CERN-IT-2001-011
September 18th 2001

Object Persistency for HEP data using an Object-Relational Database

Marcin Nowak, Dirk Düllmann, Dirk Geppert, Peter Kunszt, Stefano Paoli
CERN, IT Database Group, Geneva, Switzerland (http://cern.ch/db)

Abstract

We present an initial study of the object features of Oracle 9*i* – the first of the market-leading object-relational database systems that supports a true object model on the server side as well as an ODMG-style C++ language binding on the client side. We discuss how these features can be used to provide persistent object storage in the HEP environment.

Keywords: LHC, persistency, C++ language binding, object, database, VLDB

1. Introduction

For a number of years the former RD45 project at CERN and the IT-DB group has been investigating solutions to the problem of providing object persistency for the complex data models used by HEP experiments. Two important factors make this a very challenging task: firstly, high data volumes and throughput translate to a requirement for highly scalable and performing solutions; secondly, complex object models, that need to be exposed to many physicists, require a simple and robust navigational access from popular OO programming languages such as C++ and Java. These requirements and a study of available alternatives initially led to the proposal of a solution based on an Object Database Management System (ODBMS), coupled to a Mass Storage System (MSS). Since then several HEP experiments have implemented software frameworks using an ODBMS – Objectivity/DB – as the main persistency mechanism. In these cases the database has been used to manage the entire set of data, including raw data, derived data, "statistical" data, such as histograms and detector-related data, e.g. calibrations. The experience gained during the deployment and operation of these systems – up to the level of 300TB in the case of Babar – have confirmed that databases are a viable solution for providing persistency for HEP data.

In the early stages of the RD45 project's investigation of the database technologies, the difference between object and relational databases was clearly visible. Object database products appeared to provide the required features and in combination with the general expectation that the object database market would grow significantly – even eclipsing the relational database market according to the most extreme predictions, the project focused most of its attention on this particular solution. Since the time of the first evaluations, however, relational databases have been continuously evolving, eventually becoming object-relational hybrids. The major relational database vendors, including Oracle and IBM, started to address issues relevant to HEP data management, progressively adding support for Very Large Databases (VLDB) as well as support for user-defined data types and finally objects and OO language bindings. We believe that addition of object-oriented features to an already existing very strong relational basis makes object-relational database management systems (ORDBMS) an interesting candidate for HEP data storage.

The first ORDBMS to provide a nearly complete set of object-oriented features was Oracle 9*i* database. Oracle supports user defined object types, including inheritance, polymorphism and abstract types. Objects are understood by the database internally and can be queried with the SQL statements. The C++ language binding is provided by the Oracle C++ Call Interface (OCCI), similar to the ODMG standard used by the object databases. Navigational access to objects is possible through both SQL and OCCI. Mapping between SQL and C++ classes is done automatically by Oracle Type Translator (OTT).

Oracle 9*i* supports also a set of features that are useful in VLDB applications, i.e. partitioned tables, locally managed tablespaces and read-only and offline tablespaces, all of which are discussed in more detail below.

In this contribution we describe the new features and present the results of a preliminary evaluation, focusing on issues most relevant for the needs of HEP data management.

2. Object Modelling With SQL:1999

The SQL:1999 standard [2] introduced object-oriented extensions into the structured query language. Some reviewers even proclaim that the 'relational model is dead' [3]. SQL:1999 specifies several data types that were until now in the domain of object-oriented database systems. In particular, the standard now defines references, arrays and user defined types (UDT) along with subtypes. These features have been studied using the implementation provided in the Oracle 9*i* release, which claims to have implemented this version of the SQL standard. Oracle is also the only major vendor that provides a C++ interface to its DBMS, the Oracle C++ Call Interface. However, there is one potentially important area where Oracle deviates from the standard, which is in its support for numeric data types. SQL:1999 defines a range of data types, from SMALLINT to DOUBLE PRECISION, whereas Oracle 9*i* supports a single numeric type: Oracle NUMBER, to which all the numeric types are converted.

2.1 User Defined Types

SQL:1999 allows to create User Defined Types (UDTs) and supports single inheritance. The resulting object model is very similar to Java. Persistent objects are stored as rows in typed database tables. A given table may be filled with instances of a base type and any of its subtypes. In this sense, an object table may also be seen as a polymorph container.

2.2 Arrays

Although already implemented by many commercial database vendors, the array concept is new to the SQL standard. This enables the user to break the first normal form of the relational model (prohibiting repeating of groups) explicitly. Oracle also implements embedded tables, which is similar to an array of a UDT but with additional query capabilities.

2.3 References

The reference type enables navigational access to the data, i.e. lookup of associated objects without having to perform a table JOIN. In object-oriented programming, applications model their data as a set of inter-related objects forming graphs. *References* (REFs) can be used in a natural way to implement these relationships between objects.

3. The Oracle Type Translator for C++

Oracle provides a type translator utility, OTT, which generates C and C++ bindings to persistent classes. OTT translates schema information about Oracle object types into client-side language bindings. It generates C++ class representations of database object types, preserving inheritance hierarchy and provides default implementations for methods to read and write object data to and from the database. The user may inherit from the OTT-generated classes and thus extend the class behavior. However, the user-defined classes will not be direct subclasses of each other (see Figure 1).

4. The Oracle C++ binding

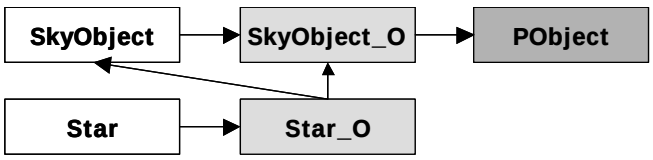

Figure 1: Inheritance in Oracle's C++ binding. All persistent classes inherit from PObject. SkyObject_O and Star_O classes are generated by OTT from their SQL equivalents. SkyObject and Star are both user-defined classes. The dashed arrow shows the inheritance as defined on the SQL side, while the solid arrows show the actual C++ data model after OTT conversion.

The Oracle C++ Call Interface OCCI [4] is an Application Program Interface (API), build on top of the lower level C binding (OCI), that has existed in Oracle for many years. OCCI provides C++ applications with both associative and navigational access to the data and with C++ language binding to the objects stored in the database.

In *associative* access, data is accessed and manipulated by executing SQL statements and PL/SQL (Procedural Language) procedures. OCCI provides C++ methods to submit the statements and to process the results. In addition, so-called Stored Procedures, i.e. methods that are part of the object type definition, may manipulate data on the server side without incurring the cost of transporting the data to the client. In this type of access, OCCI is similar to its predecessor, OCI.

Navigational access refers to accessing objects by navigating through object references. Typically, an application first retrieves one or more objects from the server by issuing a SQL query, and from then on it may use references contained in these initial objects to traverse the entire object tree.

OCCI applications use client-side object cache to speed up repeated access to the same object. The first time an object used, its content data is retrieved from the server and converted into the transient C++ object. As with most ODBMS bindings smart pointers are used to access objects in cache memory and

to automatically release objects from the cache that are not referenced anymore. Any modification to cache objects is applied also to the database if the enclosing transaction is committed.
OCCI integrates the ANSI/ISO C++ Standard including Standard Template Library (STL). For example object collections, which on the database side are represented as variable length arrays (ordered collections) or nested tables (unordered collections), are both mapped to STL vectors.

5. VLDB Features

HEP experiments collect large amounts of data. In particular, each of the upcoming LHC experiments at CERN expects to store around 1PB of raw and processed data per year. In order to consider a database system as the persistency mechanism for the HEP data, it is necessary to evaluate the VLDB (Very Large Database) features provided by the given product.

5.4 Tablespaces

In general, the implementation of the VLDB support is closely related to the database architecture and its storage technology. In Oracle the main independent storage unit is a *tablespace*. All logical database entities, like tables and indices, are stored in tablespaces, which in turn map to the operating system files and raw devices. Each tablespace can be manipulated independently and placed in different states.

- **Read-only tablespaces** – A tablespace can be marked as read only. This state guarantees that the database will prevent modifications to the data and may be useful for raw data storage. Read-only tablespace can also be copied without shutting down the database instance or they can be moved offline.
- **Offline tablespaces** – Parts of the database can be made *offline*, which makes them inaccessible until made online again. Offline tablespaces may be moved to a different location to release disk space on the server. This feature allows implementing an interface between the database and tertiary storage.
- **Transportable tablespaces** – Tablespaces themselves contain only data, while the data structure definitions, i.e. database *schema*, are grouped together in a central location. Oracle provides means to extract the schema corresponding to a given tablespace. The tablespace together with the schema describing its data structure is called a *transportable tablespace* and can be moved and attached to a different database. The main advantage of this approach over the standard import/export database functionality comes from the fact that it does not put any unnecessary load on the database server. Data copying can be performed with the system commands without interfering with the database operations. The disadvantage of the method is that it does not allow transferring data between servers of different system architecture.
- **Locally managed tablespaces** – A tablespace can be set up to automatically and independently adapt its size to the volume of data that is stored in it. Decentralization of the space management promises better scalability, and automatic adjustments remove the need of administrator's interventions.

5.5 Partitioned tables

Table partitioning is an important Oracle VLDB feature that allows creating and managing single tables of very large size. A single logical table can be divided into physically separate partitions based on the value of one key attribute. For example, all data from a single run may be stored separately. Indices created on such tables may be partitioned as well. Queries that are constrained on the same attribute that was used for partitioning will be automatically optimized by the database to access only the relevant partitions.

5.6 Storage overhead

Database storage usually introduces different types of space usage overheads. When storing persistent objects, this overhead may be caused by object identifier (OID) stored with each object or by the object attribute conversion to a different data representation. The impact of the OID overhead may be minimised by collecting smaller objects into VArrays, but the attribute conversion is difficult to avoid. In scientific applications, the most visible overhead is generated by the NUMBER type, which is used to store all numerical data. Discussions are being held with Oracle regarding efficient storage of numeric data in a future release of their product, possibly the next major release (Oracle 10*i*), or even before.

5.7 Real Application Clusters

With a database technology based on centralized servers, it is necessary to insure that the server does not become a CPU or I/O bottleneck. As single systems have limited scaling capability and often high prices, clusters of commodity computers are frequently used as an alternative. Oracle provides an architecture called Real Application Clusters (RAC) that utilises cluster technology. The RAC database

instances run on all nodes of the cluster and share the database files. All instances have a common distributed cache that is kept consistent using the cluster interconnect. Adding new nodes to the cluster – particularly in the case of read-mostly data, where cache conflicts are largely avoided – may increase the system performance at a constant price.

6. Conclusions

Object features have been added both to the SQL standard and to commercial implementations from the major database vendors. In addition, these vendors have added many performance enhancements and features oriented at the very large database community. The availability of such systems – particularly when coupled to the navigational C++ binding that is provided with Oracle 9*i*, are potentially of great interest to the HEP community. Further work is clearly needed to understand whether these products provide the required functionality, performance and scalability, but their emergence does much to allay fears about the relatively small size of the purely object database vendors.

7. References

[1] Oracle Corporation, http://www.oracle.com

[2] The SQL:1999 Standard, ISO/IEC 9075:1999

[3] Whitemarsh Information System Corp, talk at DAMA, http://www.wiscorp.com/sql/Sql99_p1.zip

[4] 'Oracle C++ Call Interface – A better OCI', Oracle Technical White Paper, April 2001

[5] 'Object-Relational Dbmss: Tracking the Next Great Wave', Morgan Kaufmann Publishers, 1998, by Michael Stonebraker, Dorothy Moore