Chapter 2. Object-Relational Database Systems

Motivation Examples

- give me the marketing campaigns that used images of sunny beaches with white sand.
- How many programmers with skills in SQL and Objects are working on the most profitable products?
- find the geographic location of Sacramento in the landmarks table.
- tell me the sale regions in which my top 5 products has a sales drop of more than 10%.

Acknowledgment: Some examples and definitions are from a tutorials by N.M. Mattos at ACM SIGMOD'97 and by J. Ullman

Going beyond the traditional

- increasing need for EXTENSIBILITY
  - new data types
    - text, images, audio, video, spatial, currencies, ...
  - integrated content-based search
    - combine text, image, spatial search, etc, in a single query
  - complex data relationships
    - hierarchies, bill of materials, travel planning
  - complex business rules
- Highly scalable performance – more than ever
  - extensible optimizer
  - core parallel, robust architecture

A relational database system extended with

- user-defined abstract data types,
- complex objects with inheritance

can be reviewed as an object-relational database system.
(DBMSs that support a dialect of SQL 2003, including non-traditional tools, and optimize for complex SQL99/ SQL2003 queries are called *Object-Relational* DBMSs.)

- they are relational in nature because they support SQL;
- they are object-oriented in nature because they support complex data.

They are a marriage of the SQL from the relational world and the modeling primitives from the object world.

2.1 A DBMS Classification Matrix

	Simple Data	Complex Data
Query	2	4
No Query	1	3

A matrix for classifying DBMS applications

[1] Model and Language Extensions of ORDBS

- mechanism for users to extend the database with application specific types and procedures (methods/functions).
  - user defined types
  - user defined procedures
- support for storage and manipulation of large data types
  - Large Object Support (BLOB)
- mechanism to improve the DB integrity, to allow checking of business rules, and to express complex data relationships inside the DBMS
  - triggers and constraints

[2] Benefits

- New functionality
  - indefinitely increase the set of types and procedures provided by the DBMS
- Simplified application development
  - code re-use
  - overloading
- Consistency
  - enables definition of standard, reusable code shared by all applications
- Easier application maintenance
- Performance

2.2 Characteristics of O-R Database Systems

[1] Primitive User-Defined Types

Distinct Types

- the most basic form of user-defined types
- based on name equivalence
- a rename type, usually with different behavior than its source type
- system-provided operators
- casting
- no inheritance or subtyping

Example 1

```
CREATE DISTINCT TYPE cnd_dollar as decimal(9, 2)
CREATE DISTINCT TYPE  us_dollar as decimal(9, 2)

CREATE TABLE cnd_sale           CREATE TABLE us_sale
  ( cust_no      integer,         ( cust_no     integer,
    contract_no  integer,           contract_no integer,
    total         cnd_dollar)        total        us_dollar)

SELECT cust_no, contract_no
FROM  cnd_sales, us_sales
WHERE cnd_sale.cust_no = us_sale.cust_no AND
      cnd_sake.total > cnd_dollar(us_sale.total)
```

[2] Complex User-defined Types

- Abstract data types
  - named, user-defined data type with *behavior* and an *encapsulated* internal structure (attributes)
  - separation of the interface of the type from its implementation
  - internal structure can be defined in (or outside of) SQL terms
- Encapsulation (ADTs are completely encapsulated)
  - access to ADT attributes is restricted to methods
  - no distinction between stored attributes and methods
  - observers and mutators can be either defined by the type definer or generated automatically
  - created by constructor
- Subtyping and inheritance
  - an ADT can be a subtype of other ADTs (single or multi inheritance)
- Use of ADT
  - ADTs can be used wherever other types can be used in SQL

Example 2

```
CREATE TYPE address
  ( street    char(30),
    city      char(20),
    province  char(3),
    zip       char(7) )

CREATE TYPE shape ...
CREATE TYPE point under shape ...
CREATE TYPE polygon under shape ...

CREATE TABLE real_estate_info
  ( address address,
    price cnd_dollars,
    owner char(40),
    property shape )
```

Example 2 (continued)

```
INSERT INTO  read_estate_info values (
   address(' ... ' ),
   us_dollar(30000),
   'John Doe',
   spatial(10, 12, 15, 20) )

SELECT us_dollar(price), owner
FROM   real_estate_inf
WHERE  overlaps (property.shape, square(5, 5, 25, 25))

SELECT owner, dollar_amount(price)
FROM read_estate_info
WHERE dollar_amount(price) < us_dollar(50000)
```

[3] Structured (un-encapsulated) User-defined Types

- row type with a name
  - user-defined data type with an non-encapsulated internal structure
  - used to define the types of rows in tables

Example 3 `CREATE ROW TYPE customer_t`

```
  ( c_id        int,
    last_name   varchar(36),
    first_name  varchar(30),
    address     address,
    phone       varchar(10)
  );

CREATE TABLE customer OF customer_t (
    PRIMARY KEY c_id )
```

- Reference types
  - used to refer to a row type (or, an object type)
  - reference can be scoped
  - reference values never changes as long as the corresponding tuples lives
  - reference can be used to write path expressions
  - do not have the same semantics as referential constraints

```
CREATE TABLE  account (
   acct_no int,
   cust    REF (customer_t),
   type    char(1),
   opened   date,
   rate    float,
   balance float,
   PRIMARY KEY acct_no
);
```

How to use a reference type

Example 4
```
SELECT a.acct_no, a.cust -> name
FROM   account a
WHERE  a.cut->address.city = 'Hollywood' and
       a.balance > 1000000
```

Example 5

```
CREATE ROW TYPE employee (
  id          integer,
  name        varchar(20),
  address     address,
  manager     REF(employee)
  projects    SET (REF(project))
  children    LIST (REF(person))
  hobbies     SET(VARCHAR(20)) )



SELECT  e.name
FROM    employee e
WHERE   'travel' IN ( SELECT *
                      FROM TABLE(e.hobbies) )
        and 5 > ( SELECT count(*)
                  FROM TABLE(e.children) )
```

- Large object data type
  - BLOB (Binary Large OBject)
  - CLOB (Character Large OBject)
  - operation supported for LOBs
  - LOB locations
- Type Constructors (collection types)
  - SETs, LISTs, BAGs, and ARRAYs
  - elements of type constructors can be any other type (including row types, ADTs, and other collection types)
  - can be used where predefined data type can be used
  - instances of collection types can be treated as tables for the purpose of queries

[4] Method

- user-defined functions and /or stored procedures defined on user-defined types
- can be generic (multi-methods) or selfish
- can be written in SQL's procedural extensions, 3GL, or 4GLs

Example 6

```
CREATE FUNCTION raise ( r REF(employee))
  RETURNS Decimal(8, 2),
  LANGUAGE SQL
  ...
  RETURN
    CASE r->status_parm
      WHEN 'EXEMPT' THEN
        (SELECT raise FROM exempt_plan
         WHERE job = r_.job and rating = r_.rating)
      WHEN 'NONEXEMPT' THEN
        (SELECT raise FROM nonexempt_plan
         WHERE job = r_.job and rating = r_.rating)
      ELSE raise_error('70001', 'Bogus job or rating')
    END
```

[5] Triggers

- rules within the database

  (active database rules)

```
CREATE TRIGGERS update_balance
  BEFORE INSERT On account_history
  REFERENCING NEW as ta
  FOR EACH ROW
  WHEN (ta.TA_type = 'W')
  UPDATE accounts
    SET balance = balance - ta.amount
    WHERE account_no = ta.account_no
```

2.3 O-R features in Current DBMS Productions

- Oracle 8-10G
  - Oracle Type System (OTS) object types
    - structure plus associated methods (C++ like)
    - REF types and collections (arrays and nested tables)
    - can appear as row of table or column table)
  - User-defined functions on any type
    - PL/SQL, or external language (C, Java, etc.)
  - Client-side object support
  - LOBs, triggers, and integrity constraints

- IBM's DB2 for Common Servers (V2)
  - user-defined column types
  - user-defined functions
  - LOBS,
  - triggers, and
  - integrity constraints
- Illustra from Informix
  - Ingres $--->$ Postgres $--->$ Illustra $--->$ Informix

2.4 Challenge on the Road to O-R Success

- O-R server challenges
  - greater functionality and performance
  - mixing O-R with parallel DB technologies
  - Broader O-R systems issues
    - client-side architecture and integration
    - legacy data access (traditional and otherwise)
  - Standard issues
    - SQL99, SQL 2003
    - other key interface

2.5 Oracle 9i's Object Relational Features

- user defined data types
- constructing object values
- user defined objects
- reference types
- nested relations
- type inheritance
- Java support for Oracle objects
- external procedures

IMPORTANT NOTES:

All object relational features of Oracle are implemented using relational technologies. That is, they are just an interface between users and the transitional relational system of oracle.

[1] Create and use of user defined types

```
CREATE TYPE PointType AS OBJECT (
    x NUMBER,
    y NUMBER
);



CREATE TYPE LineType AS OBJECT (
    end1 PointType,
    end2 PointType
);

CREATE TABLE Lines (
    lineID INT,
    line   LineType
);
```

```
     INSERT INTO Lines
         VALUES(27, LineType(
                         PointType(0.0, 0.0),
                         PointType(3.0, 4.0)
                    )
         );


SQL> select * from lines;

    LINEID      LINE(END1(X, Y), END2(X, Y))
---------------------------------------------------
        27      LINETYPE(POINTTYPE(0, 0), POINTTYPE(3, 4))
```

```
CREATE TYPE person AS OBJECT (
  name        VARCHAR2(30),
  phone       VARCHAR2(20) );

CREATE TABLE person_table OF person


INSERT INTO person_table VALUES (
       'John Smith',
       '1-800-555-1212' );
```

```
SELECT * FROM person_table p
        WHERE p.name = "John Smith";

NAME          PHONE
-----------------------------
John Smith    1-800-555-1212

SELECT VALUE(p) FROM person_table p
        WHERE p.name = "John Smith";

VALUE(P)
-----------------------------
oracle.sql.STRUCT@a2cafe2d
```

[2] User Defined Functions (procedures, methods)

Methods are functions or procedures that you can declare in an object type definition to implement behavior that you want objects of that type to perform. A principal use of methods is to provide access to an object's data. You can define methods for operations that an application is likely to want to perform on the data so that the application does not have to code these operations itself. To perform the operation, an application calls the appropriate method on the appropriate object.

- **CERATE TYPE** statement

  A type declaration can also include methods that are defined on values of that type, using

  - MEMBER FUNCTION
  - MEMBER PROCEDURE

  in the CREATE TYPE statement.

- **CREATE TYPE BODY** statement

  The code for the function itself (the definition of the method) is in a separate CREATE TYPE BODY statement.

```
CREATE TYPE LineType AS OBJECT (
         end1 PointType,
         end2 PointType,
         MEMBER FUNCTION length(scale IN NUMBER) RETURN NUMBER,
         PRAGMA RESTRICT_REFERENCES(length, WNDS)
     );


CREATE TYPE BODY LineType AS
  MEMBER FUNCTION length(scale NUMBER) RETURN NUMBER IS
    BEGIN
      RETURN scale *
       SQRT((SELF.end1.x-SELF.end2.x)*(SELF.end1.x-SELF.end2.x)+
             (SELF.end1.y-SELF.end2.y)*(SELF.end1.y-SELF.end2.y)
            );
    END;
END;
```

[3] Reference Types

- A REF is a built-in, logical “pointer” to a row object.
- A REF may be scoped, i.e., constrained to a specified object table.
- Dangling REFs: the object identified by a REF to become unavailable
- One can obtain a REF to a row object by selecting the object from its object table.

```
CREATE TABLE Lines2 (
   end1 REF PointType,
   end2 REF PointType
);

CREATE TABLE Points OF PointType;

INSERT INTO Lines2
   SELECT REF(pp), REF(qq)
   FROM Points pp, Points qq
   WHERE pp.x < qq.x;
```

```
CREATE TABLE people (
  id             NUMBER(4)
  name_obj       name_objtyp,
  address_ref    REF address_objtyp SCOPE IS address_objtab,
  phones_ntab    phone_ntabtyp)
  NESTED TABLE   phones_ntab STORE AS phone_store_ntab2 ;

--------------------------------------------------------------

DECLARE OrderRef REF to purchase_order;

SELECT REF(po) INTO OrderRef
               FROM purchase_order_table po
               WHERE po.id = 1000376;
```

There are several important prohibitions, where you might imagine you could arrange for a reference to an object,but you cannot.

- The points referred to must be tuples of a relation of type PointType, such as Points above.

  They cannot be objects appearing in some column of another relation.

- It is not permissible to invent an object outside of any relation and try to make a reference to it.

  For instance, we could not insert into Lines2 a tuple with contrived references such as VALUES(REF(PointType(1,2)), REF(PointType(3,4))), even though the types of things are right. The problem is that the points such as PointType(1,2) don't "live" in any relation.

[4] Nested Relations

A more powerful use of object types in Oracle is the fact that the type of a column can be a table-type. That is, the value of an attribute in one tuple can be an entire relation, as suggested by the picture below, where a relation with schema (a,b) has b-values that are relations with schema (x,y,z).

```
CREATE TYPE PolygonType AS TABLE OF PointType;

CREATE TABLE Polygons (
     name   VARCHAR2(20),
     points PolygonType)
     NESTED TABLE points STORE AS PointsTable;

INSERT INTO Polygons VALUES(
     'square',
     PolygonType(PointType(0.0, 0.0), PointType(0.0, 1.0),
          PointType(1.0, 0.0), PointType(1.0, 1.0)
          )
);
```

```
SELECT points
     FROM Polygons
     WHERE name = 'square';



SELECT ss.x
     FROM THE(SELECT points
              FROM Polygons
              WHERE name = 'square'
             ) ss
     WHERE ss.x = ss.y;

X
-----------------
0
1
```

[5] Type Inheritance

A type hierarchy is a sort of family tree of object types, under which subtypes automatically acquire all the attributes and methods of their parent types.

```
 CREATE TYPE Person_type AS OBJECT (
   ssn NUMBER,
   name VARCHAR(30),
   address VARCHAR(100 )
 ) NOT FINAL;

CREATE TYPE Student_type UNDER Person_type (
   deptid NUMBER,
   major VARCHAR(30)
)  NOT FINAL;
```

```
CREATE TYPE Employee_type UNDER Person_type (
  empid NUMBER,
  mgr VARCHAR(30)
);

CREATE TYPE PartTimeStudent_type UNDER Student_type (
 numhours NUMBER
);
```