Object Reuse and Integration in Object-Relational Databases Development

Ming Wang
ming.wang@calstatela.edu
Department of Information systems
California State University
Los Angeles, CA 90032, USA

Abstract

One of the most important features of the Object-relational database (ORDB) is object reuse and integration. This feature provides standard data structure, data portability, and maintainability for ORDB database applications development. Despite the undeniable object reuse and integration features in ORDBs, very little research has been published to address its importance in database application development in the real world. This paper presents a study to investigate object reuse and integration in the design and implementation of ORDBs.

Keywords: Relational database, Object-relational database, object-oriented technology

1. INTRODUCTION

With rapidly increasing volumes of digital information and a broadening range of applications development, applications have become more complex and software development costs have increased. This tremendous challenge has led to the idea of information reuse and integration in software development. The Object-relational database management system (ORDBMS) provides a way to solve the problem. ORDBMS enhances object-oriented technology into the relational database management system (RDBMS) and extends traditional RDBMS to ORDBMS. As an evolutionary technology, ORDBMS allows users to take advantages of reuse features in object-oriented technology, to map objects into relations and to maintain a consistent data structure in the existing RDBMS. If multiple database applications use the same set of database objects in ORDBMS, a de facto standard for the database objects is created, and these objects can be extended, reused and integrated in the ORDB.

In response to the evolutional change of ORDBMSs, SQL:1999 started supporting

object-relational data modeling features in database management standardization and SQL:2003 continues this evolution. Currently, all the major database vendors have converted their relational databases to object-relational databases to reflect the new SQL standards (Hoffer et al., 2009).

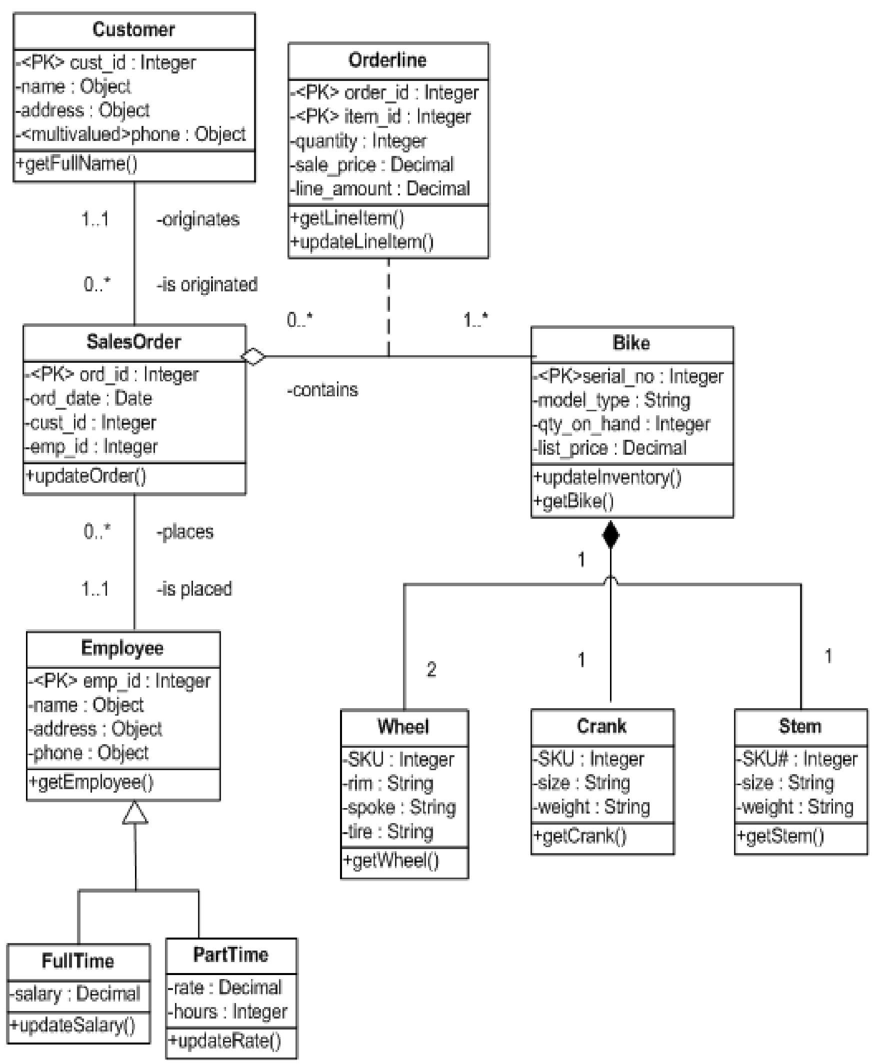
Although the ORDB technology is already available for use in all the major database vendors' products, its industrial adoption rate is not very high. One of the major criticisms of ORDBMS is that its complexity results in the loss of the essential simplicity and purity of the relational database model. It is challenging to for industrial application developers who have traditional relational database background to adopt the emergent ORDB technology.

This paper presents a case study to investigate object reuse and integration in the design and implementation of ORDBMSs. Firstly, the Unified Modeling Language (UML) class diagram is used to model the ORDB design. Secondly, Oracle SQL DCL and DML scripts are used to illustrate the ORDB implementation. Finally the paper concludes with a discussion of the advantages and implications of ORDB development. The purpose of the paper is to present object reuse and integration features in ORDBMS for industrial database application developers.

2. CASE STUDY: ORDB DESIGN

The Pacific Bike Traders Co. assembles and sells bikes to customers. The company currently accepts customer orders online and wants to be able to track orders and bike inventory. The new ORDB system will be created to handle the current transaction volume generated by employees processing incoming sales orders. The new ORDB system must be able to update the available quantity on hand to reflect that the bike has been sold and produce customer sales orders, invoices and reports showing inventory levels. Figure 1 illustrates the new object-relational database design with UML 2.0 for the The Pacific Bike Traders Company.

Figure 1 Pacific Trader's ORDB Design Using UML Class Diagram (Wang, 2006)



The following business rules are defined for the Pacific Bike ORDB scenario.

One customer may originate many orders.
One order must be originated from one customer.

One order must contain one or more bikes.
One bike may or may not be in many orders.

One employee may or may not place many orders. One order must be placed by one employee.

One bike is composed of a front wheel, rear wheel, crank, and stem.

One employee must be either full-time or part-time.

Based on the Pacific Bike Trader scenario and its business rules, a UML class diagram in Figure 1 is developed to model the Pacific Bike Trader ORDB design. Each class is displayed as a rectangle that includes three sections: the top section gives the class name; the middle section displays the attributes of the class; and the last section displays methods. Associations between classes are indicated with multiplicity ("min..max.") notation.

Aggregation is marked with an empty diamond, whereas a sales order is made of line items (bikes). The dotted line links to the associative class generated from the many-to-many relationship.

Composition models a closer whole-part relationship than aggregation. Composition is marked with a solid diamond. Aggregation models a whole-part relationship. Composition shows that inner class bike parts can be integrated in the outer class Bike.

Inheritance is indicated with an empty triangle. Inheritance means that attributes in the Employee super class are shared and reused by the Full-time and Part-time subclasses.

The object types of Name, address, and phone are reused in both customer and employee classes as well as the Full-time and Part-time subclasses.

3. CASE STUDY: ORDB DEVELOPMENT

Based on the Pacific Trader's UML class diagram in Figure 1, the six information reuse and integration features of ORDBMSs are identified and implemented with Oracle SQL Scripts. The implementation shows how the UML class diagram maps and supports Oracle ORDBMS. For the sake of simplicity, it is assumed that referential integrity constraints will be added later. This section will focus on discussion of the six information reuse and integration features: 1) Object type reuse for data standardization; 2) Encapsulated user-defined methods for standard data access; 3) Object type inheritance for data reuse; 4) Integration of nested table data; 5) Object views for reuse of relational data 6) Integration ORDB applications with object type interface.

3.1 Object type reuse

Object type is user-defined data type (UDF) or abstract data type (ADT) that is used in ORDB creation. Commonly used objects such as address and name should be defined with object types. Once object types are defined they are stored in the database permanently and can be used repetitively to create any new columns and tables in the database. Reuse of object types can standardize data stored in ORDBs. The following SQL statements define Address_ type and Name_type as object types and varray_phone_type as a VArray type for reuse in the ORDB.

```
CREATE TYPE address_type AS OBJECT
  (street  VARCHAR2(30),
   city    VARCHAR2(25),
   state   CHAR(2),
   zip     NUMBER(10));
```

```
CREATE TYPE name_type AS OBJECT
 (f_name  VARCHAR2(25),
  l_name  VARCHAR2(25),
  initial CHAR(2));
```

```
CREATE TYPE varray_phone_type AS
VARRAY(3) OF VARCHAR2(14);
```

The above Address_type, Name_type and varray_phone_type can be used to define columns in the customer table below.

```
CREATE TABLE Customer(
  Cust_ID NUMBER(5),
  CustName           name_type,
  CustAddress        address_type,
  CustPhones         varray_phone_type);
```

Object tables can also be entirely defined by an object type, instead of using relational tables consisting of one or more object columns. The employee object table can be created by the employee_type in the following statements.

```
CREATE TYPE employee_type AS OBJECT
 (emp_id       NUMBER(10),
   SSN         NUMBER(9),
   name        name_type,
   dob         DATE,
   address     address_type,
   phones      varray_phone_type);

CREATE TABLE Employee of employee_type;
```

3.2 Defined methods for reuse

Once attributes of an object type are defined, the user can define methods for each object type. Methods describe the behavior of attributes. For each object type, the user can define the methods that operate on attributes in the object type and encapsulate the methods with the attributes in the object_type. The following statements add a method to the Name_type object type interface defined in Section 3.1. The first statement adds the method header to the object type interface. The second statement adds the method body to the object type body:

```
ALTER TYPE name_ty    ADD MEMBER
 FUNCTION full_name RETURN VARCHAR2;

 CREATE TYPE BODY name_ty AS
 MEMBER FUNCTION full_name
 RETURN VARCHAR2 IS
 BEGIN
      RETURN(l_name ||  ' `  || f_name);
  END full_name;
 END;
```

The following SELECT statement calls the method defined in the Customer table.

```
SELECT c.custName.full_name ( ),
c.custAddress.City FROM customer c;
```

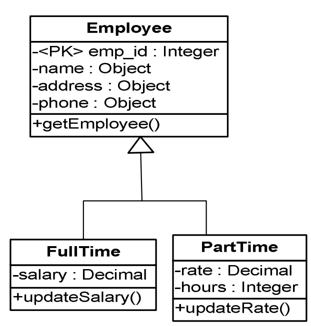
C.CUSTNAME.FULL_NAME()	CUSTADDRESS.CITY
Tommy Ford	Des Moines

The name_ty object type is associated with the full_name ( ) method, which concatenates the first and last names together. If this functionality is embedded in the server, it allows the functionality to be shared and reused by all the applications. The specified methods are privately encapsulated in the object body. Reusability of methods comes from the ability to store persistent standard data type and functionality on the server, rather than having them coded in each application.

3.3 Object type inheritance for reuse

ORDBMSs allow users to define hierarchies of data types. With this feature, users can build subtypes in hierarchies of database types. If users create standard data types to use for all employees, then all of the employee information in your database will be stored with the same internal format. Users might want to define a full time employee object type and have that type inherit existing attributes from employee_ty. The full_time_ty type can extend employee_ty with attributes to store the full time employee's salary. The part_time_ty type can extend employee_ty with attributes to store the part-time employee's hourly rates and wages. Inheritance allows for the reuse of the employee_ty object data type. The details are illustrated in the following class diagram in Figure 2.

Figure 2 Object type inheritance



Object type inheritance was one of the new features of Oracle 9i/10g. For employee_ty to be inherited, it must be defined using the

NOT FINAL clause because the default is FINAL, meaning that object type cannot be inherited. Oracle 9i can also mark an object type as NOT INSTANTIABLE; this prevents objects of that type from being derived. Users can mark an object type as NOT INSTANTIABLE when they use the type only as part of another type or as a super_type with NOT FINAL. The following example marks address type as NOT INSTANTIABLE:

```
CREATE TYPE employee_ty AS OBJECT (
  emp_id       NUMBER,
  SSN          NUMBER,
  name         name_type,
  dob          DATE,
  phone        varray_phone_type,
  address      address_type
) NOT FINAL NOT INSTANTIABLE;
```

To define a new subtype full_time_ty inheriting attributes and methods from existing types, users need to use the UNDER clause. Users can then use full_time_ty to define column objects or table objects. For example, the following statement creates an object table named FullTimeEmp.

```
CREATE TYPE full_time_ty UNDER
employee_ty (Salary  NUMBER(8,2));

CREATE TABLE FullTimeEmp of full_time_ty;
```

The preceding statement creates full_time_typ as a subtype of employee_typ. As a subtype of employee_ty, full_time_ty inherits all the attributes declared in employee_ty and any methods declared in employee_ty. The statement that defines full_time_ty specializes employee_ty by adding a new attribute "salary". New attributes declared in a subtype must have names that are different from the names of any attributes or methods declared in any of its supertypes, higher up in its type hierarchy. The following example inserts row into the FullTimeEmp table. Notice that the additional salary attribute is supplied.

A supertype can have multiple child subtypes called siblings, and these can also have subtypes. The following statement creates another subtype part_time_ty under Employee_ty.

```
CREATE OR REPLACE TYPE part_time_ty
UNDER employee_ty (rate Number(7,2),
  hours Number(3))NOT FINAL;

CREATE TABLE PartTimeEmp of part_time_ty;
```

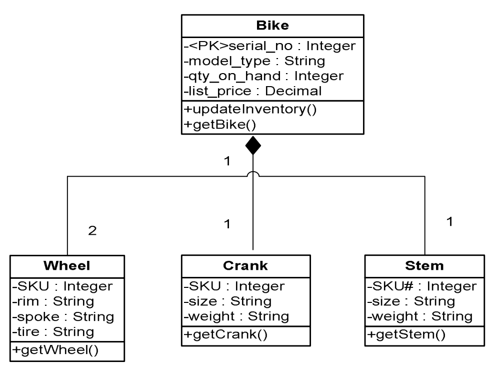
A subtype can be defined under another subtype. Again, the new subtype inherits all the attributes and methods that its parent type has, both declared and inherited. For example, the following statement defines a new subtype student_part_time _ty under part_time_ty. The new subtype inherits all the attributes and methods of student_part_time _ty and adds two attributes.

```
CREATE TYPE student_part_time_ty UNDER
part_time_ty(school  VARCHAR2(20),
      year VARCHAR2(10));
```

3.4 Integration of data in nested tables

A nested table is a table that can be stored within another table. With a nested table, a collection of multiple columns from one table can be placed into a single column in another table. Nested tables allow users to embed multi-valued attributes into a table, thus forming an object. Figure 3 illustrates the integration of three nested tables to the outer table Bike.

Figure 3 Integration of nested tables




```
CREATE TYPE wheel_type AS OBJECT(
SKU      VARCHAR2(15),
rim      VARCHAR2(30),
spoke    VARCHAR2(30),
tire     VARCHAR2(30));

CREATE TYPE crank_type AS OBJECT
(SKU       VARCHAR2(15),
crank_size  VARCHAR2(15),
crank_weight  VARCHAR2(15) );
```

```
CREATE TYPE stem_type AS OBJECT
(SKU        VARCHAR2(15),
stem_size     VARCHAR2(15),
stem_weightVARCHAR2(15));
```

The following scripts creates the nested table types: wheel_type, crank_type and stem_type.

```
CREATE TYPE nested_table_wheel_type AS
TABLE OF wheel_type;
CREATE TYPE nested_table_crank_type AS
TABLE OF crank_type;
CREATE TYPE nested_table_stem_type AS
TABLE OF stem_type;
```

The following example creates the table named Bike with that contains four nested tables:

```
CREATE TABLE bike
( serial_no      INTEGER PRIMARY KEY,
  model_type VARCHAR2(20),
  front_wheel nested_table_wheel_type,
    rear_wheel nested_table_wheel_type,
    crank nested_table_crank_type,
    stem nested_table_stem_type
)
NESTED TABLE front_wheel
  STORE AS front_wheel,
NESTED TABLE rear_wheel
  STORE AS rear_wheel,
NESTED TABLE crank
  STORE AS nested_crank,
  NESTED TABLE stem
STORE AS nested_stem;
```

The following statement shows the output of the nested tables created in the table Bike.

DESC Bike;

Name	Type
SERIAL_NO	NUMBER(38)
MODEL_TYPE	VARCHAR2(20)
FRONT_WHEEL	NESTED_TABLE_WHEEL_TYPE
REAR_WHEEL	NESTED_TABLE_WHEEL_TYPE
CRANK	NESTED_TABLE_CRANK_TYPE
STEM	NESTED_TABLE_STEM_TYPE

3.5 Object views for reuse of relations

Object view allows users to develop object structures on the top of the existing relational tables. Object view creates a layer on the relational database so that the database can be viewed in terms of objects. The object view is a bridge that can be used to create object-oriented applications without modifying existing relational database schemas (Loney, K. & Koch, 2002). This enables you to develop OO features with existing relational data. It is a bridge between the relational database and OO programming. The object view is a bridge that can be used to create object-oriented applications without modifying existing relational database schemas. By calling object views, relational data can be retrieved, updated, inserted, and deleted as if such data were stored as objects. Using object views to group logically-related data can lead to better database performance. The following example shows how the object view reuses existing relational data and retrieves Analysts as object data from the relational SalesOrder table.

Relational table: SalesOrder

ORD_ID	ORD_DATE	CUST_ID	EMP_ID
100	05-SEP-05	1	1000
101	05-OCT-05	1	1001

The following statements show how to create an object view on the SalesOrder relational table:

```
CREATE TYPE SalesOrder_type AS OBJECT(
  sales_ord_id    NUMBER(10),
  ord_date          DATE,
  cust_id NUMBER(10),
  emp_id NUMBER(10));

CREATE VIEW customer_order_view
  OF SalesOrder_type
  WITH OBJECT IDENTIFIER (sales_ord_id)
      AS
  SELECT  o.ord_id, o.ord_date, o.cust_id
  FROM salesOrder o
  WHERE o.cust_id = 1;
```

The following SQL statement generates the output of the object view:

```
SELECT * FROM customer_order_view;
```

SALES_ORD_ID	ORD_DATE	CUST_ID
100	05-SEP-05	1
101	01-SEP-05	1

An object integration solution provides an integrated view of object data, regardless of where that data is actually located in the systems.

3.6 Integration with object interface

The structure of object type includes an interface and a body. The public interface declares the data structure and the method header shows how to access the data. This public interface serves as an interface to applications. The private implementation fully defines the specified methods.

Public Interface

Specification:
Attribute declarations
Method specifications

Private Implementation

Body:
Method implementations

The following statement displays the public interface of the object type name_type. The output of the name_type public interface shows attributes and method headers as follows:

DESC name_ty;

Name	Type
F_NAME	VARCHAR2(25)
L_NAME	VARCHAR2(25)
INITIALS	CHAR(2)

METHOD
MEMBER FUNCTION **FULL_NAME** RETURNS **VARCHAR2**

Although the user-defined methods are defined with object data within the object type, they can be shared and reused in multiple database application programs. This can result in improved operational efficiency for the IT department, as well, by improving communication and cooperation between applications.

4. CONCLUSION

The main contribution of this paper is to identify, present and implement object reuse and integration features of ORDBMSs in a real-world scenario. The presented case will promote awareness and recognition of object reuse and integration features of ORDBMS.

The beauty of ORDBMSs is reusability and sharing. Reusability mainly comes from storing data and methods together in object types and performing their functionality on the ORDBMS server, rather than have them coded separately in each application. Sharing comes from using user-defined standard data types to make the database structure more standardized (Breg & Connolly. 2010)

The significance of the paper is to provide readers with guidelines on how to design and implement ORDBMSs with object reuse and integration features. The use of ORDBMS to develop applications can enforce the reuse of varying user-defined object types, provide programmers' an integrated view of data and allow multiple database applications to operate cooperatively. Ultimately, this can result in improved operational efficiency for the IT department, increase programmers' productivity, lower development effort, decrease maintenance cost, reduce the defect rate, and raise the applications' reliability. With object reuse and integration, and a standard adherence access path, database application developers can create a de facto standard for database objects and multiple database applications to make database application development more productive and efficient.

The solution to the presented case can be generalized in either the projects of advanced database courses or industrial database application development. Major relational database vendors have upgraded their products to Object-relational database management systems (ORDBMSs) and ready to be used by industrial practitioners. Practically, ORDBMSs allows the users to take advantages of OODBMS and to maintain a consistent data structure in an existing relational database. Theoretically, as Stonebraker (1996) predicted in his four-quadrant view of the database world more than ten years ago, ORDBMS may be the most appropriate DBMS that processes complex data and complex queries.

5. REFERENCES

Begg, C and Connolly, T. (2010) Database systems: A Practical Approach to Design, Implementation, and Management, 5th Ed. Addison Wesley.

Hoffer, J. A., Prescott, M. B. & McFadden F. R. (2009) Modern Database Management, 9th Ed. Prentice Hall.

Loney, K. & Koch, G. Oracle 9i: The Complete Reference, Oracle Press/McGraw-Hill/Osborne, 2002.

Stonebraker M. and Moore, D. 1996. Object-relational DBMSs: the Next Great Wave. San Francisco, CA: Morgan Kaufmann Publishers, Inc.

Wang, M. (2006) "Teaching ORDB with UML Class Diagram in an Advanced Database Course", Journal of Information Systems Education. 17(1); pp.73-83.