1 ODBMS vs RDBMS

Relational databases

Relational database

is based on the relational model of data, as proposed by E.F. Codd in 1970. This model organizes data into one or more tables (or “relations”) of rows and columns, with a unique key for each row.

A relation is defined as a set of tuples that have the same attributes. A tuple usually represents an object and information about that object. Objects are typically physical objects or concepts. A relation is usually described as a table, which is organized into rows and columns.

All the data referenced by an attribute are in the same domain and conform to the same constraints.

Advantages of Relational Model

- intuitive model
- ease transition from file systems
- use of multiprocessing
- multiplatform
- parallel queries
- advanced optimization methods
- advanced security methods
- extensive system of transactions

Disadvantages of relational model

- weak set of types
- lack of user’s functions and procedures
- problems with hierarchy
- no nested data
- lack of inheritance
- a mismatch between the language of data manipulation and application programming languages - impedance mismatch
- lack of time aspect and versioning

Mismatch

Object-oriented databases

**Object database**

is a database management system in which information is represented in the form of objects as used in object-oriented programming.

Object-oriented databases allow object-oriented programmers to develop the product, store them as objects, and replicate or modify existing objects to make new objects within the OODBMS. Because the database is integrated with the programming language, the programmer can maintain consistency within one environment, in that both the OODBMS and the programming language will use the same model of representation.

Relational DBMS projects, by way of contrast, maintain a clearer division between the database model and the application. Examples: Caché, ConceptBase, Db4o, GemStone/S, NeoDatis ODB, ObjectDatabase++, ObjectDB, Objectivity/DB, ObjectStore, Perst, VelocityDB, WakandaDB, ZooDB.

Advantages
• Objects don’t require assembly and disassembly saving coding time and execution
time to assemble or disassemble objects.
• Reduced paging
• Easier navigation
• Better concurrency control - A hierarchy of objects may be locked.
• Data model is based on the real world.
• Works well for distributed architectures.
• Less code required when applications are object oriented.

Disadvantages

• Lower efficiency when data is simple and relationships are simple.
• Relational tables are simpler.
• Late binding may slow access speed.
• More user tools exist for RDBMS.
• Standards for RDBMS are more stable.
• Support for RDBMS is more certain and change is less likely to be required.

Applied Solutions

• Object-oriented database
• Post-relational databases - Relational database extended with:
  – object-oriented features,
  – support for XML,
  – analytical solutions,
  – historical queries, etc ....

2 Object-Relational Model

Object-Relational Databases ORDBMS

ORDBMS

Databases created in this model are often called post-relational, hybrid or extended relational databases.

They provide:

• collections (sets, multiset, sequences, nested tables, arrays of variable length, tables
  of objects)

• user methods (functions and procedures defined by the user in various languages C
  ++, Visual Basic, Java, PL / SQL),

• multimedia and spatial data,
• object features (classes, methods, inheritance, aggregation, overloading)
• object views
• hierarchies of data sets,
• SQL3 query language also called ObjectSQL.

From the relational model they leave:
• client / server architecture,
• caching and indexing mechanisms,
• transaction processing,
• optimization of queries.

SQL3
SQL language standard created in 1999, expanded by:
• new data types: BLOB, CLOB (no possibilities for UNIQUE constraint, they can not be use as an argument of ORDER BY or GROUP BY), BOOLEAN, ARRAY, ROW
• broadening the scope of views through which you can update the data
• the introduction of recursive queries
• Savepoints in transactions
• the introduction of roles for users
• expansion of triggers

and the use of object-oriented features of post-relational model:
• structured user-defined types (with methods, inheritance, subtypes)
• functions and procedures written in other languages
• object tables, nested tables, varrays
• reference types - references to objects, unique and non-modifiable

The Role of Abstraction
• An abstraction is a high-level description or model of a real-world entity.
• Abstractions keep our daily lives manageable by suppressing irrelevant detail.
• Abstractions are central to the discipline of programming.
• Object types are a generalization of the built-in datatypes found in most programming languages.
2.1 Object Types

Object Type

An object type
is a user-defined composite datatype that encapsulates a data structure along with the functions and procedures needed to manipulate the data. The variables that form the data structure are called attributes. The functions and procedures that characterize the behavior of the object type are called methods.

Objects offer other advantages over a purely relational approach, such as:

- Objects Can Encapsulate Operations Along with Data
- Objects Are Efficient
- Objects Can Represent Part-Whole Relationships

Creating Object - Example

CREATE TYPE person_typ AS OBJECT (
  idno NUMBER,
  first_name VARCHAR2(20),
  last_name VARCHAR2(25),
  email VARCHAR2(25),
  phone VARCHAR2(20),
  MAP MEMBER FUNCTION get_idno RETURN NUMBER,
  MEMBER PROCEDURE display_details
    ( SELF IN OUT NOCOPY person_typ )
);
/

CREATE TYPE BODY person_typ AS
  MAP MEMBER FUNCTION get_idno RETURN NUMBER IS
    BEGIN
      RETURN idno;
    END;
  MEMBER PROCEDURE display_details
    ( SELF IN OUT NOCOPY person_typ ) IS
BEGIN
   DBMS_OUTPUT.PUT_LINE(TO_CHAR(idno) || ' ' ||
   first_name || ' ' ||
   last_name);
   DBMS_OUTPUT.PUT_LINE(email || ' ' || phone);
END;
END;
/

Object Structure

• Attributes hold the data about an object. Attributes have declared data types which can, in turn, be other object types.

• Methods are procedures or functions that applications can use to perform operations on the attributes of the object type. Methods are optional. They define the behavior of objects of that type.

Object Instance

• A variable of an object type is an instance of the type.

• Defining an object type does not allocate any storage. After they are defined, object types can be used in SQL statements.

• Storage is allocated once you create an instance of the object.

CREATE TABLE contacts (  
  contact person_typ,  
  contact_date DATE
);

INSERT INTO contacts VALUES (  
  person_typ (65, 'Verna', 'Mills',  
  'vmills@example.com', '1-650-555-0125'),  
  '24 Jun 2003' );

Object Methods

Object 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.

We distinguish:

• Member Methods

• Static Methods

• Constructor Methods

SELECT c.contact.get_idno() FROM contacts c;
Member Methods

Member methods provide an application with access to an object instance’s data. You define a member method in the object type for each operation that you want an object of that type to be able to perform. Non-comparison member methods are declared as either MEMBER FUNCTION or MEMBER PROCEDURE. Comparison methods use MAP MEMBER FUNCTION or ORDER MEMBER FUNCTION.

Parameter SELF

- MEMBER methods accept a built-in parameter named SELF, which is an instance of the object type.
- SELF is always the first parameter passed to the method.
- STATIC methods cannot accept or reference SELF.
- In member functions, if SELF is not declared, its parameter mode defaults to IN.
- In member procedures, if SELF is not declared, its parameter mode defaults to IN OUT. The default behavior does not include the NOCOPY compiler hint.
- You cannot specify the OUT parameter mode for SELF.
- You cannot specify a different datatype for SELF.

Member Methods for Comparing Objects

Map Methods
return values that can be used for comparing and sorting. Return values can be any Oracle built-in data types (except LOBs and BFILEs) and ANSI SQL types such as CHARACTER or REAL.

CREATE OR REPLACE TYPE rectangle_typ AS OBJECT (
    len NUMBER,
    wid NUMBER,
    MAP MEMBER FUNCTION area RETURN NUMBER
);
/
CREATE OR REPLACE TYPE BODY rectangle_typ AS
    MAP MEMBER FUNCTION area RETURN NUMBER IS
    BEGIN
        RETURN len * wid;
    END area;
END;
/

Member Methods for Comparing Objects

Order Methods
make direct one-to-one object comparisons. Unlike map methods, they cannot determine the order of a number of objects. They simply tell you that the current object is less than, equal to, or greater than the object that it is being compared to, based on the criterion used.
CREATE OR REPLACE TYPE location_typ AS OBJECT (
  building_no NUMBER,
  city VARCHAR2(40),
  ORDER MEMBER FUNCTION match (l location_typ)
    RETURN INTEGER);
/

CREATE OR REPLACE TYPE BODY location_typ AS
  ORDER MEMBER FUNCTION match (l location_typ)
    RETURN INTEGER IS
    BEGIN
      IF building_no < l.building_no THEN
        RETURN -1; -- any negative number will do
      ELSIF building_no > l.building_no THEN
        RETURN 1; -- any positive number will do
      ELSE
        RETURN 0;
      END IF;
    END;
END;
/

-- invoking match method
DECLARE
  loc location_typ;
  secloc location_typ;
  a number;
BEGIN
  loc := NEW location_typ(300, 'San Francisco');
  secloc := NEW location_typ(200, 'Redwood Shores');
  a := loc.match(secloc);
  DBMS_OUTPUT.PUT_LINE('order (1 is greater, -1 is lesser):'
            || a); -- prints order: 1
END;

Static Methods

Static methods are invoked on the object type, not its instances. You use a static method for operations that are global to the type and do not need to reference the data of a particular object instance. A static method has no SELF parameter.

- Static methods are declared using STATIC FUNCTION or STATIC PROCEDURE.
- You invoke a static method by using dot notation to qualify the method call with the name of the object type, for example:

  type_name.method()

Constructor Methods

Constructor method
is a function that returns a new instance of the user-defined type and sets up the values of its attributes. Constructor methods are either system-defined or user-defined.

- To invoke a constructor, the keyword NEW can be used, but is not required.
- By default, the system implicitly defines a constructor function for all object types that have attributes. This constructor is sometimes known as the attribute value constructor.

```sql
person_typ (1, 'John Smith', '1-650-555-0135'),
```

### 2.2 Using Objects

**Storing Objects in Tables**

Objects can be stored in two types of tables:

- Object tables: store only objects
- Relational tables: store objects with other table data

**Object Tables**

```sql
CREATE TABLE person_obj_table OF person_typ;
```

```sql
INSERT INTO person_obj_table VALUES ( person_typ(101, 'John', 'Smith', 'jsmith@example.com', '1-650-555-0135') );
```

```sql
SELECT VALUE(p) FROM person_obj_table p
WHERE p.last_name = 'Smith';
```

```sql
DECLARE
    person person_typ;
BEGIN
    SELECT VALUE(p) INTO person
    FROM person_obj_table p WHERE p.idno = 101;
    person.display_details();
END;
/
```

**Using Object Identifiers**

Object identifiers (OIDs) uniquely identify row objects in object tables. You cannot directly access object identifiers, but you can make references (REFs) to the object identifiers and directly access the REFs. There are two types of object identifiers:

- System-Generated Object Identifiers (default)
- Primary-Key Based Object Identifiers
Using References to Row Objects

- A REF is a logical pointer or reference to a row object that you can construct from an object identifier (OID). You can use the REF to obtain, examine, or update the object. You can change a REF so that it points to a different object of the same object type hierarchy or assign it a null value.

- REFs are Oracle Database built-in data types. REFs and collections of REFs model associations among objects, particularly many-to-one relationships, thus reducing the need for foreign keys. REFs provide an easy mechanism for navigating between objects.

Using References - Example

```sql
CREATE TYPE emp_person_typ AS OBJECT (
    name VARCHAR2(30),
    manager REF emp_person_typ
);
/

CREATE TABLE emp_person_obj_table OF emp_person_typ;

INSERT INTO emp_person_obj_table VALUES (
    emp_person_typ ('John Smith', NULL));

INSERT INTO emp_person_obj_table
    SELECT emp_person_typ ('Bob Jones', REF(e))
    FROM emp_person_obj_table e
    WHERE e.name = 'John Smith';
```

Rules for References

- In Oracle Database, a REF column or attribute can be unconstrained or constrained using a SCOPE clause or a referential constraint clause. When a REF column is unconstrained, it may store object references to row objects contained in any object table of the corresponding object type.

- A REF column may be constrained with a REFERENTIAL constraint similar to the specification for foreign keys. The rules for referential constraints apply to such columns. That is, the object reference stored in these columns must point to a valid and existing row object in the specified object table.

- PRIMARY KEY constraints cannot be specified for REF columns.

- You can specify NOT NULL constraints for such columns.

Using Scoped References

You can constrain a column type, collection element, or object type attribute to reference a specified object table by using the SQL constraint subclause SCOPE IS when you declare the REF. Scoped REF types require less storage space and allow more efficient access than unscoped REF types.
CREATE TABLE contacts_ref  
  (contact_ref REF person_typ
      SCOPE IS person_obj_table,
  contact_date DATE );

INSERT INTO contacts_ref
  SELECT REF(p), '26 Jun 2003'
  FROM person_obj_table p
  WHERE p.idno = 101;

Dangling References

- It is possible for the object identified by a REF to become unavailable if the object has been deleted or some necessary privilege has been deleted. This is a dangling REF. You can use the Oracle Database SQL predicate IS DANGLING to test REFs for this condition.

- Dangling REFs can be avoided by defining referential integrity constraints.

Dereferencing References

Accessing the object that the REF refers to is called dereferencing the REF. Oracle Database provides the DEREF operator to do this.

SELECT DEREF(e.manager) FROM emp_person_obj_table e;
--DEREF(E.MANAGER)(NAME, MANAGER)
--EMP_PERSON_TYP('John Smith', NULL)

DELETE from person_obj_table WHERE idno = 101;
/
SELECT DEREF(c.contact_ref), c.contact_date
  FROM contacts_ref c;

SELECT e.name, e.manager.name
  FROM emp_person_obj_table e
  WHERE e.name = 'Bob Jones';

2.3 Inheritance

Advantages of Type Inheritance

- Enables to create type hierarchies.

- Derived subtypes inherit the features of the parent object type and can extend the parent type definition.

- The specialized types can add new attributes or methods, or redefine methods inherited from the parent.

- The resulting type hierarchy provides a higher level of abstraction for managing the complexity of an application model.
Type Inheritance

- SQL object inheritance is based on a family tree of object types.
- The type hierarchy consists of a parent object type, called a supertype, and one or more levels of child object types, called subtypes, which are derived from the parent.
- Subtypes automatically inherit the attributes and methods of their parent type.
- Oracle only supports single inheritance. Therefore, a subtype can derive directly from only one supertype, not more than one.

Type Hierarchy

Subtypes are created using the keyword UNDER as follows:

CREATE TYPE student_typ UNDER person_typ

Specialization

- Add new attributes that its parent supertype does not have.
- Add entirely new methods that the parent does not have.
- Change the implementation of some of the methods that a subtype inherits so that the subtype’s version executes different code from the parent’s.

FINAL vs NOT FINAL

For a method, the definition must indicate whether or not it can be overridden. The keywords FINAL or NOT FINAL are used for both types and methods.

- For a type FINAL, (default) means that no subtypes can be derived from it.
- NOT FINAL means subtypes can be derived.
- For a method, FINAL means that subtypes cannot override it by providing their own implementation.
- NOT FINAL (default) means that you can override the method of the supertype.
Example

CREATE OR REPLACE TYPE person_typ AS OBJECT (  
idno NUMBER,  
name VARCHAR2(30),  
phone VARCHAR2(20),  
FINAL MAP MEMBER FUNCTION get_idno RETURN NUMBER
) NOT FINAL;
/

ALTER TYPE person_typ FINAL;

Example 2 Creating the Parent

CREATE OR REPLACE TYPE person_typ AS OBJECT (  
idno NUMBER,  
name VARCHAR2(30),  
phone VARCHAR2(20),  
MAP MEMBER FUNCTION get_idno RETURN NUMBER,  
MEMBER FUNCTION show RETURN VARCHAR2
) NOT FINAL;
/

CREATE OR REPLACE TYPE BODY person_typ AS  
MAP MEMBER FUNCTION get_idno RETURN NUMBER IS  
BEGIN  
RETURN idno;
END;
MEMBER FUNCTION show RETURN VARCHAR2 IS  
BEGIN  
RETURN 'Id: ' || TO_CHAR(idno) ||  
', Name: ' || name;
END;
END;
/

Example 2 Creating the Child

A subtype inherits the following:

- All the attributes declared in or inherited by the supertype.
- Any methods declared in or inherited by supertype.

CREATE TYPE student_typ UNDER person_typ (  
department NUMBER,  
major VARCHAR2(30),  
OVERRIDING MEMBER FUNCTION show RETURN VARCHAR2
) NOT FINAL;
/

CREATE TYPE BODY student_typ AS  
OVERRIDING MEMBER FUNCTION show RETURN VARCHAR2 IS  
BEGIN  
RETURN (self AS person_typ).show ||  
' -- Major: ' || major ;
END;
END;
/

Example 2 Creating the second Child

CREATE OR REPLACE TYPE employee_typ UNDER person_typ (  
emp_id NUMBER,  
mgr VARCHAR2(30),  
OVERRIDING MEMBER FUNCTION show RETURN VARCHAR2
);
/

CREATE OR REPLACE TYPE BODY employee_typ AS  
OVERRIDING MEMBER FUNCTION show RETURN VARCHAR2 IS  
BEGIN  
RETURN (SELF AS person_typ).show||  
' -- Employee Id: '||  
TO_CHAR(emp_id) ||',  
Manager: '|| mgr ;
END;
END;
/
Example 2 Further Levels

CREATE TYPE part_time_student_typ UNDER student_typ (
    number_hours NUMBER,
    OVERRIDDING MEMBER FUNCTION show RETURN VARCHAR2);
/

CREATE TYPE BODY part_time_student_typ AS
    OVERRIDDING MEMBER FUNCTION show RETURN VARCHAR2 IS
    BEGIN
        RETURN (SELF AS person_typ).show||
            ' -- Major: ' || major ||
            ', Hours: ' || TO_CHAR(number_hours);
    END;
END;
/

Example 2 Inserting Data

INSERT INTO person_obj_table
    VALUES (person_typ(12, 'Bob Jones', '650-555-0130'));
INSERT INTO person_obj_table
    VALUES (student_typ(51, 'Joe Lane', '1-650-555-0140', 12, 'HISTORY'));
INSERT INTO person_obj_table
    VALUES (employee_typ(55, 'Jane Smith', '1-650-555-0144', 100, 'Jennifer Nelson'));
INSERT INTO person_obj_table
    VALUES (part_time_student_typ(52, 'Kim Patel', '1-650-555-0135', 14, 'PHYSICS', 20));
SELECT p.show() FROM person_obj_table p;
--The output is similar to:
--Id: 12, Name: Bob Jones
--Id: 51, Name: Joe Lane -- Major: HISTORY
--Id: 55, Name: Jane Smith -- Employee Id: 100, Manager: Jennifer Nelson
--Id: 52, Name: Kim Patel -- Major: PHYSICS, Hours: 20

NOT INSTANTIABLE Types and Methods

- NOT INSTANTIABLE Types - you cannot instantiate instances of that type. There are no constructors (default or user-defined) for it. You might use this with types intended to serve solely as supertypes from which specialized subtypes are instantiated.

- NOT INSTANTIABLE Methods - serves as a placeholder. It is declared but not implemented in the type. You might define a non-instantiable method when you expect every subtype to override the method in a different way. In this case, there is no point in defining the method in the supertype.

A type that contains a non-instantiable method must itself be declared not instantiable.

Example 3

CREATE OR REPLACE TYPE person_typ AS OBJECT (
    idno NUMBER,
    name VARCHAR2(30),
    phone VARCHAR2(20),
    NOT INSTANTIABLE MEMBER FUNCTION get_idno
        RETURN NUMBER
) NOT INSTANTIABLE NOT FINAL;
/

--ALTER TYPE person_typ INSTANTIABLE;
Overloading and Overriding Methods

- Adding new methods that have the same names as inherited methods to the sub-type is called overloading. When they exist in the same user-defined type, methods that have the same name, but different signatures are called overloads. A method signature consists of the method’s name and the number, types, and the order of the method’s formal parameters, including the implicit self parameter.

- Redefining an inherited method to customize its behavior in a subtype is called overriding, in the case of member methods, or hiding, in the case of static methods. Unlike overloading, you do not create a new method, just redefine an existing one, using the keyword OVERRIDING.

Restrictions

- Only methods that are not declared to be final in the supertype can be overridden.

- Order methods may appear only in the root type of a type hierarchy: they may not be redefined (overridden) in subtypes.

- A static method in a subtype may not redefine a member method in the supertype.

- A member method in a subtype may not redefine a static method in the supertype.

- If a method being overridden provides default values for any parameters, then the overriding method must provide the same default values for the same parameters.

2.4 Useful Functions and Operators

Functions and Operators Useful with Objects

- **CAST** converts one built-in data type or collection-typed value into another built-in data type or collection-typed value.

- A **CURSOR** expression returns a nested cursor. This form of expression is equivalent to the PL/SQL REF CURSOR and can be passed as a REF CURSOR argument to a function.

- The **DEREF** function in a SQL statement returns the object instance corresponding to a REF. The object instance returned by DEREF may be of the declared type of the REF or any of its subtypes.

- The **IS OF type** predicate tests object instances for the level of specialization of their type.

- The **REF** function in a SQL statement takes as an argument a correlation name (or table alias) for an object table or view and returns a reference (a REF) to an object instance from that table or view. The **REF** function may return references to objects of the declared type of the table, view, or any of its subtypes.

- The **SYS_TYPEID** function can be used in a query to return the typeid (a hidden type) of the most specific type of the object instance passed as an argument.

- **TABLE** functions are functions that produce a collection of rows, a nested table or a varray, that can be queried like a physical database table or assigned to a PL/SQL collection variable. You can use a table function like the name of a database table, in the FROM clause of a query, or like a column name in the SELECT list of a query.
The **TREAT** function does a runtime check to confirm that an expression can be operated on as if it were of a different specified type in the hierarchy, normally a subtype of the declared type of the expression. In other words, the function attempts to treat a supertype instance as a subtype instance, for example, to treat a person as a student. If the person is a student, then the person is returned as a student, with the additional attributes and methods that a student may have. If the person is not a student, TREAT returns NULL in SQL.

The **VALUE** function takes as its argument a correlation variable (table alias) for an object table or object view and returns object instances corresponding to rows of the table or view. The VALUE function may return instances of the declared type of the row or any of its subtypes.

### 3 Resources

**Resources**

- [http://docs.oracle.com/cd/B10501_01/appdev.920/a96624/10_objs.htm](http://docs.oracle.com/cd/B10501_01/appdev.920/a96624/10_objs.htm)
- [http://docs.oracle.com/cd/B28359_01/appdev.111/b28371/adobjbas.htm](http://docs.oracle.com/cd/B28359_01/appdev.111/b28371/adobjbas.htm)