

Simple Strategies for Complex Data: Oracle9i Object-Relational Technology

An Oracle Technical White Paper
July 2001

Oracle9i Object-Relational Technology

INTRODUCTION.....	1
OBJECT-ORIENTED APPLICATION DEVELOPMENT.....	2
ORACLE'S OBJECT TYPE SYSTEM.....	4
Object Types.....	4
Object Views.....	5
Collection Types.....	9
Reference Types.....	11
Large Objects.....	11
Type Evolution.....	11
MAPPING OBJECTS BETWEEN SQL AND JAVA.....	12
JDBC, SQLJ and JPub.....	12
SQLJ Object Types.....	13
MAPPING SQL OBJECTS TO C++.....	14
MAPPING SQL OBJECTS TO XML.....	15
XML Generation.....	15
XML Storage.....	16
JDEVELOPER SUPPORT OF OBJECT-RELATIONAL TECHNOLOGY.....	16
Oracle Business Components for Java.....	17
Container Managed Persistence using Oracle9i Objects.....	17
Mapping Oracle Object Types to CMP Fields.....	17
JPublisher Wizard.....	17
DEPLOYING OBJECT-ORIENTED APPLICATIONS TO ORACLE INTERNET PLATFORMS.....	18
CONCLUSION.....	19

Simple Strategies for Complex Data: Oracle9i Object-Relational Technology

INTRODUCTION

Oracle9i[™] has rapidly evolved into a database for all your data - from simple to complex types. Multimedia data types like images, maps, video clips, and audio clips were once rarely seen outside of specialty software. Today, many Web-based applications require their database servers to manage such data. Other software solutions need to store data dealing with financial instruments, engineering diagrams, or molecular structures. To meet these needs, Oracle9i[™] database server features Object-Relational Technology to provide simple strategies for the development, deployment, and management involving complex data.

Oracle9i[™] has been enhanced to support full object modeling capabilities, including inheritance and multi-level collections, with type evolution capabilities.

With object-relational technology, the Oracle9i[™] server can be enhanced by developers to create their own application-domain-specific data types. Oracle9i[™] has been enhanced to support full object modeling capabilities, including inheritance and multi-level collections, with type evolution capabilities. For example, you can create new data types representing customers, financial portfolios, photographs or telephone networks – and thus ensure that your database programs deal with the same level of abstraction as your application domain. In many cases, it is desirable to integrate these new domain types as closely as possible with the server so they are treated at par with the built-in types like NUMBER or VARCHAR. This can be accomplished using the extensibility services. With such integration, the database server can be readily extended for new domains.

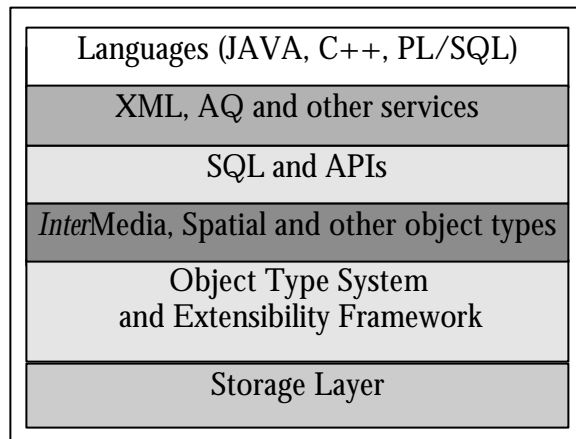


Figure 0 Oracle9i Object-Relational Architecture

Oracle9i™ also provides extensive language binding APIs. There is native support for Java inside the database and a tight integration between the object-relational type system and the Java environment. Mapping between SQL and C++ objects is also supported to provide seamless access of SQL objects from a C++ application. As XML is fast becoming the standard for information interchange. Using the object-relational framework, XML data can be stored, indexed and queried efficiently.

OBJECT-ORIENTED APPLICATION DEVELOPMENT

Since its inception in the 1960s, the breadth and the depth of object-oriented technology has steadily matured over the years. As industry standards emerged to bring about its pervasive adoption, object-oriented application development is now is the mainstream of information technology. Most notable of these standards are the UML (Unified Modeling Language) for object-oriented analysis and design, the SQL:1999 standard for object-relational database, and the Java and the C++ languages for object-oriented programming. Before delving into Oracle9i's implementation of SQL:1999 standard and its corresponding object-oriented application programming interfaces (e.g., Java, C++), it is important to understand UML as software developers do these days.

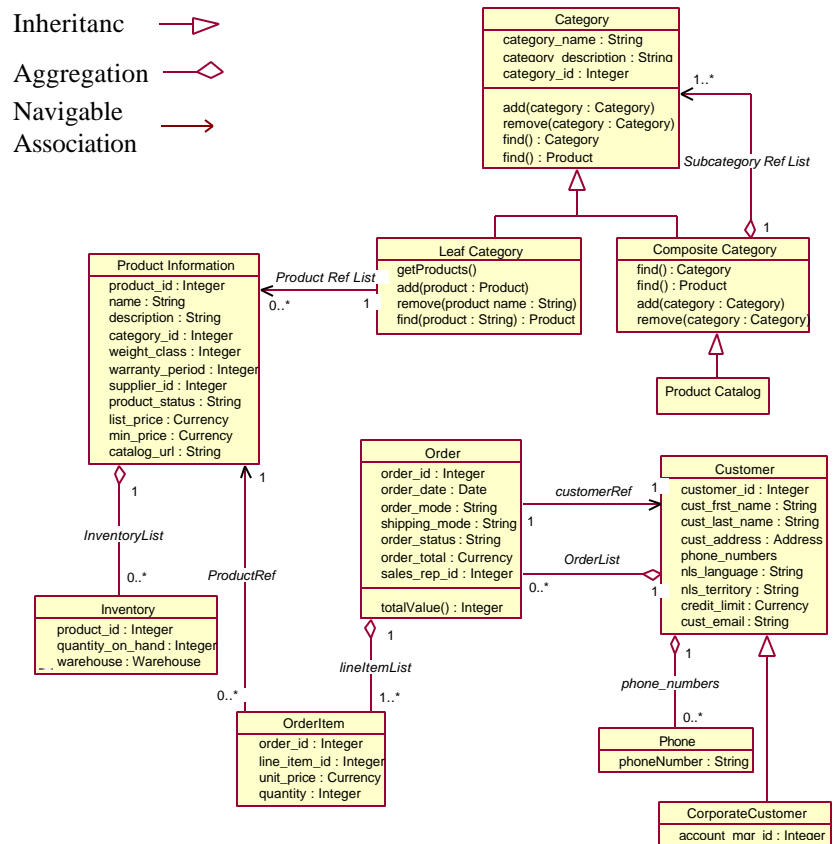


Figure 1 The UML diagram of an Online Catalog application

This UML specification represents the convergence of best practices in the object- technology industry. It is currently in release 1.3 from the Object Management Group (OMG), which is an international organization for object technology standards. UML defines standard constructs for describing object-oriented software as an object model. For example, the following UML diagram depicts the object model of an online catalog application, which Oracle has developed as part of the new Common Schema project. There are a number of

classes (e.g., Category, leafCategory) in the object model. Each of these classes have a number of attributes and operations. For example, the Category class has category_name and category_description attributes, as well as add() and remove() operations. Classes also have various relationships among them. For example, the leafCategory class inherits from the Category class; the compositeCategory class has an aggregation of Category classes; and the leafCategory class has a navigable association with Product_Information class (i.e., leafCategory classes references a list of Product_Information objects).

Once an application is analyzed and designed using UML, the resulting object model can then be mapped into target implementation with specific programming language and persistent data store. Afterward, the implemented application is ready to be deployed to specific target architecture. The benefits of Oracle9i Object-Relational Technology become apparent as this object-oriented development process unfolds. As described in the next section, Oracle9i's Object Type System allows complete one-to-one mapping of an UML object model's constructs into corresponding object-relational schema.

ORACLE'S OBJECT TYPE SYSTEM

Historically, applications have focused on accessing and modifying corporate data that is stored in tables composed of native SQL data types such as INTEGER, NUMBER, DATE, and CHAR. In Oracle9i, there is support not only for these native types, but also for new 'object' data types, that are now a part of the recent ANSI SQL99 standard. This section takes a brief look at the basic features of the object-relational type system.

Object Types

Oracle has extended SQL (DDL and DML), to allow users to define their own types (that represent their business objects) and relationships (e.g., inheritance, aggregation) among these types, store them as base or native types within the database (either within a column of a table or as tables themselves), and query, insert, and update them. They can contain one business object inside of another, point from one business object to another (using a pointer called a REF), and access and manipulate collections or sets of these objects, using structures called VARRAYS and Nested Tables. Users can define operations on business objects as methods of the objects. Methods can be implemented as PL/SQL stored procedures. Objects have globally unique identifiers, called Object IDs, that capture references between objects.

Oracle9i allows a user to treat object data relationally and relational data as objects. For example, users can use SQL to query on object data in the same way that they access relational data. Users can access an object (using SQL DML for the query), the object types attributes and methods, with extended path expressions (for example, object.attribute). They can also use SQL to perform explicit joins between objects in tables. In addition, Oracle9i lets users perform

An object type, distinct from native SQL data types, is user-defined and specifies both the underlying persistent data (called 'attributes' of the object type) and the related behaviors ('methods' of the object type).

implicit joins between objects, by traversing or navigating the REF from one object to the other. Object types are indexable, using MAP or ORDER methods to convert them to scalar values, which can then be indexed.

Oracle9i's object constructs have a close correspondence with the relational constructs with which Oracle's customers are familiar. For example, a REF is very similar to a foreign key, methods are stored procedures (which can be written in Java, PL/SQL or C/C++), the security and transaction models that operates on object types are exactly the same that Oracle defined for relational tables.

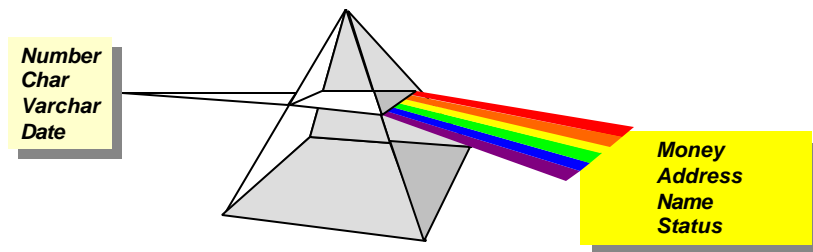


Figure 2 User defined types allow application-domain constructs

The object type system essentially raises the level of abstraction with which database programs are written. Instead of dealing with NUMBERS, CHARs, and so on, programs can deal with application-domain constructs, such as Customer, Portfolio, or Money. This leads to many benefits, not the least of which is a better modeling of your business in the database.

Object Views

Object Views allow you to synthesize a business object from data that continues to be stored in relational tables. Specifically,

- Define objects you can use in your applications without migrating any relational data.
- Combine objects developed for one application in different ways for use with other applications. Objects in an object view have much of the functionality object tables have. They can have methods, belong to collections, point to one another, have object identity, and accessed from SQL or via pointer traversal. Further, Oracle has extended the view mechanism, to use special INSTEAD OF triggers to provide fully updatable views.

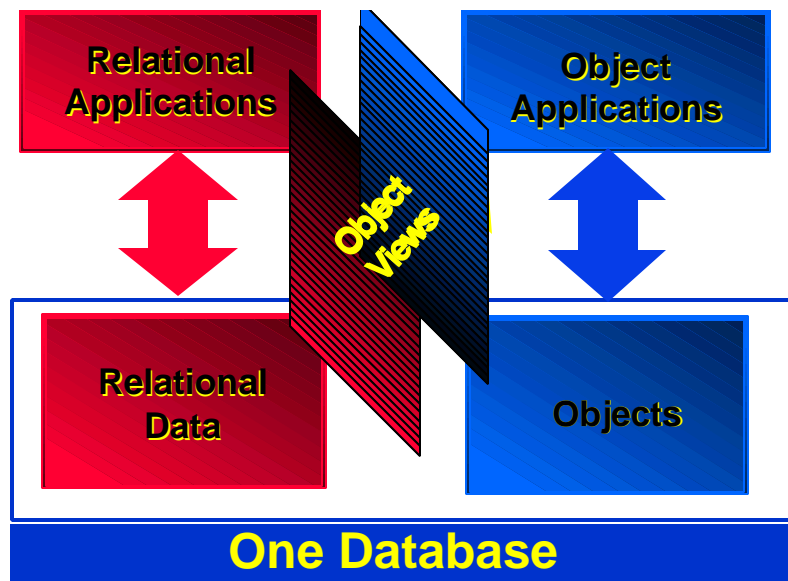


Figure 3 Object Views add flexibility to legacy data access

Type inheritance allows sharing similarities between types as well as extending their characteristics.

Inheritance

Type inheritance is a fundamental concept in any object-oriented system. Type inheritance allows sharing similarities between types as well as extending their characteristics.

Most object-oriented applications organize their objects into types, and types into type hierarchies. Empirically, it is sufficient to organize type hierarchies into a set of trees. Thus, single type inheritance is sufficient to support type organization for most applications. Java is an object-oriented programming languages supporting single inheritance. With single inheritance, a type may extend (inherit from) one supertype. Such a type (called subtype) inherits all its supertype's attributes and methods. A subtype may also add new attributes and methods and/or override inherited methods.

Substitutability is the primary characteristic of type polymorphism, which allows a value of some subtype to be used where a supertype is expected.

One of the major benefits of inheritance is substitutability. Substitutability is the primary characteristic of type polymorphism, which allows a value of some subtype to be used where a supertype is expected (e.g. method parameters), without any specific knowledge of the subtype being needed in advance. Instance substitutability refers to the ability to use an object value of a subtype in a context declared in terms of a supertype. REF substitutability refers to the ability to use a REF to a subtype in a context declared in terms of a REF to a supertype.

Oracle supports the single type inheritance model. This is closely aligned with the ANSI SQL99 standards. The next few sections provide details on Oracle's support for inheritance.

Type Hierarchy

The root type of a hierarchy is created using the CREATE TYPE statement and should be declared to be NOT FINAL.

```
CREATE TYPE Person_t AS OBJECT(  
    name VARCHAR2(100),  
    dob DATE,  
    MEMBER FUNCTION age() RETURN number,  
    MEMBER FUNCTION print() RETURN varchar2) NOT FINAL;
```

A subtype can be created under a non final type. It inherits all attributes and methods from its supertype. It can add new attributes and methods and/or override inherited methods.

```
CREATE TYPE Employee_t UNDER Person_t(  
    salary NUMBER,  
    bonus NUMBER,  
    MEMBER FUNCTION wages() RETURN number,  
    OVERRIDING MEMBER FUNCTION print() RETURN varchar2);
```

In the Oracle Common Schema, we have a more elaborate example of type inheritance hierarchy. As shown in Figure 1, the Category class and its sub-classes are modeled with an elegant yet simple construct to represent tree-structured part-whole hierarchies. This example reveals further the major benefits of Oracle9i's object type system in preserving all aspects of an application's object model.

```
create type category_typ as object  
    ( category_name          varchar2(50)  
      , category_description  varchar2(1000)  
      , category_id          number(2)  
    )  
NOT INSTANTIABLE NOT FINAL;  
  
create type subcategory_ref_list_typ as table of ref  
category_typ;  
  
create type product_ref_list_typ as table of number(6);  
/  
reate type corporate_customer_typ under customer_typ  
    ( account_mgr_id        number(6)  
    );  
  
create type leaf_category_typ under category_typ  
    (  
    product_ref_list        product_ref_list_typ  
    );  
  
create type composite_category_typ under category_typ  
    (  
    subcategory_ref_list    subcategory_ref_list_typ  
    )  
NOT FINAL;
```

```

create type catalog_typ under composite_category_typ
(
    member function getCatalogName return varchar2
);

```

View Hierarchy

An object view can be created as a subview of another object view, thereby creating a view hierarchy. The benefits of a view hierarchy are due to the following fact : By default, the rows of an object view in a view hierarchy includes all the rows of all its subviews. Due to this property, you can query objects of types belonging to a hierarchy in a meaningful fashion.

```

CREATE VIEW Persons OF Person_t WITH OBJECT ID (name) AS
    SELECT name, dob FROM r_persons;

```

```

CREATE VIEW Employees OF Employee_t UNDER Persons AS
    SELECT name, dob, salary, bonus from r_employees;

```

A query over *Persons* view will retrieve all persons including employees.

```

SELECT VALUE(p) FROM Persons p;

```

Oracle provides new operators to test the most-specific-type of an instance and to convert an object of a supertype T to that of a subtype of T (if legal). For example, the following query retrieves all persons with a given *dob* and are employees.

```

SELECT TREAT(VALUE(p) AS Employee_t)FROM Persons p
WHERE dob = '01-01-1970' AND VALUE(p) IS OF(Employee_t);

```

Some other properties of a view hierarchy :

- The type of the superview must be the immediate supertype of the type of the object view being created.
- The subview inherits the object identifier (OID) from its superview.

Substitutability

Object columns and object tables are substitutable, by default. This implies that subtype instances can be stored within supertype containers. The underlying storage model is a single flat table - with columns corresponding to all the attributes of all possible subtypes. A type id column stores the most specific type information of the object.

```

CREATE TABLE dept (id NUMBER, mgr Person_t);
INSERT INTO dept VALUES(1, Employee_t(...));

```

Further, REF columns and collection elements are also substitutable.

Method Invocation

A method is a procedure or a function that is part of an object type definition. Methods can be run within the execution environment of Oracle9i or dispatched to run outside it. Methods can be implemented in a variety of languages including PL/SQL, C/C++, and Java.

Oracle supports a multi-language dynamic method dispatch capability. When a method is invoked on an object instance, it is dispatched to a specific implementation based on most specific type of the instance.

A subtype can override any of the non-final member methods defined within its supertype and supply a different implementation. The methods in the supertype can be implemented in a different language from the overriding methods in the subtype. Oracle supports a multi-language dynamic method dispatch capability. When a method is invoked on an object instance, it is dispatched to a specific implementation based on most specific type of the instance.

Client Environments

Full support for type inheritance is available in a variety of client environments including PL/SQL, Java and C/C++. Subtype instances can be accessed and manipulated via API's such as JDBC and OCCI (i.e., Oracle C++ Call Interface). Oracle also provides tools like JPublisher and Object Type Translator(OTT) to generate respective Java and C++ mappings from database object type hierarchies. Instance and REF substitutability is also supported within these language environments.

Collection Types

Collections are SQL data types that contain multiple elements. Each element or value for a collection has the same substitutable data type. In Oracle, there are two collection types – Varrays and Nested Tables.

A Varray contains a variable number of ordered elements. Varray data types can be used as a column of a table or as an attribute of an object type.

Using Oracle SQL, a (named) table type can be created. These can be used as Nested Tables to provide the semantics of an unordered collection. As with Varray, a Nested Table type can be used as a column of a table or as an attribute of an object type.

Multi-Level Collections

Oracle9i supports multiple levels of nesting within collections, e.g. Nested Tables or Varrays embedded within a Nested Table or Varray.

There is a perfect example in the Oracle Common Schema to take advantage of multi-level collections support. In the UML diagram of Figure 1, the *Customer* class is shown to have an aggregation of *Order* class, which in turn has an aggregation of *Order_Item* class. The following example shows how to map these constructs by creating a *customer_typ* type that contains a Nested Table collection

of `order_typ` type, which in turn has a Nested Table collection of `order_item_typ` type.

```
create type order_item_typ as object
( order_id          number(12)
, line_item_id     number(3)
, unit_price       number(8,2)
, quantity         number(8)
, product_ref     REF  product_information_typ
) ;

create type order_item_list_typ as table of order_item_typ;

create type customer_typ;

create type order_typ as object
( order_id          number(12)
, order_mode       varchar2(8)
, customer_ref     REF  customer_typ
, order_status     number(2)
, order_total      number(8,2)
, sales_rep_id     number(6)
, order_item_list  order_item_list_typ
) ;

create type order_list_typ as table of order_typ;

create or replace type customer_typ as object
( customer_id      number(6)
, cust_first_name  varchar2(20)
, cust_last_name   varchar2(20)
, cust_address     cust_address_typ
, phone_numbers    phone_list_typ
, nls_language     varchar2(3)
, nls_territory    varchar2(30)
, credit_limit     number(9,2)
, cust_email       varchar2(30)
, cust_orders      order_list_typ
);
```

The same example from the Oracle Common Schema creates an object view for the multi-level collections.

```
create or replace view oc_customers of customer_typ
with object oid (customer_id)
as select c.customer_id, c.cust_first_name,
         c.cust_last_name, c.cust_address,
         c.phone_numbers, c.nls_language,
         c.nls_territory, c.credit_limit,
         c.cust_email,
         cast(multiset(select o.order_id, o.order_mode,
                           make_ref(oc_customers, o.customer_id),
                           o.order_status,
                           o.order_total, o.sales_rep_id,
```

```

        cast(multiset(select
            l.order_id,l.line_item_id,
            l.unit_price,l.quantity,
            make_ref(oc_product_information,
            l.product_id) from order_items l
            where o.order_id = l.order_id)
        as order_item_list_typ)
    from orders o
    where c.customer_id = o.customer_id)
as order_list_typ)
from customers c;

```

Reference Types

If you create an object table or an object view in Oracle9i, it is possible to obtain a reference (or the database *pointer*) to an associated row object. References are important for modeling relationships and navigating among object instances particularly in client-side applications.

Large Objects

Oracle9i provides the large object (LOB) types to handle the storage demands of images, video clips, documents, and other such forms of unstructured data. Large objects are stored in a manner that optimizes space utilization and provides efficient access. More specifically, large objects are composed of locators and the related binary or character data. The LOB locators are stored in-line with other table record columns. In case of *internal* LOBs (BLOB, CLOB, and NCLOB) the data can reside in a separate storage area. However, for external LOBs (BFILEs), the data is stored outside the database in operating system files.

Type Evolution

The usage of type enables the user to evolve their business logic captured in the behavior of the type. Type Evolution is a mechanism that enables the user to change the type and propagate these changes to other schema objects that references the modified type. The schema objects that may reference a type include other types, subtypes, row objects, column objects, program units (packages, functions, procedures), views, functional indexes, or triggers.

The type evolution operations supported include the following:

1. Operation on a type attribute:
 - Add an attribute to a type.
 - Drop an attribute from a type.
 - Modify the type of an attribute by increasing its length, precision, or scale.
2. Operation on a type method:
 - Add a method to a type.

Type Evolution is a mechanism that enables the user to change the type and propagate these changes to other schema objects that references the modified type.

- Drop a method from a type.
3. Change the INSTANTIATABLE and FINAL property of a SQL Object type.
 4. Support for explicit propagation of a type change to its dependent types and tables.

These type evolution changes are either *structural* or *non-structural*. Structural changes are those affecting the state of the object such as adding or dropping an attribute. A non-structural change has no effect on the state of the object. For example, renaming an attribute or adding a method.

Consider the following example:

```
CREATE TYPE address_t AS OBJECT (
    street VARCHAR(100),
    zip_code NUMBER);

CREATE TYPE person_t AS OBJECT (
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    age NUMBER,
    address address_t);

CREATE TABLE person_tab OF person_t;
```

In the example, the person_tab is created with a column for each attribute in person_t as well as one for each address attribute. Consider the following type change example:

```
ALTER TYPE address_t
    ADD ATTRIBUTE (country NVARCHAR(100), int_postal_code
    NVARCHAR(20)) CASCADE;
```

The effect of this change is that two columns are added to the person_tab table corresponding to the newly added attributes. The CACADE keyword propagates the type change to dependent types and tables.

MAPPING OBJECTS BETWEEN SQL AND JAVA

JDBC, SQLJ and JPub

The object-relational facilities provide a more natural and productive way to maintain a consistent structure between a set of Java classes at the application level and the data model at the data storage level. In Oracle, the object-relational facilities have been tightly integrated with the Java environment using standard JDBC and SQLJ APIs. The SQL object types can be mapped to Java classes. The JPublisher (JPub) utility automatically generates Java or SQLJ files of Java classes implementing this mapping. Each generated Java class contains associated

In Oracle, the object-relational facilities have been tightly integrated with the Java environment using standard JDBC and SQLJ APIs.

methods to read and write the object state from and to the database server. In turn, Java application programs can use these generated Java classes to store and retrieve objects in the database

The following is the code fragments of a JPub-generated Java class.

```
// Create Java class that implements the SQLData interface
public class JPurchaseOrder implements SQLData
{
    ...
    public void readSQL (SQLInput stream, String typeName)
        throws SQLException {...}
    public void writeSQL (SQLOutput stream)
        throws SQLException {...}
    ...
}
```

This Java class JPurchaseOrder can then be used in the following Java program to retrieve objects from the database.

```
// An object instance can be viewed as a Java object
ResultSet rs = stmt.executeQuery("select value(p) from
purchase_order_tab p");

rs.next();
JPurchaseOrder jp = (JPurchaseOrder) rs.getObject(1);
String streetName = jp.shipAddr.street;
```

SQLJ Object Types

Oracle9i introduces the ability to seamlessly map a given Java class to a SQL object. In accordance with SQLJ Part II standard, a *SQLJ Object Type* defines the mapping between a SQL object to its corresponding implementation class in Java. SQLJ Object Type is also in the process of being standardized as a SQL99 extension.

Consider the following example:

```
CREATE TYPE ADDRESS_T AS OBJECT EXTERNAL NAME 'Address'
LANGUAGE JAVA USING SQLDATA (
STREET  VARCHAR(50) EXTERNAL NAME 'street',
ZIP_CODE VARCHAR(10) EXTERNAL NAME 'zip',

    STATIC FUNCTION GET_CITY (VARCHAR zip) RETURN VARCHAR
        EXTERNAL NAME 'get_city(java.lang.String)
        RETURN java.lang.String',
    STATIC FUNCTION GET_STATE (VARCHAR zip) RETURN VARCHAR
        EXTERNAL NAME 'get_state(java.lang.String)
        RETURN java.lang.String',
    MEMBER FUNCTION CITY RETURN VARCHAR EXTERNAL NAME 'getCity()
        RETURN java.lang.String',
```

```
MEMBER FUNCTION STATE RETURN VARCHAR EXTERNAL NAME
  'getState() RETURN java.lang.String');
```

In the above example, the CREATE TYPE statement specifies an external name clause that identifies the Java class used in the implementation. The USING clause specifies the interface used to persist the object state.

Each of the attributes of the object type has an optional EXTERNAL NAME clause to specify their corresponding Java field. For methods, the EXTERNAL NAME clause specifies the corresponding Java function name and its signature.

The above mapping information is stored as part of the type metadata. This enables seamless access to SQL objects from Java via JDBC without requiring the user to either explicitly implement the SQLData interface or register the class with the Java typemap.

A SQLJ object type hierarchy can be mapped to a Java inheritance hierarchy. The method dispatch including overrides for these types is performed by the Java runtime.

MAPPING SQL OBJECTS TO C++

The Oracle C++ Call Interface (OCCI) specifies a C++-based mapping of object types. Thus, object type instances in the database can be accessed and modified to and from C++ objects in your application.

The Oracle C++ Call Interface (OCCI) specifies a C++-based mapping of object types. Thus, object type instances in the database can be accessed and modified to and from C++ objects in your application. The OCCI navigational interface lets you access and modify object-relational data as C++ objects without using explicit SQL.

The Object Types Translator (OTT) generates default C++ classes corresponding to object types, including type hierarchies. It also supplies default implementations for the methods needed to read and write object data from and to the database. For example, consider the following object type hierarchy:

```
CREATE TYPE Person AS OBJECT (name VARCHAR2(100), age NUMBER)
NOT FINAL;
CREATE TYPE Student UNDER Person(dept VARCHAR2(50), advisor REF
Person);
```

The following are the C++ classes generated by OTT for the above type hierarchy. The OCCIPObject class is provided by the OCCI API as a base class for classes having persistent or transient objects.

```
class Person : public OCCIPObject { ...
    void *operator new(size_t size, const OCCIConnection& con,
        const OCCIString& table);
    static void readSQL(const OCCIAnyData& stream, Person *obj);
    static void writeSQL(const Person *obj, OCCIAnyData& stream);
    ...}

class Student : public Person {...
```

```

static void readSQL(const OCCIAnyData& stream, Student *obj);
static void writeSQL(const Student *obj, OCCIAnyData&
stream); ...}

```

The code fragments corresponding to a couple of the methods are given below :

```

void Person::readSQL(const OCCIAnyData& stream, Person *obj){
obj->name = stream.getString(1);
obj->age = stream.getNumber(2); }

void Student::readSQL(const OCCIAnyData& stream, Student *obj) {
Person::readSQL(stream, obj);
obj->dept = stream.getString(3);
obj->advisor = stream.getRef(4); }

```

A type mapping table maintains the association between SQL type names and C++ class names, and is used at runtime to determine the class whose readSQL/writeSQL routines need to be invoked. OTT also provides the flexibility for users to extend the generated classes to add on more functionality.

The following code fragment illustrates the retrieval of a SQL object into a C++ class instance :

```

OCCIResultSet *resultSet =
stmt->executeQuery("select VALUE(p) from person_tab p where
name = 'Joe'");

/* fetching the object creates the appropriate class instance */
Student *joe = (Student *)resultSet.getObject(1);

/* dereferencing the REF value yields the object */
Person *joe_advisor = joe->advisor->ptr();

```

MAPPING SQL OBJECTS TO XML

Oracle supports efficient and flexible mechanisms for mapping SQL objects to XML and vice-versa.

XML is a meta-markup language that is fast becoming the de-facto standard for exchanging data between businesses and applications. XML Schema specifies the structure of the XML document in terms of the datatypes and composition of each of the elements in the document. Oracle supports efficient and flexible mechanisms for mapping SQL objects to XML and vice-versa.

XML Generation

The DBMS_XMLGEN package can take in any arbitrary SQL query and map the result of the query into XML. Thus the entire result set fetched from the query is transformed into a single XML document. There are several parameters to the package for controlling the different aspects of XML generation including : generating the XMLSchema associated with the document (analogous to the describe functionality), and restricting the number of retrieved rows. For example,

the following code fragment retrieves the results of the query in XML form as shown below.

```
qryCtx := dbms_xmlgen.getContextHandle('select * from
scott.emp');
result := dbms_xmlgen.getXMLClob(qryCtx);
```

```
<?xml version="1.0" encoding="SHIFT_JIS"?>
<ROWSET>
  <ROW>
    <EMPNO>30</EMPNO>
    <ENAME>Scott</ENAME>
    <SALARY>20000</SALARY>
  </ROW>
  <ROW>
    <EMPNO>30</EMPNO>
    <ENAME>Mary</ENAME>
    <AGE>40</AGE>
  </ROW>
</ROWSET>
```

With relational data, the results are a flat non-nested XML document. To obtain nested XML structures, you can use object-relational data. An object is mapped to an XML element, with the attributes of the object mapping to sub-elements of the parent element. A collection instance is also mapped to an XML element with the collection elements appearing as repeated occurrences of a sub-element.

XML Storage

To facilitate native storage and access of XML documents, a new datatype called *XMLType* has been introduced. Users can create instances and columns of this datatype and use the type methods to extract, traverse and transform XML documents.

A *XMLType* column can store documents in either a packed form (as CLOBs) or in an exploded object-relational fashion. It provides a choice of operators to query the XML document : *extract()* to traverse and extract a fragment of the document based on a path expression and *existsNode()* to check for the existence of a node satisfying the given condition. The user can also define *interMedia* text index on this column and query the XML document using *Contains* and other text-based operators.

```
-- select the customer name for all messages where the message
-- is urgent and has a customer inside a PO
SELECT extract(e.msgVal, '/po/cust/custname')
FROM message_tab e
WHERE CONTAINS(e.msgVal, 'URGENT') > 0
AND existsNode(e.msgVal, "//po/cust") > 0;
```

JDEVELOPER SUPPORT OF OBJECT-RELATIONAL TECHNOLOGY

Oracle provides powerful tools to support object-oriented application development. For Java developers, JDeveloper offers two types of integrated support for developing applications that utilize Oracle's Object-Relational technology. One type uses the integrated Oracle Business Components for Java framework to map domains to Oracle9i object types and entity objects to object tables and object views. Entity objects can be deployed as Enterprise Javabeans (EJB) using Container Managed Persistence (CMP). The other type of support uses the JPublisher wizard to generate Java classes from Oracle9i object types. Developers use Java classes generated from JPublisher wizard to access and manipulate Oracle9i row objects and column objects.

Oracle Business Components for Java

Oracle Business Components for Java framework maps its domains to Oracle9i object types and its entity objects to object tables and object views. A JDeveloper wizard can create entity object from an Oracle9i object table or object view. An entity object can be deployed as EJB with Container Managed Persistence.

Container Managed Persistence using Oracle9i Objects

Oracle9i manages EJB Entity bean persistence using database tables. Each instance of an entity bean corresponds to a row in the table, and each CMP field corresponds to a column in the table. An Entity bean class also needs a primary key, corresponding to one or more columns of the table, that allow instances to be retrieved using the `findByPrimaryKey()` method.

Corresponding to the tables are two Business Components for Java components:

1. An EJB/9i deployment object, which is very similar to a standard Entity Object and stands for the table that manages persistence
2. A view object which selects the relevant columns of the table and aliases them to the EJB fields

CMP beans can also make use of other business components classes: domains that allow EJB fields to be based on Oracle object types and secondary view objects corresponding to EJB finder methods.

Mapping Oracle Object Types to CMP Fields

The database table managing the CMP persistence may include columns containing Oracle Object types. In JDeveloper, a Domain Wizard helps to create domain objects from Oracle Object type.

JPublisher Wizard

JDeveloper has a Database Browser that lets developers navigate the contents of their database schemas, locating Oracle9 Objects. From this interface the JPublisher Wizard can be invoked to generate Java wrapper classes for Oracle9i

Objects. JPublisher makes it easier to use Oracle9i Objects in Java programs, by automatically generating the appropriate Java class definitions. JPublisher-generated Java wrapper classes include methods to convert data from SQL to Java and from Java to SQL, as well as getter and setter methods for the attributes of the objects. JPublisher increases developer productivity, while still providing the flexibility to extend the generated classes to suit custom needs.

Once a Java application is developed with JDeveloper, it can be deployed to various target architectures with a combination of Oracle9i database servers and Oracle9i application servers, which are versatile software platforms for handling highly demanding applications.

DEPLOYING OBJECT-ORIENTED APPLICATIONS TO ORACLE INTERNET PLATFORMS

Once the development of an application using Oracle9i Object-Relational Technology is complete, there are a number of ways to deploy and manage the application. Oracle9i AS (i.e., Application Server) and Oracle9i Database Server provide everything necessary to deploy and manage e-business applications. Together, Oracle9i AS and the Oracle Database constitute a simple, complete, and integrated Internet platform.

- **Simple.** Oracle9i AS and the Oracle Database are simple to buy, simple to install, and simple to manage. All of Oracle's core middle-tier services have been integrated into Oracle9i AS, enabling customers to build and deploy portals, transactional applications, and business intelligence facilities with a single product. The full set of services of Oracle9i AS are integrated out-of-the-box with a single installation, and Oracle9i AS and the Oracle Database are managed with a single management tool.
- **Complete.** With the Oracle9i Database to manage data and Oracle9i AS to run applications, the Oracle Internet Platform is a complete solution for building and deploying any type of application to the web, including content management, OLTP, mobile, business intelligence, and enterprise integration applications. Oracle9i AS and the Oracle9i Database provide a scalable and highly available infrastructure that enables customers to easily accommodate growing user populations without needing to recode their applications.
- **Integrated.** Oracle9i AS is simply the best application server for the Oracle9i Database. By leveraging a common technology stack, Oracle9i AS can transparently scale an Oracle9i Database by caching data and application logic on the middle tier. Additionally, Oracle9i AS inherits much of its robust scalability and availability features from the mature technology of Oracle9i.

Oracle's object-relational technology has grown to maturity over the years to provide a complete object type system, extensive language binding APIs, and a rich set of utilities and tools.

CONCLUSION

Oracle's Object-Relational Technology has grown to maturity over the years to provide a complete object type system, extensive language binding APIs, and a rich set of utilities and tools. This complete object type system is based on the recent ANSI SQL-99 standard. Oracle has optimized the object type performance in the database server for object-oriented applications. Oracle's language binding APIs in Java, C/C++, and XML provide direct interfaces to database server object type system. These comprehensive APIs support the most recent standards to access database object type system services. The attendant rich set of utilities for object-relational data include import/export, SQL loader, replication, etc.

In addition, there are also a number of development tools that supports Oracle's object-relational technology for developing object-oriented software, such as Oracle JDeveloper BC4J, and other products offered by Oracle partners. Oracle also provides high performance, robust, and scaleable internet platforms for application deployment and management, namely the Oracle9i Database and the Oracle9i Application Server.

IT industry facing rapid changes demands practical solutions to tackle its problems of complex applications processing various data types. Oracle's object-relational technology addresses these problems with the most comprehensive solution for the development, deployment, and management of such applications. Oracle has been the leader of object-relational technology. Oracle will continue to meet the needs of our partners and customers with the best object-relational technology.



Oracle9i Objects

November 2000

Author: Geoff Lee

Contributing Authors: Sandeepan Banerjee, Vishu Krishnamurthy

Oracle Corporation

World Headquarters

500 Oracle Parkway

Redwood Shores, CA 94065

U.S.A.

Worldwide Inquiries:

Phone: +1.650.506.7000

Fax: +1.650.506.7200

Web: www.oracle.com

This document is provided for informational purposes only and the information herein is subject to change without notice. Please report any errors herein to Oracle Corporation. Oracle Corporation does not provide any warranties covering and specifically disclaims any liability in connection with this document.

Oracle is a registered trademark, and Oracle9i is a trademark(s) or registered trademark(s) of Oracle corporation. All other names may be trademarks of their respective owners.

Copyright © Oracle Corporation 2000

All Rights Reserved