Chapter 9
Case Studies: Relational, Object-Relational, and Object-Oriented Database Implementations

Suzanne W. Dietrich and Susan D. Urban
Arizona State University
Revised: December, 2004
CHAPTER LOG

Synopsis
This chapter provides an object-relational case study database implementation in Oracle of the School Database Enterprise. This case study reinforces the concepts covered in previous chapters and supplements the material covered in Chapters 3, 4, 7, and 8 with specific examples of database implementations using commercial database products. The case studies presented in this chapter provide the opportunity for a comparative analysis of the different database paradigms.

Assumed Knowledge
- Enhanced Entity Relationship (EER) Diagrams (Chapter 1)
- Unified Modeling Language (UML) Conceptual Class Diagrams (Chapter 2)
- The SQL Standard: Advanced Relational Features (Chapter 3)
- Mapping Object-Oriented Conceptual Models to the Relational Data Model (Chapter 4)
- Object-Oriented Databases and the ODMG Standard (Chapter 7)
- The SQL Standard: Object-Relational Features (Chapter 8)

Implementation Examples on Web (http://www.eas.asu.edu/~advdb)
- School Database Enterprise: Object-Relational Version
OUTLINE

• Oracle Overview
• Object Types
• Embedded Objects
• Column Objects
• Object Tables
• Object References
• Inheritance
• Varrays and Nested Tables
• Querying Objects, Varrays, and Nested Tables
• Maintaining Inverses
ORACLE OVERVIEW

Oracle supports object-relational extensions to the relational model:

- **Object Types** for creating user-defined types.
- **Object Tables** for creating objects from object types.
- **Type hierarchies** with inheritance and corresponding *table hierarchies*.
- **VARRAYS** and **Nested Tables** that allow a structured collection of data to be the type of a table column.
- **Refs** (or object references) that are used to store logical pointers to objects.
THE SCHOOL DATABASE EXAMPLE

OBJECT TYPES

Object types in Oracle (or abstract data types - ADTs) are abstractions of a real world object. Object types consist of two main components:

- **Attributes**
  - Contain built-in types or other object types as values.
  - Attributes model the structure of the real world entity.

- **Methods**
  - Methods are functions or procedures written in PL/SQL or an external language like Java and stored in the database.
  - Methods implement the operations that an application can perform on the real world entity.
OBJECT TYPE EXAMPLE

create or replace type person_t as object
    pId          varchar2(9),
    firstName    varchar2(20),
    lastName     varchar2(20),
    dob          date);

Assigning data to an object type:
declare
    p        person_t;
bEGIN
    p.pId := 'PR123456789';
    p.firstName := 'Terrence';
    p.lastName := 'Grand';
    p.dob := '10-NOV-1975';
end;
EMBEDDED OBJECTS

- When an object type is used as the type of an attribute in another object type, the value of the attribute is referred to as an *embedded object*.
- Embedded objects do not have object identifiers (OIDs) and accessed, queried, and manipulated the same as structured types in SQL.

```sql
create or replace type location_t as object (
  street      varchar2(30),
  bldg        varchar2(5),
  room        varchar2(5)
);
create or replace type campusClub_t as object
  cId           number,
  name       varchar2(50),
  phone      varchar2(25),
  location    location_t, -- embedded object
  ...
);```

```sql
```
COLUMN OBJECTS

• The SQL statement:

```
create table courseOffering(code int, loc location_t);
```

creates a relational table with two columns, where `loc` is the name of a column in the table `courseOffering` and `loc` is referred to as a column object.

• Inserting a tuple with a column object:

```
insert into courseOffering values (123, location_t('Orange Mall', 'CPCOM', '207'));
```
OBJECT TYPE WITH METHODS

create or replace type campusClub_t as object (
  cld number,
  name varchar2(50),
  phone varchar2(25),
  location location_t,
  advisor ref faculty_t,
  members student_ntable,
  member function isAssociatedMember
    (candidate_member in ref student_t) return boolean,
  member function isAssociatedAdvisor
    (candidate_advisor in ref faculty_t) return boolean,
  pragma restrict_references (default, wnds, wnps));
create or replace type body campusClub_t is

... 

member function isAssociatedAdvisor (candidate_advisor in ref faculty_t)
  return boolean is
    begin
      return (self.Advisor = candidate_advisor);
    end isAssociatedAdvisor;

end;
PRAGMAS

• Functions are not allowed to modify table data.
• The PRAGMA statement is a compiler directive used to declare that functions do not modify the database.
  o wnds – write no database state
  o rnds – read no database state
  o wnps – write no package state
  o rnps – read no package state
OBJECT TABLES

- As in SQL, object tables can be created based on object types.
- Each row in an object table is referred to as a *row object*.
- A row object has an OID referred to as a reference.
- Table and column constraints can be specified in the definition of an object table.
- OIDs can be system generated or based on the primary key of the object table.

```sql
create table person of person_t (  
  pId primary key,  
  firstName not null,  
  lastName not null,  
  dob not null  
object id primary key;
```
OBJECT REFERENCES

- Oracle supports the reference type (ref) as defined in SQL.
- A ref is a logical pointer to a row object.
- In the example below, the campusClub_t object type contains a ref to a faculty_t object.

```sql
create or replace type campusClub_t as object (
    cId number,
    name varchar2(50),
    phone varchar2(25),
    location location_t,
    advisor ref faculty_t, -- relationship Advised by
    members student_netable,
    ...)
```

CONSTRAINTS ON REFS
Unconstrained Refs

- A ref column or attribute can be unconstrained or constrained.
- If a ref is unconstrained, it may contain object references to row objects in any object table of the appropriate object type.
- Oracle does not ensure that object refs in unconstrained columns point to valid row objects, creating the potential for dangling references.
- Primary key based object references must be constrained.
CONSTRAINTS ON REFS

Constrained Refs

- An object ref can be constrained using a referential constraint.
- A referential constraint is used in an object table definition to constrain object refs to point to valid row objects in a specific object table.
- In the example below, the deptChair column must contain valid row objects from the faculty object table.

```sql
create or replace type department_t as object (  
    ...  
    deptChair ref faculty_t,  
    ... );
create table department of department_t  
(  ...  
  object id primary key;
alter table department add (  
    constraint department_chair foreign key ( deptChair ) references faculty  
    on delete set null);```
CONSTRAINTS ON REFS
The Scope Constraint

- Object refs can also be constrained using the scope clause to indicate the table that object refs should refer to.
- The scope clause must be used in a separate alter table statement.
- The scope clause is different from a referential constraint in that a scope clause does not protect against dangling references.

```sql
alter table campusClub add (
    scope for (advisor) is faculty);
```
QUERYING OBJECTS

- Queries involving objects must make a distinction between row objects, refs, and object types.
- Oracle provides three functions to support queries involving objects:
  - `ref()` if a function that takes a row object as its argument and returns the ref to that object.
  - `value()` is a function that takes a row object and returns an instance of the object type.
  - `deref()` is a function that takes a ref to an object and returns an instance of the object type.
REF, DEREF, AND VALUE EXAMPLE

Recall the campusClub_t object type and table definition:

```sql
create or replace type campusClub_t as object (  
  cId number,  
  name varchar2(50),  
  phone varchar2(25),  
  location location_t,  
  advisor ref faculty_t, -- relationship Advised by  
  members student_ntable, -- relationship Member of  
);  
create table campusClub of campusClub_t (  
  cId primary key,  
  name not null)  
object id primary key  
nested table members store as club_members;
```
QUERY USING REF, DEREFS, AND VALUE

```sql
declare
    club_ref ref campusClub_t;
    club campusClub_t;
    club_adv faculty_t;
begin
    select value(c), ref(c), deref(c.advisor) into club, club_ref, club_adv
    from campusClub c
    where c.name='The Hiking Club';
end;
```

- In the above example, the table alias `c` contains a row object.
- The object type of the row object is `campusClub_t`.
USING REF, DEREF, AND VALUE

- The get the ref to the row object, use ref(c).
- To see the values of the attributes of the row object as defined in the object type, use value(c).
- To get values of the attributes of the object type associated with a ref, use the deref() function, where the input to deref() must be a ref.
  - In the Campus Club example, c.advisor contains a ref to a faculty_t object.
- What is the value of club, club_ref, and club_adv after executing the select statement?
USING PATH EXPRESSIONS

- *Path expressions* can be used to traverse through object references in queries.
- The query below returns a string value representing the name of the department in which the advisor of Epsilon Sigma works.
- The expression `c.advisor.worksIn.name` is a path expression, representing implicit joins between the object tables involved.

```sql
select c.advisor.worksIn.name
from campusClub c
where c.name = 'Epsilon Sigma';
```
INHERITANCE

- Oracle supports the `UNDER` clause as defined in SQL for forming types/tables into hierarchies.
- Subtypes inherit attributes and methods from supertypes.
- Difference between Oracle and SQL: Objects in subtables are not visible in supertables. As a result, a query on a supertable will not return the contents of subtables.
- To query the contents of a table hierarchy, the user must explicitly join the supertable with its subtables.
- An alternative is to simulate inheritance.
SIMULATING INHERITANCE

- A class hierarchy such as the Person hierarchy in the School DB example can be implemented using a super attribute at the subclass level to relate a subclass to its superclass.
- Procedures and triggers can be used to maintain specialization constraints.
- Definition of the person_t superclass object type:

```sql
create or replace type person_t as object (
  pId       varchar2(9),
  firstName  varchar2(20),
  lastName   varchar2(20),
  dob        date
);
```
DEFINING OBJECT TYPES FOR SUBCLASSES

Relating the faculty_t and student_t object types to the person_t object type using the super attribute:

create or replace type faculty_t as object (  
   super ref person_t, --person super class  
   ...  
);  
/
create or replace type student_t as object (  
   super ref person_t, --person super class  
   ...  
);
TRIGGER FOR MAINTAINING TOTAL SUBCLASS PARTICIPATION

create or replace trigger faculty_rowdelete
  after delete on faculty
  for each row
  begin
    delete person p where ref(p) = :old.super;
  end faculty_rowdelete;
VARRAYS AND NESTED TABLES

• Variable sized arrays (varrays) and nested tables are collection types that can be used to represent the “many” side of 1:N and M:N relationships.

• Varrays and nested tables are similar in that the data types of all elements in each collection must be the same (i.e., each collection must be of a homogeneous type).

• Varrays and nested tables are different in the way they are internally managed by Oracle.
CHARACTERISTICS OF VARARRAYS

- Vararrays cannot be indexed and are normally used when the number of objects contained in a multi-valued attribute is small and does not change.
- A varray declaration must specify the maximum number of objects to hold.
- Vararrays are dense in that all positions from the first to last must be filled. Individual elements cannot be deleted from a varray.
- The elements of a varray are ordered.
- If the size of the varray is smaller than 4000 bytes, Oracle stores the varray in line; if it is greater than 4000 bytes, Oracle stores it as a Binary Large Object (BLOB).
USING VARRAYS

The 1:N relationship between Faculty and CampusClub is implemented using a varray on the Faculty side of the relationship.

create or replace type campusClub_t;
create or replace type campusClub_array as varray(50) of ref campusClub_t;
create or replace type faculty_t as object (  
  super ref person_t, -- person super class
  rank integer,
  advisorOf campusClub_array, -- relationship advisorOf
  worksIn ref department_t, -- relationship worksIn
  chairOf ref department_t,
  ...
);

CHARACTERISTICS OF NESTED TABLES

- If the number of objects in a multi-valued attribute is large, then a nested table should be used instead of a varray.
- A nested table is an unordered set of data elements, all of the same data type having a single column.
- The type of the column is either a built-in type or an object type.
- If an object type is used as the type of the column, the table can be viewed as a multi-column table, with a column for each attribute of the object type.
- Nested tables are stored in a separate storage table and can be indexed for efficient access.
- Nested tables are sparse (i.e., individual elements can be deleted from a nested table).
USING NESTED TABLES

In the School Database example, the collection of students that join a CampusClub is modeled as a nested table.

create or replace type student_ntable as table of ref student_t;

create or replace type campusClub_t as object ( 
    ... 
    members student_ntable, -- relationship Member of 
    ... 
);

create table campusClub of campusClub_t ( 
    clId primary key, 
    name not null) 
object id primary key 
nested table members store as club_members;
CONSTRAINTS ON NESTED TABLES

- The scope for clause can be used to restrict the originating table for the refs. The scope for constraint on a ref is not allowed in a create table statement.

- Example: To specify that the table column value for the advised_clubs nested table can reference only the object table student, an alter table statement must be used to add the constraint.
  
  ```sql
  alter table club_members add
  (scope for (column_value) is student);
  ```

- A unique index on the nested table advised_clubs, using pseudocolumns nested_table_id and column_value, restricts a student to be associated with a campusclub only once (prevents duplicate entries in the nested table)
  
  ```sql
  create unique index club_members_idx on
  club_members(nested_table_id, column_value);
  ```
### FUNCTIONS/PROCEDURES FOR ACCESSING COLLECTIONS

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection.count</td>
<td>Function: Returns the number of elements in the collection.</td>
</tr>
<tr>
<td>collection(i)</td>
<td>Function: Returns the element at location $i$ in the collection.</td>
</tr>
<tr>
<td>collection.first</td>
<td>Function: Returns the lowest-valued index of the collection containing an element (returns null if the collection is empty).</td>
</tr>
<tr>
<td>collection.last</td>
<td>Function: Returns the highest-valued index of the collection containing an element (returns null if the collection is empty).</td>
</tr>
<tr>
<td>collection.exists(n)</td>
<td>Function: Returns true if there is a row in the specified index of the collection.</td>
</tr>
<tr>
<td>collection.next(n)</td>
<td>Function: Returns the next higher-valued index of the collection where an element exists (otherwise returns null).</td>
</tr>
<tr>
<td>collection.prior(n)</td>
<td>Function: Returns the next lower-valued index of the collection where an element exists (otherwise returns null).</td>
</tr>
<tr>
<td>collection.extend</td>
<td>Procedure: Creates memory for storing an element at the end of the collection.</td>
</tr>
<tr>
<td>collection.trim</td>
<td>Procedure: Deletes the element at the end of the collection.</td>
</tr>
<tr>
<td>table.delete(n)</td>
<td>Procedure: Delete row $n$ of the nested table (Cannot be used with varrays).</td>
</tr>
</tbody>
</table>
DELETING AN ELEMENT FROM A VARRAY

- Since a varray is dense, changes occur to the contents of the varray by completely replacing the array with a new array.
- The example on the next page illustrates how to delete a club from the list of clubs that a person belongs to.
DELETING AN ELEMENT FROM A VARRAY

create or replace trigger campusClub_delete
  after delete on campusClub
declare
  deleted_cld number
  c_obj campusClub_t;
  f_obj faculty_t;
  temp_array campusClub_array;
  local_array campusClub_array;
... continued on next slide
begin
  ...
  select f_obj.AdvisorOf into local_array from dual;
  for i in 1..local_array.count loop
    select deref(local_array(i)) into c_obj from dual;
    if not (c_obj.cId = deleted_cId) then
      temp_array.extend;
      temp_array(temp_array.last):= local_array(i);
    end if;
  end loop;
  ...
end;
QUERYING VARRAYS

- Example: Print the names of students in a department:

```sql
create or replace procedure listDeptStudents (dept_code in varchar2) is
    stu_array  student_array;
    s_obj      student_t;
    p_obj      person_t;
begin
    ...
    select a.get_students( ) into  stu_array from department a where a.code = dept_code;
    ...
    for i in 1.. stu_array.count loop
        select deref(club_array(i)) into s_obj from dual;
        select deref(s_obj.super) into p_obj from dual;
        dbms_output.put_line(p_obj.pId || ' ' || p_obj.firstname || ' ' || p_obj.lastname);
    end loop;
    ...
end listDeptStudents;
```
INSERTING TO AND DELETING FROM NESTED TABLES

The `select` statement retrieves the nested table as the target of the `insert` statement.

```
insert into table
  (select c.members from campusClub c
   where c.name = club_name) values (s_ref);
```

The `get_sref` function returns the `ref` of the member to delete.

```
delete from table
  (select c.members from campusClub c where c.name = club_name) t
  where t.column_value = get_sref(member_to_delete);
```
QUERYING NESTED TABLES

- Nested tables can be queried directly with a select statement:

```sql
create or replace type body campusClub_t is
member function isAssociatedMember
  (candidate_member in ref student_t) return boolean is
  cnt integer;
begin
  select count(1) into cnt
  from table (cast(self.members as student_ntable)) t
  where t.column_value = candidate_member;
  return (cnt > 0);
end isAssociatedMember;

...
end;
```
MAINTAINING INVERSES

• Bidirectional Relationships
  o Use triggers on both sides of the association.
  o Constrain updates to one side with a trigger to update the other side.
  o Use a procedure to update both sides.
  o Beware of mutating tables in Oracle (cannot get the ref of a new or updated object within the body of a row trigger)!

• Unidirectional Relationships
  o Store the relationship on one side and use a function to derive the value on the other side.