan Aljifri
Computer Information Systems, University of Miami

Handling Unstructured Data Type in DB2 and Oracle

Alexander P. Pons

Computer Information Systems, University of Miami, 421 Jenkins Building, Coral Gables, FL 33146
Phone: 305-284-1960, Fax: 305-284-5161, apons@miami.edu

Hassan Aljifri

Computer Information Systems, University of Miami, 421 Jenkins Building, Coral Gables, FL 33146
Phone: 305-284-4767, Fax: 305-284-5161, hassan@miami.edu

ABSTRACT

The objective of our work is to determine which mainstream object-relational database management systems (ORDMS) provide convenient facilities for the storage and manipulation of unstructured data objects. These objects, which consist of video, audio, photographs, and even executable code such as Java applets, are becoming readily employed by desktop, network, and Internet applications. Typically, these ORDMSs must store the objects in a manner by which they can be easily accessed, but more importantly, easily processed during either storage or retrieval. Our focus is on two of the ORDMS market leaders: IBM's DB2 and Oracle. The salient facilities of DB2 and Oracle in handling object types are analyzed, considering their advantages and disadvantages.

INTRODUCTION

As competition increases and companies are driven to construct their applications in not only a timely fashion, but also at a cost-effective price, companies need to build their applications to closely match both their business models and their business processes. These conditions, in turn, require that the information systems and applications that support business processes are flexible and customizable, while keeping costs reduced for the design, development, deployment, and maintenance of business applications. The tremendous use of the World Wide Web has increased the complexity and richness of data managed by the traditional business applications, making a key requirement of a database is to understand popular Web data types, including those supporting multimedia content. A database supporting these data types should be able to manage Web-page content, to create dynamic Web pages "on the fly," and to track user access to company Web sites (Shah, 2002).

In addition to scalar data, IT applications must be able to integrate documents, images, and sound and video clips. With the ability to handle all of these data types, a business would be able to not only store and manage all the data about their customers, including photographs and any other unstructured data in a single database, but also to leverage their data to make production more efficient and to increase their market share. Over time, hardware has become not only more powerful but also less expensive to purchase. As a result, databases are being forced to manipulate data in new and complicated ways, and in many cases, share this complex processing across multiple nodes on a computer network. Due to a changing marketplace, many application design and development teams have learned that developing applications based on object technology from the ground up can best satisfy their business needs and meet the challenges posed by the market forces.

Current relational database systems of a large-scale enterprise-computing environment are both scalable and robust, which is essential to a multi-user, data-rich environment. Additionally, the database must be able to handle and store complex, structured data as well as large, unstructured, domain-specific data, which include text, image, audio, and video types. Not only should the database be able to store and manage these data types, but also should be able to provide query capabilities for the data. Users of database technology make huge investments in building relational applications, and need their database to satisfy all their business requirements in such a manner that existing relational applications, schemas, and data can co-exist with new object-based schemas, data, and applications. The database system also should allow new object-oriented applications to run on existing relational schemas and data in such a way that users do not feel the “downtime” of the database or suffer a lag in performance. Most current database technologies are able to meet enterprise requirements for scalability, replication, and data distribution, which are invaluable technology features for business. The data server should not be compromised in terms of scalability, performance, and data replication and distribution, while supporting unstructured objects in the database.

To compare Oracle (Kumar, 1997), (Ballantyne, 2002), (Oracle, 2002) to IBM’s DB2 (Zeidenstein, 2001a and 2001b) we have chosen to focus on the following areas:

- User defined types (UDTs)
- Processing and manipulation of large object types (LOB) and their extensions
- Creating, inserting, and updating objects
- Inheritance
- Polymorphism

WHY OBJECT-RELATIONAL DATABASES?

Not only are most companies using these databases successfully, they are also constantly pushing database technology. The application of these technologies, however, has not been well-served by relational database management systems (RDBMS) because of its inherit inability to understand and process complex data types. The RDBMS is good only at handling traditional business data in the form of simple, alphanumeric data —

customer name, invoice number, part number, price, quantity, student name, grade, course, description, etc. This is one of the primary reasons why a large portion of most business data is not captured in any formal database anywhere, relational or other. Data that is not captured in any way include countless documents, forms, images, photographs, email messages, reports, and audio and video clips.

In order to effectively build new applications and enhance/extend existing applications we need to manage all types of data – both structured and unstructured– within a database (Bray, 1997). The complex data types, video and image and such, are often referred to as "rich data types" or "objects" because each type is associated with its own set of attributes and behaviors. Take for example, an image file. An image file has attributes including "file format" and "size." The users of such a file my require search through a collection of images for those that meet specific criteria, and/or perform special functions on the image such as display, rotate, crop, and scale. Not only will they want to process and manipulate the image file, but they also will want to combine those image searches with those on traditional data. As business needs and the technology that supports these needs change, business applications will require the database to be extensible, so that it can learn to understand new content, functions, and search methods, which will enable it to accommodate unanticipated application requirements. Such is the case with the need to store and manipulate multimedia Web pages, formatted in hypertext markup language (HTML) in a DBMS.

Unfortunately in the past, users were forced to find other solutions outside of their RDBMS to manage complex data (Kulkarni, 1993) and (Sullivan, 2002):

- Sometimes users would choose to store the complex data in undifferentiated BLOBs (Binary Large Objects), which could be gigabytes in size, in the database, and then implement the logic for those BLOBs in a client application. To create and manipulate the data, the user would be forced to use their customized client application again.

- Another way users manipulated complex data was to introduce a separate and what they called "specialized" system. Such a system would be a document- or image-management system, which was only capable of effectively handling one type of data.

- Finally, users would resort to utilizing another database manager, such as an object DBMS, for complex-data applications.

Although users have had to assimilate to database technology that is available, they have done a rather good job of coming up with "on-the-fly" solutions; however, each of these solutions has one or more drawbacks for the RDBMS user: (1) it may limit the sharing and reuse of business logic, (2) it may require the use of an API other than SQL, (3) it may be unable to integrate with the rest of the organization's critical data in the database, or (4) it may necessitate the management of multiple, heterogeneous databases.

Over the past several years, the inception of object-relational DBMSs has highlighted many weaknesses of RDBMS implementations. The RDBMS provides poor support for applications involving "rich data types" (i.e., video and audio data). However, the RDBMS has shown significant advantages of large-scale functionality, in addition to the availability of access and manipulation procedures for heterogeneous data, the possibility of being able to distribute data among multiple sites while preserving the integrity of the data, and finally providing integrated tools for managing the database environment. Many of today's customers want one database system to successfully meet multiple needs: for example, access to both traditional and complex data, extensibility, and adequate performance across many types of applications. The creation of the object relational database is focused directly on this type of customer.

Currently, a key feature that allows the RDBMS to manage complex data intelligently is under development for complex data in the form of text, spatial data, images, video and audio clips, time series data, and ultimately, any user-defined data type needed to meet the ever-changing and unique business requirements. There are several key forces that are pushing this development along:

- Organizations need the RDBMS to meet a broad range of application needs. Many companies involved in investments and applications want the RDBMS platform extended in such a far-reaching way in order to leverage those investments while enabling the DBMS to support new applications. This extended platform is now called "object-relational" DBMS. The goal of the company is not only to build on the positive aspects of the RDBMS and its query language, SQL (Structured Query Language), but also to provide integrated access to all the data all across the entire enterprise, as the company enriches the database with application-specific semantics.

- As mentioned before, it is advantageous for a RDBMS to understand Web data types and multimedia data types. An RDBMS that can do that will be a good candidate to manage Web-page content, to create dynamic pages "on the fly," and to track user access to specific sites.

- Finally, the benefit of object modeling techniques has created a desire for their support in the database. These techniques include encapsulation, inheritance, polymorphism, and others that push the benefits of object-oriented application development into the database server itself.

REQUIRED DATA TYPES

Extending RDBMS by enabling new data types that can even encapsulate complex internal structures and attributes is only one area that has been extended. Once the new data types are achieved, methods are needed to allow the manipulation and sorting of their data. As a result, additional extensions are needed for the comprehensive management of complex data:

- User-defined Functions (UDFs) allow the developer to define various methods in order for the applications to be able to create, manipulate, and access the data stored in the new data types. This extension is valuable because of the ability to operate on new data types without having to understand their internal structures.

- User-defined indexing structures are designed to efficiently retrieve the contents of structured and unstructured data such as text, images, video, etc. For manipulation of these types of data, traditional RDBMS indexing methods fall short of the requirement and usually are not appropriate.

- In order for a user to have the most efficient way to execute a query, an extensible optimizer which enables the database to assess the value and cost of user-defined functions and index structures is needed.

USER DEFINED DATA TYPES (UDT)

User–defined data types (UDTs) allow users or developers to define new, custom data types. UDTs can apply to either a column or a row in a table. The most basic UDT is a renamed or distinct type that extends an existing base data type for a column within the DBMS. The next UDT type is the abstract data types (ADTs) that bring together arbitrarily complex internal structures and attributes; examples include spatial data types, video data types, financial data types, and audio data types.

Row UDTs, or row types, are used to define an entire row or a set of nested columns in a table. They allow the DBMS to represent an “entity” such as an employee, a department, a customer, or a project in a single UDT. Row types are valuable to apply functions to rows and to establish relationships among entities within the database by means of references among corresponding rows. Reference types define relationships between row types. Reference types have two benefits: (1) the user can write simpler path expressions in queries instead of specifying complex join definitions, and (2) regardless of how the query is written, the optimizer can navigate by “walking” the pointers created by references as an alternative to using the value-based join access method of traditional RDBMS.

Other important aspects of UDT support include the following, and apply to both column and row types:

- The ability to support aggregate data types, lists, and arrays for defining collections of other types. One special collection type is a nested table, which allows a single column in one table to contain another table of multiple rows.

- The ability to define hierarchies of types, or objects with support for subtypes and inheritance.

- The ability to support the replication of data stored in UDTs.

User-defined data types are available in both Oracle and IBM's DB2 to make the database better understand the purpose of the data that was stored in it. Users can define distinct types, based on existing types, but which are "distinct" from all other types defined in the system by their name and semantics. This is a crucial step toward defining data in the business worlds and bringing together business logic in the database.

Since these types are created upon already existing types, it means that they are tightly integrated into the database management system for performance. The types share the same code, which allows for more efficient processing and control by database management systems.

USER DEFINED FUNCTIONS (UDF)

User-defined functions (UDFs or "user-defined methods" are the second focus of object-relational extensibility. UDFs can apply to both base data types and UDTs, but are particularly important for defining the methods by which applications create, manipulate, and access data stored as UDTs. Functions provide encapsulation so that applications do not depend on the internal representation of a data type in order to manipulate data. To evaluate UDF support, the following criteria must be examined:

- Types of UDFs supported — scalar UDFs return a single scalar value, column UDFs examine the values in an entire column and return an aggregate (i.e., a sum or average), and table UDFs return data in the form of a table of values. Table UDFs enable the DBMS to easily manipulate the returned data and to join it with other data.

- Flexible options for executing UDFs — An important issue is whether UDFs execute in the same address space as the database server (for better performance), or in a separate address space (for better security, given that UDFs are user-written code). The DBMS should offer both options and require special authorization to create a UDF that runs as part of the database server. Other options include specifying whether a UDF can be executed in parallel and where a UDF can run on the client, on the server, or on the remote server as a distributed function.

- Support for polymorphism — Polymorphism supports the use of the same name for different functions, which the DBMS will chose to invoke based on the arguments passed with the function call. Function overloading simplifies application development. The ability to resolve functions based on multiple attributes is important to maintain the consistency of subtypes.

CLOBS, BLOBS, LOBS

IBM

In 1995, before any other major database vendor could hit the market, IBM’s DB2 delivered the first object-relational extensions. These extensions were necessary to satisfy many applications’ needs to move beyond what a “normal” relational database could handle. Although the majority of transactional data could be mapped to the usual data types, the arrival of the “rich text” imaging systems and text management systems meant that data values were getting larger. Video and audio files are even larger. Using the existing technology, it has been nearly impossible to handle this type of data.

To address this pressing issue of the application requirement, DB2 Universal Database and Trade (UDB), in its Version 2 release, was able to store very large amounts of data in a single data value using their new data types. This alleviated applications from the burden of piecing together character strings to form a document. As an alternative, the entire image or document could be stored in one of the new large object types: character large object (CLOB), binary large object (BLOB), and double-byte character large object (DBCLOB). By having built in functions for manipulating the LOBs and adding subsequent technology to handle LOB data in client applications as files or through locators, it was easy to handle LOBs in any application. Operations on LOBs are highly optimized by deferring LOB materialization as much as possible.

Oracle

The latest release of Oracle provides loud support for defining and manipulating LOBs. It is able to extend the SQL Data Definition Language (DDL) and Data Manipulation Language (DML) commands to create and update LOB columns in a table or LOB attributes of an object type. Also included are the Oracle Call Interface (OCI) and PL/SQL package Application Programming Interface (API) to support random, piece-by-piece operations on LOBs. In order to support both internal and external data representations, the new release offers four extra data types to accommodate LOBs.

There are three SQL data types for defining instances of internal LOBs; these LOBs are similar to those of the DB2 Version 2:

- BLOB – a LOB whose value is composed of unstructured binary “raw” data.
- CLOB – a LOB whose value is composed of single-byte fixed-width character data that corresponds to the database character set defined for the Oracle database.
- NCLOB – a LOB whose value is composed of fixed-width, multi-byte character data that corresponds to the natural character set defined for the Oracle database.

Additionally, there is one external LOB data type: BFILE, a LOB whose value is composed solely of binary "raw" data, and is stored completely outside the database tablespace in a server-side operating system file. However, this method is not something new, as mentioned before, where users had to improvise ways of handling large data types in a similar way.

INHERITANCE

Object practitioners agree that any language that claims to be object-oriented must support inheritance. Inheritance allows for hierarchies in which each child has characteristics of its parent. Each level in the type hierarchy has properties, which can be shared by those beneath it; lower levels can have their own specialized attributes or functions as well.

From an object-programming standpoint, Oracle 8.0 is at a disadvantage in that it does not directly support inheritance. A user-defined object type cannot be created as a subtype of another. As a corollary, you cannot reuse method definitions. However, ANSI and ISO committees are working on object-oriented extensions to SQL and forming a position on how inheritance should behave in object relational databases.

Although inheritance gets much attention in object technologies, it is one of several types of relationships that can be incorporated into an object model or, in some languages, an object implementation. Inheritance is often described as an "is-a" relationship. Other relationships include aggregation and association. Aggregation occurs where one object is composed, at least in part, of other objects (a "part-of" relationship), indicating some other link between object types. While these relationships types are common in object modeling nomenclatures such as the Unified Modeling Language (UML), relational models rarely categorize them using these names. However, a kind of pseudo-inheritance is available in an entity relationship (ER) model that uses supertype/subtype entities, which can be transformed into several different physical implementations. Aggregation and association can be represented as named relationships among entities and then transformed into foreign keys. The Oracle object extension does give us the ability to create relationships using a new kind of pointer called a "reference" (REF).

Oracle

Type inheritance is an essential new feature in Oracle9*i* Object-Relational Technology. With type inheritance, extending another user-defined type can create a new user-defined type. The new user-defined type is then a subtype of the supertype from which it extends. The subtype automatically inherits all the attributes and methods defined in the supertype. The subtype can add attributes and methods, and overload or override methods inherited from the supertype. Oracle supports this single-type inheritance model, which closely aligns with the ANSI/OSI SQL99 standards.

Furthermore, Oracle provides tight integration between its object-relational features and its Java Database Connectivity (JDBC) functionality. One can use a standard, generic

JDBC type to map to Oracle objects, or can customize the mapping by creating custom Java type definition classes. Custom object classes can implement either a JDBC standard SQLData interface, or the Oracle extension ORAData and ORADataFactory interfaces to read and write data. Oracle supports type inheritance in JDBC drivers that conform to the JDBC 2.0 specification.

JDBC materializes Oracle objects as instances of particular Java classes. Two main steps in using JDBC to access Oracle objects are: (1) creating the Java classes for the Oracle objects, and (2) populating these classes. Using an inheritance hierarchy of object types, one may create a corresponding inheritance hierarchy of Java classes. For example, a type inheritance hierarchy with PERSON_T and STUDENT_T types can be created in the database as follows:

```
CREATE TYPE Person_T (SSN NUMBER, name VARCHAR2(30),
address VARCHAR2(255));

CREATE TYPE Student_T UNDER Person_T (deptid NUMBER, major
VARCHAR2(100));
```

A hierarchy of Java classes implementing the ORAData interface can mirror this database object type hierarchy as follows:

Person.java

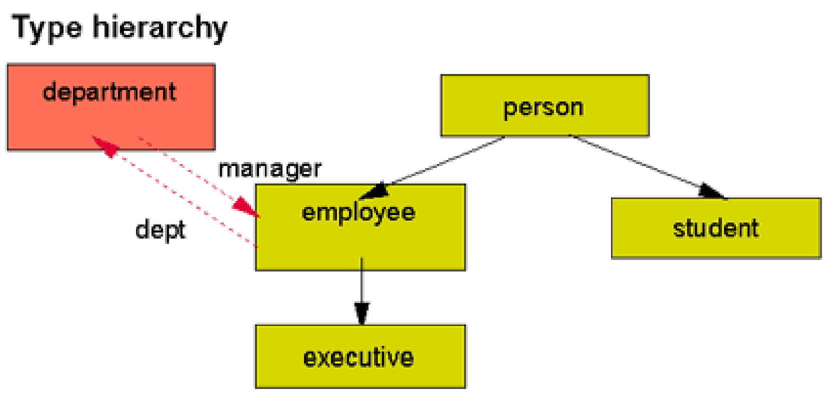
```
class Person implements ORAData, ORADataFactory
{
  static final Person _personFactory = new Person();

  public NUMBER ssn;
  public CHAR name;
  public CHAR address;

  public static ORADataFactory getORADataFactory()
  {
    return _personFactory;
  }

  public Person () {}

  public Person(NUMBER ssn, CHAR name, CHAR address)
  {
    this.ssn = ssn;
    this.name = name;
    this.address = address;
  }
  ...
}
```

Student.java (extends Person.java)

```
class Student extends Person
{
  static final Student _studentFactory = new Student ();

  public NUMBER deptid;
  public CHAR major;

  public static ORADataFactory getORADataFactory()
  {
    return _studentFactory;
  }

  public Student () {}

  public Student (NUMBER ssn, CHAR name, CHAR address,
                NUMBER deptid, CHAR major)
  {
    super (ssn, name, address);
    this.deptid = deptid;
    this.major = major;
  }
  ...
}
```

IBM

When you create a user-defined structured type, the syntax is very similar to that of creating a table: the name of the type is specified, along with its attribute names and the data types of the attributes. If you create subtypes under a structured type, those subtypes will automatically inherit the attributes of the type above it in the hierarchy (the supertype).



SQL Definitional Statements	
CREATE TYPE person AS (name VARCHAR(20), birthyear INTEGER, address VARCHAR(40)) MODE DB2SQL;	Create the root type of the hierarchy first.

CREATE TYPE employee UNDER person AS
(salary INTEGER)
MODE DB2SQL;
	Create the subtype “employee” UNDER the “person” type. All “person” attributes are inherited by “employees.” Additional attributes that are unique to “employees” (or to types created UNDER employees) can be specified.

CREATE TYPE executive UNDER employee AS
(bonus INTEGER)
MODE DB2SQL;
	Executives get bonuses.

CREATE TYPE department AS (ID INTEGER, manager REF(employee), budget INTEGER) MODE DB2SQL	The “department” type has no subtypes or supertypes. REF(employee) indicates that the manager is an “employee,” and information about employees is in the “employee” type.
METHOD BudgetperPerson() RETURNS INTEGER ...	Method specifications are included with the type definition (or can be added later with ALTER TYPE).
CREATE METHOD BudgetperPerson() RETURNS INTEGER FOR department ...	As specified in the SQL99 standard, the method body is specified in a separate CREATE METHOD statement.
ALTER TYPE employee ADD ATTRIBUTE dept _ref(department)	Now that “department” is defined, create a cyclic reference (departments referencing employees (i.e. as managers) and employees working in departments).

CREATE TYPE student UNDER person AS
(major VARCHAR(10),
wage DECIMAL)
MODE DB2SQL;
	Students are the other leg of our “person” hierarchy. They share the attributes of “person,” but also have an additional attribute to indicate their major field of study.

POLYMORPHISM

Polymorphism, by definition, means that a given operation behaves consistently even when applied to different data types. Polymorphism can be implemented at least two different ways, even without the Oracle object option:

- Module overloading allows a given PL/SQL module to have multiple specifications and bodies, as distinguished by the data types of the arguments supplied.

- Programmers can implement a kind of *ad hoc* polymorphism in the way they program object types.

The idea here is that a common core of functionality exists across the population of object types; each object knows how to respond to the requests sensibly. The classical form of polymorphism is one in which an object is not of a single type, but actually represents many types, all of which are descendants of some common supertype. This behavior in fact exists in PL/SQL's implicit data type conversions.

CONCLUSION

By comparing both Oracle and IBM's DB2 we have determined that Oracle produces a more functionally efficient and cost effective system than IBM. With Oracle's add-ons, such as the Application Programming Interface (API) and Oracle Call Interface (OCI), its Data Definition Language (DDL) and Data Manipulation Language (DML), LOBs are handled with ease and are readily manageable. Also, Oracle has created a new data type for handling large object types with BFILE, which will prove advantageous in the future. The Oracle database possesses certain advantages over its IBM competitor in that it can accommodate the ever-changing Web. They both have many similar functions and capabilities, but Oracle's data types, manipulation capabilities and functionality establish a more attractive database environment for developers when having to consider complex data objects.

REFERENCES

Ballantyne, S. (2002). Oracle's World Boils Down To a Case of Us and Them. *National Business Review 50*

Bray, M. (1997). Object-Oriented Programming Languages. *Lockheed Martin Corporation White Paper*

Kulkarni, k. (1993). Object-Orientation and the SQL Standard. *Computer Standards & Interfaces 15*, 287-300

Kumar, S. and Nori, A. (1997). Oracle 8 Object Relational Database: An Overview. *Oracle Corp. White Paper*

Oracle Corporation. (2002). Oracle 9i Object-Relational Technology. *Oracle Corp. White Paper*

Shah, A. (2002). Island Data Tames Unstructured Customer Data. *Information World Daily News*

Sullivan, D. (2002). The 80 Percent Solution. *Intelligence Enterprises Product Review 44*

Zeidenstein, K. (2001a). DB2 Object-Relational Highlights (Part 1of 2). *IBM White Paper*

Zeidenstein, K. (2001b). DB2 Object-Relational Highlights (Part 2 of 2). *IBM White paper*