

用于复杂数据的简单策略:

Oracle9i 对象关系技术

*Oracle 技术白皮书*

2002 年 5 月

# 用于复杂数据的简单策略： Oracle9i 对象关系技术

简介 .....	3
面向对象的应用程序开发 .....	4
Oracle 的对象类型系统 .....	5
对象类型.....	5
对象视图.....	6
集合类型.....	10
引用类型.....	11
类型演变.....	12
语言绑定 .....	13
在 SQL 和 JAVA 间映射对象.....	13
将 SQL 对象映射到 C++.....	14
映射 SQL 对象至 XML .....	15
Oracle XML DB.....	16
XML 生成.....	16
XML 存储.....	17
对象关系技术的 JDeveloper 支持 .....	17
Oracle Business Components for Java.....	17
Container Managed Persistence 使用 Oracle9i 对象 .....	18
映射 Oracle 对象类型至 CMP 字段.....	18
JPublisher 向导.....	18
部署面向对象应用程序至 Oracle Internet 平台 .....	19
总结 .....	19

## 用于复杂数据的简单策略： Oracle9i 对象关系技术

### 简介

Oracle9i™ 已经迅速发展成为可用于所有数据（从简单到复杂的各种类型数据）的数据库。多媒体数据类型（如图像、地图、视频剪辑和音频剪辑等）曾经只能在专业软件中见到。现在很多基于 Web 的应用程序需要它们的数据库服务器管理此类数据。其他的软件解决方案需要存储涉及金融工具、设计图表或分子结构的数据。为了满足这些需求，Oracle9i™ 数据库服务器采用“对象关系技术”，为涉及复杂数据的开发、部署和管理提供简单策略。

采用类型演变功能，Oracle9i™ 已经增强为可以支持包括继承和多级别集合在内的全面的对象建模功能。

通过使用对象关系技术，开发人员可以增强 Oracle9i™ 服务器，从而创建他们自己的“应用程序域特定”的数据类型。采用类型演变功能，Oracle9i™ 已经增强了支持全面的对象建模功能，包括继承和多级集合在内。例如，可以创建代表客户、资产组合、照片或电话网络的新数据类型，然后，确保数据库程序可以处理与应用程序域同样的抽象级别。很多情况下，需要将这些新域类型与服务器尽可能紧密地集成，这样它们便能作为内置类型（如 NUMBER 或 VARCHAR）看待。使用扩充性服务可以实现这一目的。采用这样的集成方式，可以将数据库服务器方便地扩展到新域中。

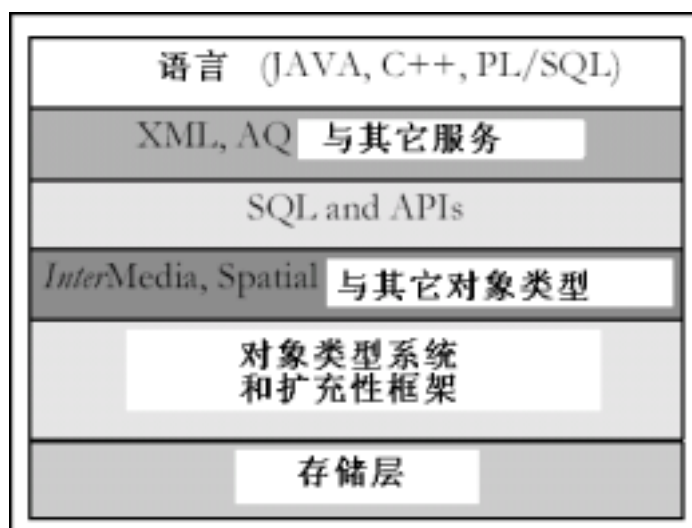


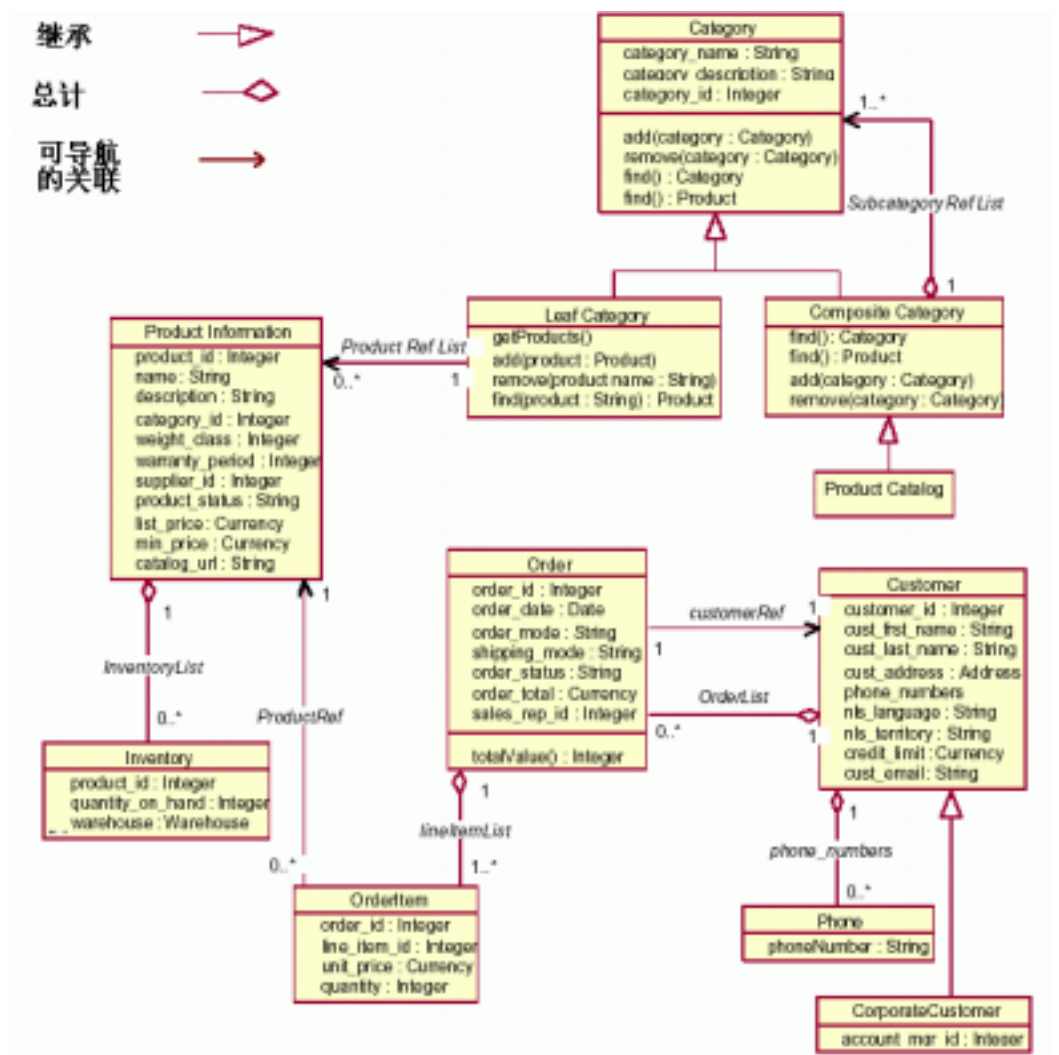
图1 Oracle9i 对象关系体系结构

Oracle9i™ 还提供广泛的语言绑定 API。数据库内部对 Java 提供本地支持，对象关系类型系统和 Java 环境间可以紧密封集。它还支持 SQL 和 C++ 对象间的映射，从而可以从 C++ 应用程序无缝地访问 SQL 对象。当前，XML 正迅

速成为信息交换的标准。通过使用对象关系框架，可以有效地存储、索引和查询 XML 数据。

## 面向对象的应用程序开发

自从 20 世纪 60 年代面向对象技术出现，数十年来该技术在广度和深度上不断发展、成熟。逐渐成为被广泛采用的行业标准，面向对象的应用程序开发目前已经是信息技术的主流。最著名的一些标准包括：用于面向对象分析和设计的 UML（统一建模语言）、用于对象关系数据库的 SQL:1999 标准以及用于面向对象编程的 Java 和 C++ 语言。在深入研究 Oracle9i 的 SQL:1999 标准的实现及其相应的面向对象应用程序编程接口（例如 Java、C++）之前，当今的软件开发人员应该充分了解 UML，这一点非常重要。



此 UML 规范汇集了对象技术行业中的最佳实践应用。它当前的版本是由对象技术标准的国际组织“对象管理小组”(OMG) 发布的 1.3 版。UML 为将面向对象的软件作为对象模型来描述定义了标准结构。例如, 以下 UML 图说明某个在线目录应用程序的对象模型, Oracle 已经将该应用程序发展为新“通用模式”项目的一部分。

对象模型中有很多类(例如, Category、leafCategory)。其中的每个类都有很多属性和操作。例如, Category 类具有 category\_name 和 category\_description 属性, 以及 add() 和 remove() 操作。各个类之间也存在各种关系。例如, leafCategory 类从 Category 类继承而来; compositeCategory 类具有 Category 类的集合; leafCategory 类与 Product\_Information 类具有可导航的关联(即, leafCategory 类引用 Product\_Information 对象的列表)。

使用 UML 分析和设计应用程序后, 即可使用特定编程语言和永久数据存储将生成的对象模型映射为目标实施。然后, 可将已实施的应用程序部署到特定目标体系结构。随着该面向对象开发过程的进行, Oracle9i 对象关系技术的优势逐渐显现出来。如下节所述, 使用 Oracle9i 的“对象类型系统”, 可以将 UML 对象模型的结构完全一对一地映射到相应的对象关系模式中。

## Oracle 的对象类型系统

以前, 应用程序注重访问和修改存储在诸如 INTEGER、NUMBER、DATE 和 CHAR 等本地 SQL 数据类型组成的表中的共同数据。在 Oracle9i 中, 不但支持这些本地类型, 而且也支持新的“对象”数据类型, 这是最近的 ANSI SQL99 标准中的新部分。本节将简介对象关系类型系统的基本特性。

## 对象类型

与本地 SQL 数据类型完全不同, 对象类型是用户定义的, 并指定底层永久数据(称作对象类型的“属性”)和相关的行为(对象类型的“方法”)。

Oracle 已经扩展了 SQL (DDL 和 DML), 并允许用户定义他们自己的类型(表示他们的商务对象)以及这些类型间的关系(例如, 继承、集合)、将它们作为基本或本地类型存储在数据库中(存储在表的某列中, 或存储为表本身), 以及查询、插入和更新它们。它们可以在一个商务对象中包括另一个商务对象、从一个商务对象指向另一个商务对象(使用称作 REF 的指针), 以及使用称作 VARRAYS 和“嵌套表”的结构访问和操作这些对象的集合或组合。用户可以象对象的方法一样定义对商务对象的操作。可以如同 PL/SQL 存储过程一样执行方法。对象具有全局唯一标识符, 称作 Object ID, 它用来捕获对象间的引用。

Oracle9i 允许用户相关地对待对象数据, 同时将关系数据作为对象看待。例如, 用户可以使用 SQL 按照与访问关系数据相同的方式查询对象数据。用户使用扩展的路径表达式(例如, object.attribute)可以访问对象(通过用于查询的 SQL DML)、对象类型属性和方法。他们也可以使用 SQL 在表中的各个对象间执行

显式联接。另外，通过将 REF 从一个对象传送或导航到另一个对象，Oracle9i 允许用户执行对象间的隐式联合。对象类型可以编入索引，使用 MAP 或 ORDER 方法将对象类型转换为标量值，然后可以对这些值进行索引。

Oracle9i 的对象结构与 Oracle 客户所熟悉的关系结构具有紧密的对应关系。例如，REF 与外部键很相似，方法是存储过程(可以采用 Java、PL/SQL 或 C/C++ 来编写它们)，并且，在对象类型上操作的安全和事务方法与 Oracle 为关系表定义的方法完全一样。

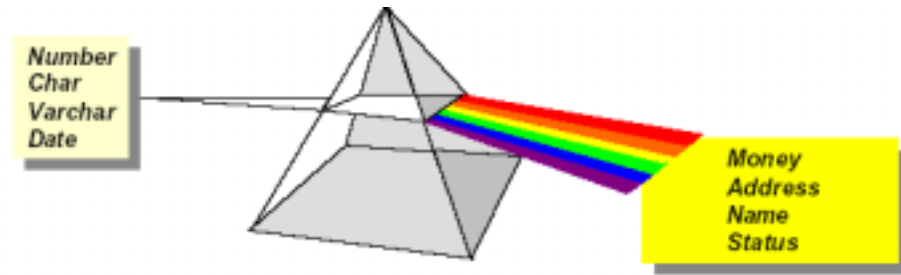


图3 用户定义类型允许采用应用程序域结构

对象类型系统从本质上提高了编写数据库程序所使用的抽象级别。因为程序并不处理 NUMBER、CHAR 等，而是处理诸如 Customer、Portfolio 或 Money 等应用程序域结构。这带来很多优点，并不仅仅只是在数据库中优化对商务的建模。

## 对象视图

另外 Oracle 9i 允许用户象对待对象一样对待关系数据。对象视图允许从持续存储在关系表中的数据合成商务对象。具体地说，

- 定义要在应用程序中使用的对象，而不用移植任何关系数据。
- 以不同的方式将为一个应用程序开发的对象与其他应用程序结合使用。对象视图中的对象具有很多对象表所具有的功能。它们可以具有方法、属于集合、指向另一个对象、具有对象身份，以及从 SQL 或通过指针遍历访问它们。另外，Oracle 已经扩展视图机制，使用特殊的 INSTEAD OF 触发器提供完全可更新的视图。

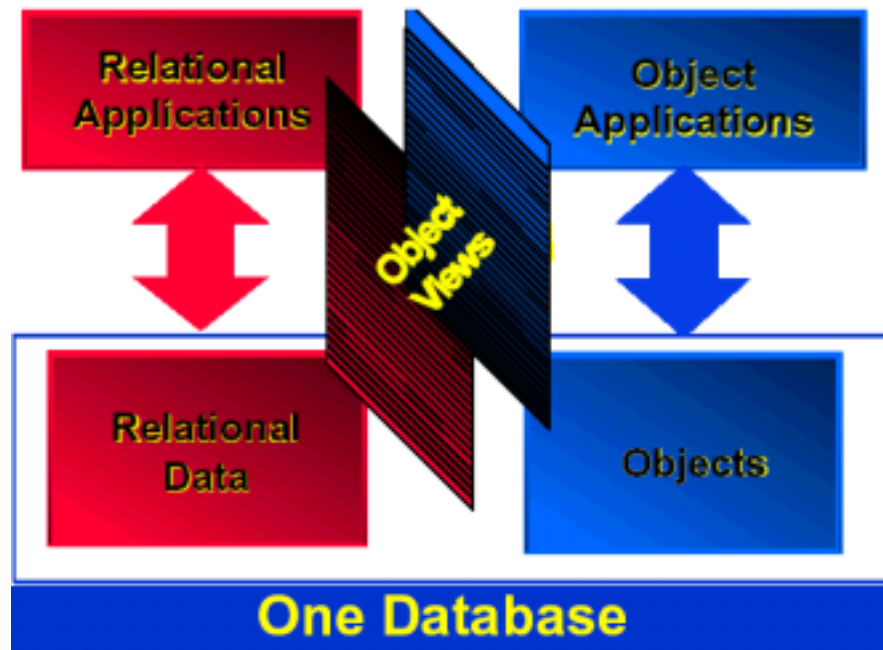


图4 对象视图为传统的数据访问增加了灵活性

## 继承

类型继承是任何面向对象系统中的基本概念。类型继承允许共享类型间的相似之处，并扩展它们的特性。

类型继承允许共享类型间的相似之处，并扩展它们的特性。

大多数面向对象应用程序将它们的对象按类型组织，并将类型按类型层次组织。根据经验表明，足可以将类型层次组织到一个树集合中。这样，单一的类型继承足可以支持大多数应用程序的类型组织。Java 是一种支持单一继承的面向对象的编程语言。采用单一继承，某个类型可以扩展（继承自）一个超类（supertype）。这样的类型（称作子类/subtype）将继承其超类的所有属性和方法。子类也可添加新属性和方法，或覆盖已继承的方法。

可替换性是类型多态性的主要特性，类型多态性允许在需要超类的地方使用某些子类的值。

可替换性是继承的主要优点之一。可替换性是类型多态性的主要特性，类型多态性允许在需要超类的地方（例如，方法参数）使用某些子类的值，而无须提前了解子类的任何特定情况。实例可替换性指在按照超类声明的上下文中使用子类的对象值的能力。REF 可替换性指在按照超类的 REF 声明的上下文中使用子类 REF 的能力。

Oracle 支持单一类型继承模型。这与 ANSI SQL99 标准紧密结合。以下几节将详细说明 Oracle 对继承的支持。

## 类型层次

使用 CREATE TYPE 语句创建层次的根类型，并应将其声明为 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;
```

可在“非最终”类型下创建子类。它从其超类继承所有属性和方法。它可添加新属性和方法，以及/或覆盖已继承的方法。

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

在“Oracle 通用模式”中，有一个更加详细的有关类型继承层次的示例。如图 2 所示，Category 类及其子类采用精致而简单的结构进行建模，表示树状结构的部分与整体间的层次。该示例进一步展示了 Oracle9i 对象类型系统在保留应用程序对象模型的所有方面中的主要优点。

```
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);  
/  
create 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;  
Oracle9i Object-Relational Technology Page 8  
create type catalog_typ under composite_category_typ  
(  
member function getCatalogName return varchar2  
);
```

## 视图层次

可以作为某个对象视图的子视图来创建另一个对象视图，从而创建视图层次。以下事实体现了视图层次的好处：默认情况下，视图层次中对象视图的行包括其子视图的所有行。通过该属性，您可以采用有意义的方式查询属于某个层次的对象。

```
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;
```

*Persons* 视图上的某个查询将检索包括雇员在内的所有人员。

```
SELECT VALUE(p) FROM Persons p;
```

Oracle 提供了新操作符以测试实例最具体的类型，并将超类 T 的对象转换为 T 的子类的对象（如果合法的话）。例如，以下查询将检索名称为指定的 *dob*，且为雇员的所有人员。

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

视图层次的某些其他属性：

- 父视图的类型必须是所创建对象视图的类型的直接超类。
- 子视图从它的父视图那里继承对象标识符（OID）。

### 可替换性

默认情况下，可以替换对象列和对象表。这意味可以将子类实例存储在超类容器中。底层存储模型是单独的一层表，具有对应于所有可能子类的所有属性的列。类型 *id* 列存储对象的最具体类型信息。

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

另外，也可替换 REF 列和集合元素。

### 动态方法分派

方法是作为对象类型定义一部分的过程或函数。

可以在 Oracle9i 的执行环境中运行方法，或将它分派到 Oracle9i 的执行环境外进行运行。可以采用包括 PL/SQL、C/C++ 和 Java 在内的各种语言执行方法。

**Oracle 支持多语言动态方法分派功能。方法被调用到对象实例时，根据该实例的最具体类型将方法分派到特定实施中。**

子类可以覆盖其超类中定义的任何非最终成员方法，并提供不同的实现。可以采用与子类中覆盖方法不同的语言执行超类中的方法。Oracle 支持多语言动态方法分派功能。

方法被调用到对象实例时，根据该实例的最具体类型将方法分派到具体实现。

### 客户机环境

包括 PL/SQL、Java 和 C/C++ 在内的各种客户机环境中都可以全面支持类型继承。通过 JDBC 和 OCCI（即 Oracle C++ Call Interface）等 API 可以访问和处理子类实例。

Oracle 也提供了 JPublisher 和 Object Type Translator(OTT) 等工具，以便从数据库对象类型层次生成各自的 Java 和 C++ 映射。这些语言环境中也支持实例和 REF 可替换性。

## 集合类型

集合是包括多个元素的 SQL 数据类型。集合的每个元素或值具有相同的可替换数据类型。在 Oracle 中，有两个集合类型，可变数组和嵌套表。

可变数组含有不定数目的有序元素。可变数组数据类型可作为表的列或作为对象类型的属性。

使用 Oracle SQL，可以创建（命名的）表类型。这些可以作为嵌套表，提供无序集合的语义。与可变数组一样，嵌套表类型可用作表的列或用作对象类型的属性。

### 多层集合

Oracle9i 在集合中支持多层嵌套，例如在嵌套表或可变数组中嵌入嵌套表或可变数组。

“Oracle 通用模式”中有一个理想的示例，它利用了多层集合支持。在图 1 的 UML 图表中，*Customer* 类具有 *Order* 类的一个集合，而 *Order* 类又有 *Order\_Item* 类的一个集合。以下示例显示如何映射这些结构。它创建包含 *order\_typ* 类型的嵌套表集合的 *customer\_typ* 类型，而 *order\_typ* 类型又包含有 *order\_item\_typ* 类型的嵌套表集合。

```
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
);
```

“Oracle 通用模式”中的同一示例将为多层集合创建对象视图。

```
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;
```

## 引用类型

如果在 Oracle9i 中创建对象表或对象视图，则可以获得指向相关行对象的引用（或数据库指针）。引用对于建模关系和在各个对象实例间浏览非常重要（特别是在客户端的应用程序中）。

### 大型对象

Oracle9i 提供了大型对象（LOB）类型，以处理图像、视频剪辑、文档和其他类似形式的非结构化数据的存储需要。采用能优化空间利用和提供高效访问的方式存储大型对象。更明确地讲，大型对象由定位器和相关的二进制或字符数据组成。LOB 定位器内嵌地存储在其他表记录列中。在具有内部 LOB（BLOB、CLOB 和 NLOB）的情况下，数据可存储在单独的存储区域。但是，对于外部 LOB（BFILE），数据则存储在操作系统文件中的数据库外。

### 用户定义的Constructor

用户定义的Constructor可以避免attribute-value constructor所带来的问题，因为用户定义的constructor不需要显示地问类型的每个属性设置值。一个用户定义的constructor可以有任何数目的参数，可以是任何类型，而这些不需要直接映射到类型的属性。在你定义的constructor里，你可以将属性初始化为任何合适的值。没有初始化的属性由系统初始化为NULL。

### 类型别名

正如你可以创建数据表，视图和其它方案对象别名一样，你也可以创建自定义类型

的别名。类型别名具有其它方案对象别名一样的好处：提供不依赖位置的方法引用方案对象。一个使用公共类型别名的应用在任何方案中不需要替换就可以配置，不需要验证所使用的别名是否在当前方案中是否定义。

## 类型演变

类型演变是一种机制，它可使用户更改类型并将这些更改传送到引用已修改类型的其他模式对象。

使用类型，用户可以通过捕获类型的行为来演变商务逻辑。类型演变是一种机制，它可使用户更改类型，并将这些更改传送到引用已修改类型的其他方案对象。可以引用类型的方案对象包括其他类型、子类、行对象、列对象、程序单元（程序包、函数、过程）、视图、功能性索引或触发器。

以下为支持的类型演变操作：

1. 类型属性上的操作：
  - 为类型添加属性。
  - 从类型中删除属性。
  - 通过增加长度、精度或比例来修改属性的类型。
2. 类型方法上的操作：
  - 为类型添加方法。
  - 从类型中删除方法。
3. 更改 SQL 对象类型的 INSTANTIATABLE 和 FINAL 属性。
4. 支持显式传送类型更改至其相关的类型和表。

类型演变更改分为 *结构化或非结构化更改*。结构化更改影响对象状态（如添加或删除属性）。非结构化更改不影响对象的状态。例如，重新命名属性或添加方法。

如以下示例所示：

```
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;
```

该示例中，对于 person\_t 中的每个属性以及每个地址属性，都会创建带有列的 person\_tab。考虑以下类型更改示例：

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

NVARCHAR(20) CASCADE;

此更改的效果为：在与新添加属性相应的 person\_tab 表中添加了两个列。CASCADE 关键字将类型更改传送到相关的类型和表。

## 语言绑定

许多语言全面支持 Oracle 的对象关系类型系统，包括 PL/SQL，Java 和 C/C++。通过 API，如 JDBC 和 OCCI (Oracle C++ Call Interface)，可以访问和操作类型实例。Oracle 还提供象 Jpublisher 和对象类型转换器 (OTT) 等工具，从数据库对象类型树产生相应的 Java 和 C++ 语言。另外这些语言还支持实例和 REF 的替代。

## 在 SQL 和 JAVA 间映射对象

### JDBC、SQLJ 和 JPub

对象关系工具为在应用程序级别的 Java 类集合和数据存储级别的数据模型间维护一致的结构提供了更为自然和高效的方式。在 Oracle 中，对象关系工具通过使用标准 JDBC 和 SQLJ API 已经与 Java 环境紧密集成。SQL 对象类型可以映射为 Java 类。JPublisher (JPub) 实用程序自动生成实施此映射的 Java 类的 Java 或 SQLJ 文件。每个生成的 Java 类包括相关的方法，可以从数据库服务器中读取和向数据库服务器中写入对象状态。

并且，Java 应用程序可使用这些生成的 Java 类在数据库中存储和检索对象。以下为 JPub 生成的 Java 类的代码段。

```
// 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 {...}
...
}
```

然后，可在以下 Java 程序中使用此 Java 类 JpurchaseOrder，以从数据库中检索对象。

```
// 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;
```

在 Oracle 中，对象关系工具通过使用标准 JDBC 和 SQLJ API 已经与 Java 环境紧密集成。

## SQLJ 对象类型

Oracle9i 具有无缝地将指定 Java 类映射为 SQL 对象的能力。*SQLJ 对象类型* 遵照 SQLJ Part II 标准，它定义 Java 中 SQL 对象到其相应的实现类间的映射。SQLJ 对象类型正在被标准化为 SQL99 扩展。

如以下示例所示：

```
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');
```

以上示例中，CREATE TYPE 语句指定一个外部名称子句，该子句标识实现中使用的 Java 类。USING 子句指定用于持续对象状态的接口。对象类型的每个属性具有一个可选的 EXTERNAL NAME 子句，以指定它们相应的 Java 字段。对于方法，EXTERNAL NAME 子句指定相应的 Java 函数名及其签名。

上述映射信息作为类型元数据的一部分存储。这可通过 JDBC 从 Java 无缝地访问 SQL 对象，而不需要用户显式地实现 SQLData 接口或使用 Java 类型图注册类。

SQLJ 对象类型层次可映射为 Java 继承层次。Java 运行时执行方法调度（包括这些类型的覆盖）。

## 将 SQL 对象映射到 C++

**Oracle C++ Call Interface (OCCI)** 指定对象类型的基于 C++ 的映射，从而可以访问和修改数据库中的对象类型实例至应用程序中的 C++ 对象，也可以从应用程序中的 C++ 对象访问和修改数据库中的对象类型实例。

Oracle C++ Call Interface (OCCI) 指定对象类型的基于 C++ 的映射，从而可以访问和修改数据库中的对象类型实例至应用程序中的 C++ 对象，也可以从应用程序中的 C++ 对象访问和修改数据库中的对象类型实例。OCCI 可导航的接口允许象 C++ 对象那样访问和修改对象关系数据，而不用使用显式 SQL。Object Types Translator (OTT) 生成对应于对象类型（包括类型层次）的默认 C++ 类。它也为从/向数据库中读取、写入对象数据提供了默认的方法实现。例如，考虑以下对象类型层次：

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

```
Person);
```

以下是 OTT 为上述类型层次生成的 C++ 类。OCCI API 提供的 OCCIPObject 类作为具有永久或临时对象的类的基类。

```
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); ...}
```

该代码段对应于以下给出的几个方法：

```
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); }
```

类型映射表维护 SQL 类型名和 C++ 类名间的关联，并用于在运行态时确定需要调用其 readSQL/writeSQL 例程的类。OTT 也为用户提供了灵活性以扩展生成的类，从而增加更多的功能。

以下代码段说明在 C++ 类实例中检索 SQL 对象：

```
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();
```

## 映射 SQL 对象至 XML

XML 是一种元标记语言，它正迅速成为在商务和应用程序间交换数据的事实上的标准。XML 模式根据数据类型指定 XML 文档的结构，以及文档中每一元素的构成。Oracle 支持用于将 SQL 对象映射为 XML（反之亦然）的有效和灵活的机制。

**Oracle 支持用于将 SQL 对象映射为 XML（反之亦然）的有效和灵活的机制。**

## Oracle XML DB

Oracle 9i Release 2 引入了 Oracle 9i XML 数据库 (Oracle XML DB)。通过将 XML 数据和内容模块直接联系到 Oracle 9i 应用, 扩展了 Oracle 数据库内在对 XML 的支持。Oracle XML DB 基于 W3C XML 数据模型, 提供对 XML 数据的高性能存取技术。它结合了两种技术的优势

- 对象关系数据库技术
- XML 技术

Oracle XML DB 是一个内嵌的 XML 数据库。提供了存储独立, 内容独立和开发语言独立的体系结构来管理数据。它和 Oracle SQL 引擎紧密结合, 带来了以往关系数据库所没有的崭新概念。例如, 它将 XML Document Object Model (DOM) 建在 Oracle 的核心, 因此对 XML 数据的操作可以是普通数据库处理的一部分。这就避免了分两步来取模型和处理模型。通过将 Oracle 对象关系技术和 PL/SQL 紧密结合, Oracle XML DB 同时结合对象关系数据库和 XML 的灵活性。

## XML 生成

DBMS\_XMLGEN 程序包可以包括任意的 SQL 查询, 并将查询结果映射为 XML。这样, 从查询获得的整个结果集将转换为单一 XML 文档。程序包有几个参数可以控制 XML 生成的不同方面, 包括: 生成与文档相关联的 XMLSchema (类似于描述功能), 以及限制检索行的数目。例如, 以下代码段将以 XML 形式检索查询的结果, 如下所示:

```
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>
```

采用关系数据, 结果成为无嵌套的纯 XML 文档。若要获得嵌套的 XML 结构, 可以使用对象关系数据。对象映射为 XML 元素, 同时对象的属性映射为父元素的子元素。集合实例也可映射为 XML 元素, 集合元素显示为重复出现的子元素。

## XML 存储

为了便于本地存储以及访问 XML 文档，引入了一种新的数据类型：*XMLType*。用户可以创建此数据类型的实例和列，并使用类型方法提取、传送和转换 XML 文档。

*XMLType* 列可采用打包的形式(如 CLOB)或采用分离的对象关系方式存储文档。它提供操作符来查询 XML 文档：使用 *extract()* 根据路径表达式遍历和提取文档段，使用 *existsNode()* 检查是否有满足给定条件的节点存在。用户也可在此列上定义 *interMedia* 文本索引，并使用 *Contains* 和其他基于文本的操作符查询 XML 文档。

```
-- 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;
```

## 对象关系技术的 JDeveloper 支持

Oracle 为支持面向对象应用程序开发提供了功能强大的工具。对于 Java 开发人员，JDeveloper 为开发利用 Oracle 对象关系技术的应用程序提供了两类集成的支持。其中一类使用集成的 Oracle Business Components for Java 框架，将域映射为 Oracle9i 对象类型，并将实体对象映射为对象表和对象视图。使用 Container Managed Persistence (CMP)，可将实体对象部署为 Enterprise Javabeans (EJB)。另一类支持使用 JPublisher 向导从 Oracle9i 对象类型生成 Java 类。开发人员使用从 JPublisher 向导生成的 Java 类访问和操纵 Oracle9i 行对象和列对象。

## Oracle Business Components for Java

Oracle Business Components for Java 框架将它的域映射为 Oracle9i 对象类型，并将它的实体对象映射为对象表和对象视图。JDeveloper 向导可从 Oracle9i 对象表或对象视图创建实体对象。使用 Container Managed Persistence 可将实体对象部署为 EJB。

# Container Managed Persistence 使用 Oracle9i 对象

Oracle9i 使用数据库表管理 EJB 实体组件的可持续性。实体组件的每个实例对应于表中的一行，每个 CMP 字段对应于表中的一列。实体组件类也需要主键（它对应于表的一个或多个列），它允许使用 `findByPrimaryKey()` 方法检索实例。

与表对应的是两个 Business Components for Java 组件：

- 1、EJB/9i 部署对象，它与标准实体对象很相似，并支持管理可持续性的表
- 2、视图对象，它选择表的相关的列，并在 EJB 字段中为它们指定别名

CMP 组件也可使用其他商务组件类：域

它允许 EJB 字段基于 Oracle 对象类型和对应于 EJB 查找程序方法的辅助视图对象。

## 映射 Oracle 对象类型至 CMP 字段

管理 CMP 可持续性的数据库表可以包括含有 Oracle 对象类型的列。在 JDeveloper 中，“域向导”可以帮助从 Oracle 对象类型创建域对象。

## JPublisher 向导

JDeveloper 的“数据库浏览器”可以帮助开发人员浏览他们数据库模式的内容，以及定位 Oracle9 对象。可以从此接口调用 JPublisher 向导，生成 Oracle9i 对象的 Java 绕接器类。

通过自动生成合适的 Java 类定义，JPublisher 使得在 Java 程序中使用 Oracle9i 对象变得更加容易。JPublisher 生成的 Java 绕接器类包括用来在 SQL 和 Java 间相互转换数据的方法，以及用于对象属性的 getter 和 setter 方法。JPublisher 提高了开发人员的生产率，同时仍能灵活地扩展生成的类以满足自定义的需要。

使用 JDeveloper 开发完 Java 应用程序后，可将它部署到组合了 Oracle9i 数据库服务器和 Oracle9i 应用服务器的各种目标体系结构，Oracle9i 应用程序服务器是用来处理高需求应用程序的通用软件平台。

## 部署面向对象应用程序至 Oracle Internet 平台

使用 Oracle9i 对象关系技术完成应用程序开发后，有很多方法可以部署和管理应用程序。Oracle9iAS（即 Oracle9i Application Server — Oracle9i 应用服务器）和 Oracle9i Database Server（Oracle9i 数据库服务器）为部署和管理电子商务应用程序提供了所有必需的一切。

Oracle9i AS 和 Oracle Database 一起构成一个简单、完整和集成的 Internet 平台。

- **简单** Oracle9iAS 和 Oracle Database 的购买、安装和管理都很简单。Oracle 的所有核心中间层服务都已集成到 Oracle9iAS 中，可以使客户通过单一产品建立和部署门户网站、事务处理应用程序以及商务智能工具。Oracle9iAS 的全套服务通过单一安装进行系统外的集成，且通过单一管理工具管理 Oracle9iAS 和 Oracle Database。
- **完整** 采用 Oracle9i Database 管理数据以及 Oracle9iAS 运行应用程序，Oracle Internet 平台成为将任何类型应用程序建立和部署到 Web 的完整解决方案，它包括内容管理、OLTP、移动功能、商务智能以及企业集成应用程序。Oracle9i AS 和 Oracle9i Database 提供了具有可伸缩性和高可用性的基础架构，这可使客户方便地适应不断增长的用户数量，而不用对他们的应用程序重新编写代码。
- **集成** Oracle9iAS 完全可以称得上 Oracle9i Database 的最佳应用程序服务器。通过利用通用技术堆，Oracle9i AS 可以通过在中间层高速缓存数据和应用程序逻辑，透明地扩大 Oracle9i Database 的规模。另外，Oracle9iAS 从 Oracle9i 的成熟技术中继承了它的很多强健的可伸缩性和可用性特性。

## 总结

Oracle 的对象关系技术经过数年的发展已经十分成熟，它提供了完整的对象类型系统、广泛的语言绑定 API 以及丰富的实用程序和工具集。

Oracle 的对象关系技术经过数年的发展已经十分成熟，它提供了完整的对象类型系统、广泛的语言绑定 API 以及丰富的实用程序和工具集。这一完整的对象类型系统基于最新的 ANSI SQL-99 标准。Oracle 为面向对象的应用程序在数据库服务器中优化了对象类型的性能。Oracle 在 Java、C/C++ 和 XML 中的语言绑定 API 提供了到数据库服务器对象类型系统的直接接口。这些广泛的 API 支持最新的标准，可以访问数据库对象类型系统服务。附带的对象关系数据的丰富实用程序集包括导入/导出、SQL 加载器、复制等。

另外，为开发面向对象的软件也提供了很多支持 Oracle 对象关系技术的开发工具，如 Oracle JDeveloper BC4J 以及 Oracle 合作伙伴提供的其他产品。Oracle 也为应用程序部署和管理提供了高性能、强健的以及可伸缩的 Internet 平台，即 Oracle9i Database 和 Oracle9i Application Server。瞬息万变的 IT 行业需要实用的解决方案，解决处理各种数据类型的复杂应用程序的问题。Oracle 的对象关系技术凭借最全面的解决方案，为开发、部署和管理这样的应用程序解决了这些问题。Oracle 一直以来在对象关系技术方面处

于领先地位。Oracle 将以最佳的对象关系技术继续满足我们的合作伙伴和客户的需要。



Oracle9i 对象  
2002 年 1 月  
作者: Geoff Lee  
协作者: Sandeepan Banerjee, Vishu Krishnamurthy

Oracle 公司  
全球总部  
500 Oracle Parkway  
Redwood Shores, CA 94065  
U. S. A.

全球咨询热线:  
电话: +1. 650. 506. 7000  
传真: +1. 650. 506. 7200  
[www.oracle.com](http://www.oracle.com)

Oracle 是 Oracle 公司的注册商标。 此处引用的各种产品和服务名称可能是 Oracle 公司的商标。  
提及的所有其他产品和服务名称可能是它们相应所有者的商标。

版权所有 © 2002 Oracle 公司  
保留所有权利。