Teaching ORDB with UML Class Diagram in an Advanced Database Course

Ming Wang
California State University, Los Angeles
5151 State University Drive
Los Angeles, CA 90032
ming.wang@calstatela.edu

ABSTRACT

Object-relational database technology emerged as a way of enhancing object-oriented features in relational database management systems (RDBMSs). In response to this evolutionary change, the author has incorporated the technology into her advanced database course. This paper presents a teaching case on using UML (Unified Modeling Language) for object-relational database (ORDB) design and its implementation with Oracle. Course organization, course content, class activities, and impacts on the students' learning outcomes are discussed. The paper is intended to provide a guide for database instructors who desire to incorporate object-relational technology and design in their traditional database courses.

Keyword: UML, Object-relational database, Relational database, Database education, Database course

1. INTRODUCTION

The success of Relational Database Systems (RDBMSs) cannot be denied, but they experience difficulty when confronted with the kinds of "complex data" found in advanced application areas such as geographic locations and computer aided design (CAD). To meet the challenges, major database vendors such as Oracle, IBM and Microsoft have moved to incorporate object-oriented features into their RDBMSs under the name of object-relational database management systems (ORDBMSs). In response to the evolutionary change of ORDBMSs, SQL:1999 started supporting object-relational features in database management standardization. SQL:2003 continued this evolution.

An object-relational database model (ORDBM) is a hybrid of a relational database model (RDBM) and an object-oriented database model (OODBM). ORDBM is essentially a relational data model with object-oriented extensions (Grorge, 2003, Elmasri & Navathe, 2003). As an evolutionary technology, ORDBM has inherited the robust transaction and performance-management features of its relational ancestor and the flexibility of its object-oriented cousin.

ORDBMSs allow users to take advantage of OODBM and to maintain a consistent data structure in an existing RDB. With ORDBMS, database designers can work with familiar tabular structures and data definition languages (DDLs) while assimilating new object-management possibilities (Krishnamurthy *et al.,* 1999). Many organizations have invested heavily in relational databases for the past decades and are not ready to switch to object-oriented database management systems (OODBMSs). These organizations would appreciate ORDBMSs that allow them to take advantage of the new extensions in an evolutionary way without losing the benefits of current database features and functions (Garcia-Molina *et al.*, 2003).

Stonebraker and Moore (1996) conclude that ORDBMSs will be spurred by the rise of new multimedia applications and Web applications in traditional organizations. Their prediction has been vindicated by recent developments in the database market. Garcia-Molina *et al.* (2003) indicate that OODBMSs made limited inroads during the 1990's, but interest has waned. Instead of a migration from relational to object-oriented DBMSs, as was widely predicted around 1990, the vendors of RDBMSs have moved to incorporate many of the ideas found in object-oriented database proposals.

The emergence of object-relational technology into the commercial database market has piqued the database professional's interest in conceptual database design and brought new challenges for IS instructors in teaching ORDBMS in their database courses. Although the UML class diagram was not created for database design, it can be utilized to model ORDB design because it primarily supports the ORDB's data representation and functionality. Transforming a UML class diagram an ORDBM consists of its logical data structure and data behavior. This is not the case with the traditional Entity-Relationship (ER) model. Unlike RDB, object types in ORDB fully support encapsulation, where method definitions can be associated explicitly with object type definitions.

This paper introduces a model of teaching ORDBM using UML class diagrams with ORDBM design, mapping UML class diagrams to ORDBM and implementing ORDBM with Oracle. The paper provides course organization, class activities, and concludes with a summary of learning outcomes and a discussion of the significance of this practice. The paper also presents a case study to illustrate the design, mapping and implementation of UML class diagram for ORDBM in details. The ORDBMS script included in the case study has been tested with Oracle 9i/10g software packages. The teaching model is based on the author's teaching experience in the advanced database course. The author started teaching UML for ORDB design and implementing it with Oracle 8i in database courses five years ago (Wang, 2001). Through experience, a teaching model has been developed and refined. The paper is intended to serve as a useful teaching resource for those who are interested in teaching object-relational technology in their traditional database courses. Since researchers, major database vendors, industrial users, and the SQL standards have warmly embraced ORDBMS, integration of ORDBM into the advanced database course should become an essential topic in IS education.

2. COURSE ORGANIZATION AND ACTIVITIES

This paper describes a teaching model for an advanced database course for seniors in the quarterly academic system. Students enrolled in this course are required to have a prerequisite in an introductory database systems course. The introductory database course covers relational database concepts, database design, management and implementation. Conceptual and logical design is covered in three weeks, normalization in one week, Physical design and database creation in one week, and structured query language (SQL) in two weeks. The course project provides students hands-on experience in Oracle 9i/10G, goes through all the phases of the database development process and summarizes all the topics of the course.

The goal of the advanced database system course is to introduce students to advanced database development technology and database application development with tools. With recent advances in object-relational technology and its acceptance in the database industry, it is important for undergraduates to understand and implement ORDBMSs (Urban and Dietrich 2003). It will be appropriate to add this new topic to the following IS2002 curriculum courses (Gorgone et al., 2002):

IS2002.8 Physical Design and Implementation with DBMS
IS2002.9 Physical Design and Implementation in Emerging Environments
IS2002.10 Project Management and Practice

2.1 Course Outline and Activities

Class activities are based on solving problems associated with complex data in the real world. The strategy is to pick a mini-world scenario in which students can have hands-on experience through the learning process. The class activities concentrate on following topics:

2.2 Re-engineering From Database Creation Script To A Relational Schema

Given an Oracle relational database creation script, students are asked to re-engineer it to a relational schema, load data in the tables and create database applications with Access. An order system is the core the e-business system. The following order database creation script is chosen as an in-class example for students to refresh their knowledge learned in their introductory database class.

```
CREATE TABLE Customer
                                                    (Cust_ID
INTEGER        NOT NULL,
            Cust_LName     VARCHAR(25),
  Cust_FName     VARCHAR(25),
  Cust_Zip       VARCHAR(10),
CONSTRAINT    CUSTOMER_PK    PRIMARY    KEY
(Cust_Id));

CREATE TABLE Sales_Order
     (Order_ID        INTEGER       NOT NULL,
      Order_Date      DATE,
       Cust_ID         INTEGER,
CONSTRAINT Order_PK PRIMARY KEY (Order_ID),
CONSTRAINT    Order_FK    FOREIGN    KEY    (Cust_Id)
REFERENCES Customer(Cust_Id));

CREATE TABLE Product  (
       SKU                     VARCHAR(10),
       Description   VARCHAR(50),
       Price                   NUMBER(6,2),
CONSTRAINT Product_PK PRIMARY KEY (SKU));

CREATE TABLE ORDER_LINE (
      Order_ID                INTEGER,
                                                  SKU
                  VARCHAR(10),
            Order_Qty     INTEGER       NOT NULL,
CONSTRAINT    Order_Line_PK    PRIMARY    KEY
(Order_Id, SKU),
CONSTRAINT    Order_Line_FK1    FOREIGN    KEY
(Order_Id) REFERENCES Sales_Order(Order_Id),
CONSTRAINT  Order_Line_FK2  FOREIGN  KEY  (SKU)
REFERENCES Product(SKU));
```

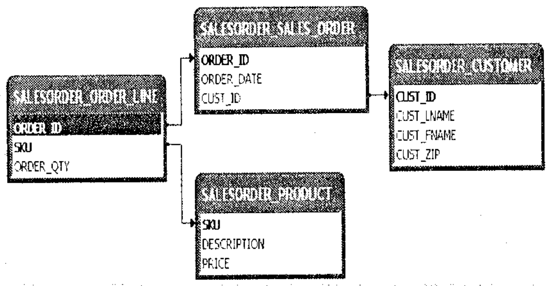


Figure 1 Relational Schema of the Order System

Students are asked to install the above SQL script into their own Oracle accounts, export the four tables to a Microsoft Access DBMS using an Oracle ODBC link, and generate the following relational schema with referential integrities via Access with the guide provided by the instructor.

Further students are asked to normalize the table up to the Third Normal Form (3rd NF) and to indicate the normal form of each table based on their functional dependency analysis. The normalization form table template is shown as follows:

	1st NF	2ndNF	3rdNF
Customer			
Product			
Order_Line			
Sales_Order			

Table 2 Normalization Form Table Template *

* Write down an X in the table to show the normal form of each relational table.

Students are also asked to enter three to five rows of data in each of the above tables, to create order application forms and report based on the above relational schema using Access. Not only do the activities review what students have learned in their introductory database course, they also lay the foundation for learning the UML class diagram for ORDB development.

2.1.1 Conversion of the Relational Schema to an ER Diagram: Based on the above relational schema, students are asked to convert it to an ER diagram, to illustrate its business rules and to indicate the normal forms for the tables after functional dependency analysis.

Further students are asked to write down the business rules for the above ER diagram as follows:

One customer may or may not originate many orders.
One order must be originated by only one customer.

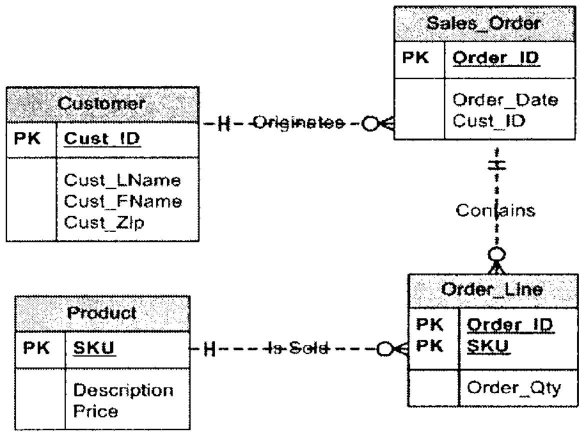


Figure 2 ER Diagram of the Order System

One sales order may contain many ordered lines.
One order line must be contained in only one sales order.

One product may or may not be sold for many order lines.
One order line must sell only one product.

2.1.2 Transformation from an ER diagram to a UML Class diagram: Given the mapping rules in Table 1, students transform the above re-engineered ER diagram to an UML class diagram. The students' mastery of UML class diagrams is demonstrated by appropriately mapping from an ERD to a UML diagram or vice versa.

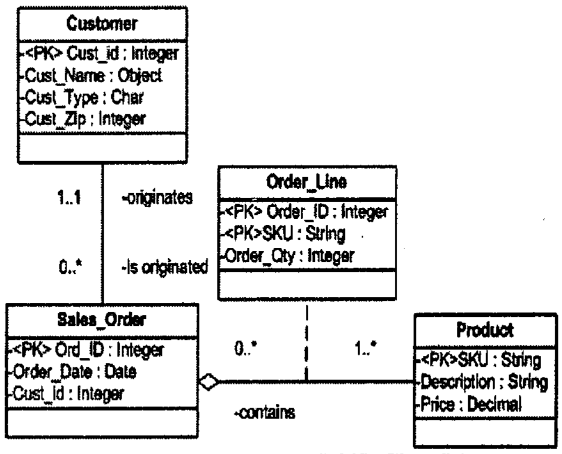


Figure 3 Product Order UML Class Diagram

ERD and UML have similarities. The rules of transformation from ERD to UML are summarized in Table 1.

Of course, the UML class diagram provides additional advanced concepts that are beyond the ERD scope. Aggregation and composition are association details that are absent from the ERD. Aggregation is an association that models a whole-part relationship. Composition is a stronger aggregation association. In Figure 3, the empty diamond symbol shows the aggregation relationship between Product and Sale Order class. In Figure 4 in Section 3, the solid diamond symbol shows a composition relationship between Bike and Wheel, Crank and Stem.

ERD	UML
Entity	Class
Composite entity	Associative class
Attributes	Attributes
Roles	Roles
Connectivity	Multiplicity
Participation	Multiplicity
Cardinality	Multiplicity
Generalization/Specialization	Inheritance
Relationship	Association

Table 1 Transforming from ERD and UML*

* Modified from Chaudhri, A. And Zicari, R. 2000

2.1.3 Implementation of the UML Class Diagram for ORDB Design: Given the above UML class diagram for the RDB, students are asked to extend it to ORDB by adding the appropriate object types, methods, collection types, inheritance, and object-views using Oracle 9i /10g. SQL:1999 and SQL:2003 object type syntax is used to manipulate and query the developed ORDB. The following object-relational features are implemented (see the case study in Section 4 in details:

- — User-defined object types
- — User-defined methods
- — VARRAY for multi-value attribute
- — Nested table for composition
- — Inheritance
- — Object views on a relational table

2.1.4 Development of database applications using Oracle Tools: Based on the created Oracle ORDB, students are asked to develop event triggers, cursors, and object views of an object hierarchy with PL/SQL and embed them in Oracle forms and reports using Oracle Form Builder and Report Developers.

2.1.5 Development of Database Applications with Java Server Technology: By querying over objects to the created Oracle ORDB, students develop database applications with Java Database Connectivity (JDBC) and Java Server Page (JSP)/Java Servlet. JDBC essentially defines the implementation of SQL in the database using the Java language. Its definition ranges from the low-level API that connects and communicates with the database to the high-level API used to move data to and from the Java application. It works as a connector and translator between the database and Java applications.

Table 3 shows a set of JDBC programming templates that guide students in writing database applications. A set of the four JDBC console templates (insertion, deletion, update, and selection) are the theme of the course. JDBC Graphical User Interface (GUI) application, JSP Web application, JSP PL/SQL application and JavaBean application templates are derived from the set of four JDBC console templates on Unix. The menu interface template integrates the four application templates and calls the four operations. Six sets of program templates utilized in the course are illustrated from top-down sequentially in the following table (Wang, 2003).

	Application Templates	Menu Interface
JDBC Unix	I, D, U,S	Java file
JDBCGUI	I, D, U,S	JavaSwing
JSPServlet	I, D, U,S	Index.html file
JDBC&PL/SQL	I, D, U,S	Java Swing File
JSP&PL/SQL	I, D, U,S	Index.html file
JSP&JavaBean	I, D,	index.html file

S = Selection I = Insertion D = Deletion U = Update

Table 3 Database Web Application Templates

3. CASE STUDY: ORDB DESIGN WITH UML

3.1 Case Scenario

Pacific Bike Traders 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 existing database system cannot handle the current transaction volume generated by employees processing incoming sales orders. When a customer orders a bike, the system must confirm that the ordered item is in stock. The system must update the available quantity on hand to reflect that the bike has been sold. When Pacific Bike Traders receives new shipments, a receiving clerk must update the inventory to show the new quantity on hand. The system must produce invoices and reports showing inventory levels.

3.2. Business Rules

The following business rules are developed for the new database system:

One customer may or may not originate many orders.
One order must be originated from one and only 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 and only one employee.

One bike is composed with a front wheel, rear wheel, crank, and stem.
One front wheel, rear wheel, crank, and stem compose one bike.

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

3.3. UML Class Diagram

The Pacific Trader Database design is illustrated with the UML class diagram in Figure 4. Each of the classes 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 that operate on the data in the object. Associations between classes are indicated with multiplicity ("min..max." notation). Inheritance is indicated with an empty triangle. Aggregation is marked with an empty diamond, whereas composition is marked with a solid diamond. Aggregation models a whole-part relationship where individual items become elements in a new class. In Figure 4, a sales order is made of line items (bikes). Aggregation is indicated by a small empty diamond next to the SalesOrder class. The dotted line links to the associative class generated from the many-to-many relationship.

4. CASE STUDY: IMPLEMENTATION OF ORDBM

Based on the Pacific Trader's UML class diagram, the following ORDB features are implemented with Oracle 9i/10g. The implementation shows how the UML class diagram maps and supports major ORDB features. For the

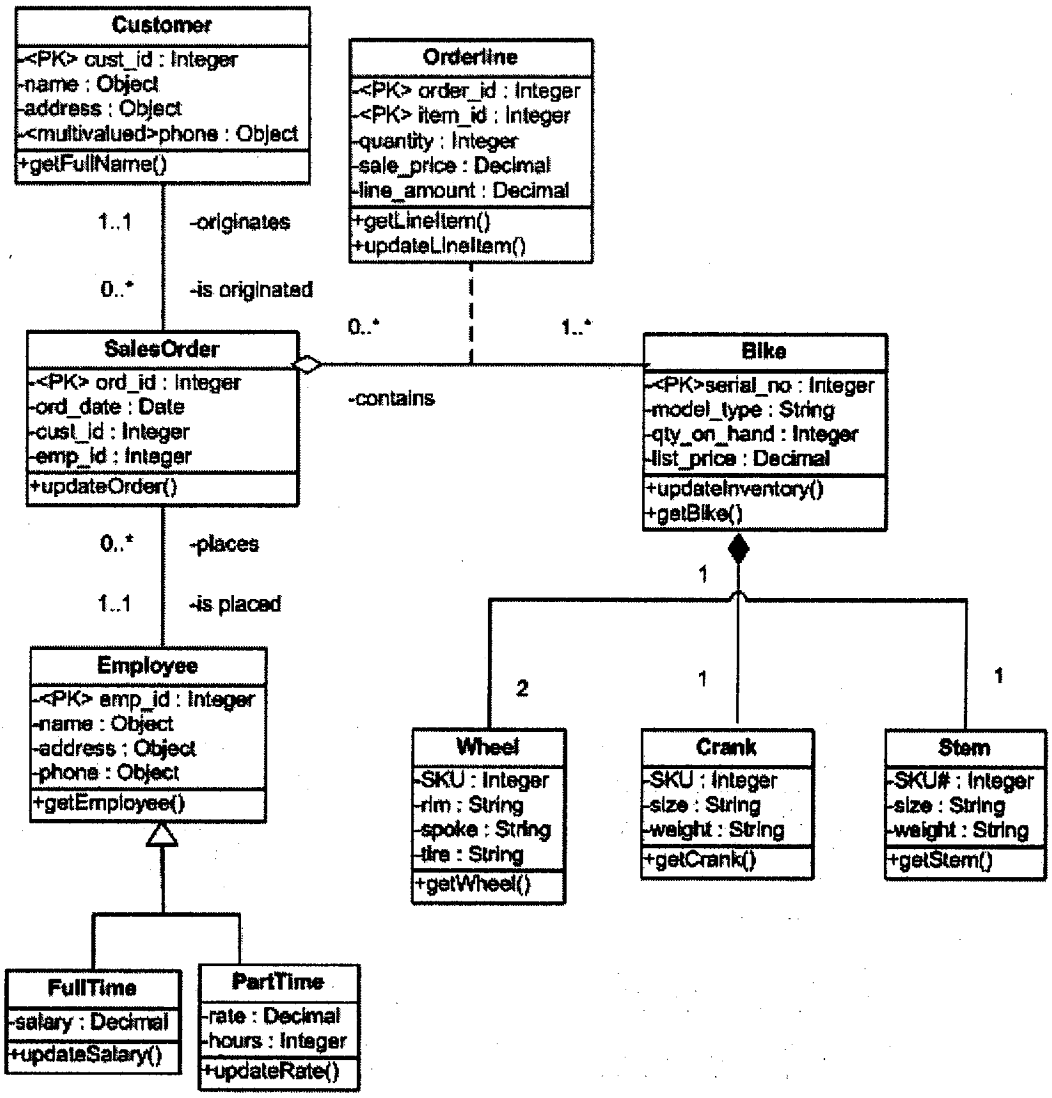
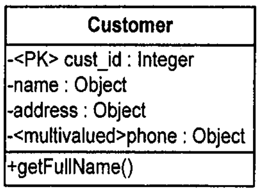


Figure 4 Pacific Trader's UML Class Diagram

sake of simplicity, it is assumed that referential integrity constraints will be added later. The following object-relational features are implemented for the case:

4.1 User-Defined Object Types

User-defined data types (UDT) or abstract data types (ADT) are referred to as object types. Object types are used to define either object columns or object tables. The following UML Customer class is defined with object columns.



The following SQL statements define the object types: address_ty and name_ty.

```
CREATE OR REPLACE TYPE address_ty AS OBJECT
(street     NVARCHAR2(30),
 city       VARCHAR2(25),
 state      CHAR(2),
 zip        NUMBER(10));

CREATE OR REPLACE TYPE name_ty AS OBJECT (
f_name    VARCHAR2(25),
l_name    VARCHAR2(25),
initials   CHAR(2));
```

Mapping the above customer class, the following statement is used to create the Customer table with the CustName and

CustAddress object columns using name_ty and address_ty. The column phone will be added to the table later.

```
CREATE TABLE Customer(
Cust_ID            Number(5),
CustName           name_ty,
CustAddress        address_ty);
```

Type constructors are used to insert object data into the table. The following INSERT statement uses constructors name_ty() and address_ty() to add data into the two object columns.

```
INSERT INTO Customer VALUES (
1, name_ty ('Tommy', 'Ford', 'TF' ),
address_ty ('3 Pine Street', 'Des Moines', 'IA', 52345));
```

The following SELECT statement retrieves data from object columns in the Customer table. The syntax of the second SELECT statement displays an individual attribute of the Address object column. The letter "c" used as a prefix to Address is an alias of the faculty table. Since Address was created as an object column, it is referred to by table_name.object_column_name.data.

```
SELECT c.custName.l_name, c.custAddress.City,
c.custAddress.state FROM Customer c;
```

C.L NAME	C.CITY	State
Ford	Des Moines	IA

```
SELECT * from Customer;
```

CUST_ID	CUSTNAME(F_NAME, L_NAME, INITIALS)	CUSTADDRESS(STREET, CITY, STATE, ZIP)
1	NAME_TY('Tommy', 'Ford', 'TF')	ADDRESS_TY('3 Pine Street', 'Des Moines', 'IA', 52345)

4.2. User-Defined Methods

Once an object type is defined, the user can define methods for each object type for data access. Methods define the behavior of data. The following statements modify the name_ty interface and add a method to the defined object type name_ty. The first statement adds the method header to the object interface. The second statement adds the method implementation to the body of the object type.

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

        CREATE OR REPLACE TYPE BODY name_ty
        AS MEMBER FUNCTION full_name
        RETURN VARCHAR2 IS
          BEGIN
                  RETURN(f_name || ' ' || l_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 by all the applications, instead of being defined in each application. 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.

The structure of an object type includes an interface and a body. Methods are defined in the object interface and implemented in the object body. In Oracle, 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. The rule is to modify the body without changing the public interface and without affecting the client program.

Public interface

Specification:
Attribute declarations
Method specifications

Private implementation

Body:
Method implementations

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

```
DESC name_ty
```

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

```
    METHOD
    MEMBER FUNCTION FULL_NAME
RETURNS VARCHAR2
```

4.3. VARRAY for Multi-Value Attribute

VARRAY is a collection type in ORDBMSs. A VARRAY consists of a set of objects that have the same predefined data type in an array. In a relational model, multi-valued attributes are not allowed in the first normalization form. The solution to the problem is that each multiple-valued attribute is handled by forming a new table. If a table has five multi-valued attributes, that table would have to be split into six tables after the First Form of normalization. To retrieve the data back from that original table, the student would have to do five joins across these six tables. ORDBMs allow multi-valued attributes to be represented in a database. ORDBMSs allow users to create the varying length array (VARRAY) data type to be used as a new data storage method for multi-valued attributes. The following statement defines a VARRAY type of three VARCHAR2 strings named varray_phone_ty to represent a list of three phone numbers in the Customer table.

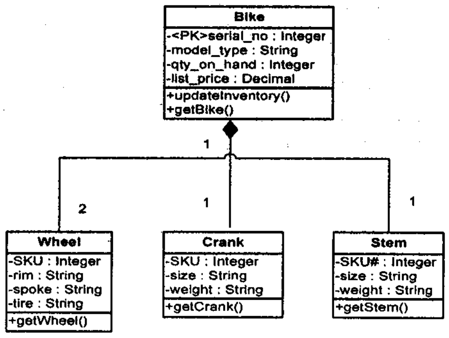
```
CREATE TYPE varray_phone_ty AS VARRAY(3) OF
VARCHAR2(14);
          ALTER TABLE Customer ADD (phones
varray_phone_ty);
         UPDATE customer
         SET phones = (varray_phone_ty('(800)555-1211',
                 '(800)555-1212','(800)555-1213'))
                 WHERE cust_id = 1;

         INSERT INTO customer(phones) values
(varray_phone_ty('(800)555-
1211','(800)555-1212','(800)555-1213'));
```

The above example shows that using the varying length array (VARRAY) data type can not only solve the Non-1NF problem for the customer table, but also can speed up the query process on customer data.

4.4. Nested Table for Composition Association

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 user to embed multi-valued attributes into a table, thus forming an object.



```
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_weight       VARCHAR2(15)
        );
```

The following statement creates 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 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;
```

Finally the next statement inserts a row into the Bike table with nested tables using the three defined constructors:

```
INSERT INTO bike VALUES (1000, 'K2 2.0 Road',
          nested_table_wheel_type(wheel_type('w7023',
'4R500', '32 spokes', '700x26c' )),
          nested_table_wheel_type(wheel_type('w7023',
'4R500', '32 spokes', '700x26c' )),
          nested_table_crank_type(crank_type('c7023',
'30X42X52', '4 pounds')),
          nested_table_stem_type(stem_type('s7023',
'M5254', '2 pounds')));
```

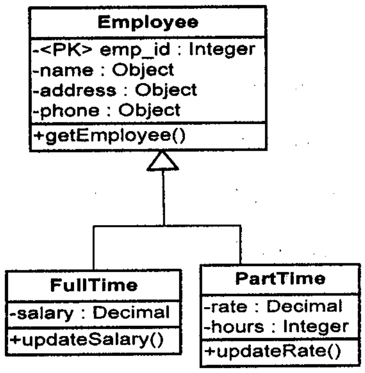
The following statement shows the nested tables in the table Bike.

SERIAL_NO	FRONT_WHEEL (SKU, RIM, SPOKE, TIRE)	REAR_WHEEL (SKU, RIM, SPOKE, TIRE)	CRANK (SKU, CRANK_SIZE, CRANK_WEIGHT)	STEM (SKU, STEM_SIZE, STEM_WEIGHT)
1000	(WHEEL_TYPE ('w7023', '4R500', '32 spokes', '700x26c'))	(WHEEL_TYPE ('w7023', '4R500', '32 spokes', '700x26c'))	(CRANK_TYPE('c7023', '30X42X52', '4 pounds'))	(STEM_TYPE('s7023', 'M5254', '2 pounds'))

The above example shows that using the NESTED TABLE can implement composition association, store multiple parts and also speed up the data retrieval speed for the Bike table.

4.5 Object type Inheritance

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 employees in your database will use 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:



Object type inheritance is one of new features of Oracle 9i. For employee_ty to be inherited from, 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 since employee_ty is a super_type. 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_ty,
 dob               DATE,
 phone             varray_phone_ty,
 address           address_ty
 ) 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.

```
INSERT INTO FullTimeEmp VALUES (1000, 123456789,
name_ty('Jim', 'Fox', 'K'), '12-MAY-1960',
varray_phone_ty('(626)123-5678', '(323)343-2983',
'(626)789-1234'),
Address_ty ('3 Lost Spring Way', 'Orlando', 'FL', 32145),
45000.00);

SELECT f.emp_id, f.name, f.phone  FROM FullTimeEmp f;
```

EMP_ID	NAME (F_NAME, L_NAME, INITIALS)	PHONE
1001	NAME_TY ('Jim', 'Fox', 'K ')	VARRAY_PHONE_TY ('(626)123-5678', '(323)343-2983', '(626)789-1234')

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. 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));
```

4.6. Object Views on a Relational Table
Object view allows users to develop object structures in existing relational tables. Object view creates a layer on top of the relational database so that the database can be viewed in terms of objects (Loney & Koch, 2002). Object views are virtual object tables, which allow you to add Object-Oriented (OO) structures on top of your existing relational tables. This enables you to develop OO features with existing relational data. It is a bridge between the relational database and OO programming. The following statements show how to create the SalesOrder table:

```
CREATE TABLE SalesOrder (
  ord_id  NUMBER(10),
  ord_date DATE,
  cust_id NUMBER(10),
  emp_id NUMBER(10));

INSERT INTO SalesOrder VALUES
     (100,'5-Sep-05', 1, '1000');
INSERT INTO salesOrder VALUES
     (101, '1-Sep-05', 1, '1000');
```

The following statements show how to create an object view on the top of 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, o.emp_id
                  FROM salesOrder o
                     WHERE o.cust_id = 1;
```

The following SQL statement generates the view output:

```
          SELECT * FROM customer_order_view;
```

SALES_ORD_ID	ORD_DATE	CUST_ID	EMP_ID
100	05-SEP-05	1	1000
101	01-SEP-05	1	1000

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. The following statement can retrieve Analysts as object data from the relational SalesOrder table. Using object views to group logically-related data can lead to better database performance.

5. LEARNING OUTCOMES

Teaching ORDBMSs with the UML class diagram in the advanced database course provides students hands-on experience with the challenge of the new object-relational technology. At the end of the course, students are able to utilize the UML class diagram to design ORDB, implement it with Oracle 9i/10g, and create ORDB applications using various tools. As a result, the following learning outcomes are demonstrated at the end of the class. Students are able to:

1. Transform an ER diagram to a UML class diagram
2. Design ORDB using UML class diagrams
3. Create object columns and object tables with user-defined object types and methods
4. Solve non-1NF problems with the VARRAY type
5. Implement composition associations with the NESTED TABLE type
6. Define object type inheritance
7. Develop database applications using PL/SQL and Oracle tools
8. Deploy database applications on the Web with persistent data in ORDB

ORDB technology helps students to better understand object-oriented principles such as encapsulation, inheritance, and reusability. During the learning process, they have reviewed the object-oriented paradigm they learned from their previous programming courses and are able to tie it to ORDBMS and object-oriented system design. With a grasp of ORDB technology, students are able to make their database design more structured and utilize data in a more reusable way. With object reuse, standard adherence, and defined access paths, students are able to create a de facto standard for database objects and multiple database applications. In addition, they have learned to solve complex database problems with object-relational technology, to eliminate potential database complexity, and to improve database performance. The motivation to learn in class is high because students realize that object-relational technology is an important methodology that will help their career development in a competitive job market.

6. CONCLUSION

Using a UML class diagram for ORDB design has great significance in advanced system development. UML class diagrams are used to show the primary entities or objects in the system. They are the most important tools used in systems design. Event models such as a sequence, activity, or state chart diagrams are based on the class diagrams that illustrate the timing of various events and show how messages are passed between objects on the system. Eventually, ORDB design will become an essential component in object-oriented information systems.

The beauty of ORDBMSs is reusability and sharing. Reusability mainly comes from storing methods in object types and performing their functionality on the ORDBMS server, rather than have it coded in each application. Sharing comes from using user-defined standard data types to make database structure more standardized (Connolly & Begg, 2005). If multiple applications use the same set of database objects, then a de facto standard for the database objects has been created, and these objects can be extended (Price, 2002). Furthermore, ORDBMSs have advantages in solving non- 1NF problems in RDB and in speeding up query processes with collection types. To sum up, object-relational technology can be utilized to improve some weaknesses of RDB. UML class diagram is an appropriate tool to model the ORDB.

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. That might be so for those traditional database users who do not know object-oriented technology. However, the author found implementation of ORDBMSs at the IS senior level to be straightforward, because these students have already grasped the concept of relational databases and the object-oriented paradigms, and have learned either Java or C++.

7. REFERENCES

Connolly, T., and Begg C. 2005. Database systems: A practical approach to design, implementation, and management, 4th Ed. Addison Wesley

Chaudhri, A. And Zicari, R. 2000. Succeeding with object database: A practical look at today's implementations with Java and XML, Wiley.

Elmasri, R. & Navathe, S. 2003. Fundamentals of Database Systems, 4th Edition, Page 44, Addison Wesley.

Fortier, P. May 1999 SQL3: Implementing the Object-Relational Database, Osborne McGraw-Hill,

Garcia-Molina, H., Ullman, J. & Widom, J. 2003. Database Systems: The Complete Book, Prentice Hall, Upper Saddle River, NJ. Page 166.

Grorge, J., Batra, D., Valacich, J. and Hoffer, J., 2003. Object-oriented systems analysis and design, Prentice Hall, Upper Saddle River. Page 242.

Gorgone, J., Davis, G., Valacich, J., Topi, H., Feinsytein, D. and Longenecker, H., 2002. "IS 2002 model curriculum and guidelines for undergraduate degree programs in information systems, Association for Information Systems. URL: http://192.245.222.212:8009/IS2002Doc/Main_Frame.htm

Krishnamurthy, Banerjee and Nori, 1999. Bringing object-relational technology to the mainstream, Proceedings of the ACM SIGMOD International Conference on Management of Data and Symposium on Principles of Database Systems, May 31 - June 3, Philadelphia, PA

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

Price, J. 2002. Oracle9i, JDBC Programming, Oracle Press/McGraw-Hill/Osborne

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

Urban, S. and Dietrich, S. 2003, Using UML class diagrams for a comparative analysis of relational, object-oriented, and object-relational database mappings, SIGCSE Bulletin, Volume 33 Issue 1, p 21-25

Wang, M. 2001. Implementation of object-relational DBMSs in a relational database course: ACM SIGCSE Bulletin, Volume 33, Issue 1 February 2001, p 367-370

Wang, M. 2003. "E-business application development with java technology and oracle: The Fortune Invest Inc. Case", Journal of Information Systems Education, 14(3). 293-299.

AUTHOR BIOGRAPHY

Ming Wang teaches in the Department of Computer Information Systems at California State University in Los Angeles. She received her Ph. D. from Southern Illinois University in 1993 and taught previously at Embry-Riddle Aeronautical University in Daytona Beach, Florida. She has more than 20 publications in refereed journals and books, and more than a dozen of publications in international conference proceedings. Her current research interests are in ERP, e-commerce globalization, e-service quality, e-satisfaction, agent technology, and Information Systems education.

STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 1055-3096