

## 第12章 面向对象的特性

本章要点：

背景

面向对象技术

Oracle 8对象选项

REF属性

方法

集合——变长数组与嵌套表

对象视图

### 12.1 背景

面向对象的DBMS（OODBMS），也就是那些非主流的关系型（RDBMS）的DBMS，是面向对象程序语言（如 Actor和SmallTalk）的直接继承。永久存储的需要是促使 DBMS和RDBMS的出现来替代第三代语言（3GL）/基于文件的信息系统的最基本的原因之一，也促使OODBMS的产生来替代面向对象的编程语言信息系统。很有趣的是，喜爱面向对象系统的人通常认为，面向对象系统比关系数据模型的派生物（如实体关系图的变体）提供更有力的语义模型，并具有更高的层次。然而，与此同时，反对面向对象系统的人认为，由于它们关注3GL/基于文件的信息系统，它们的实现和语言要素偶尔会下降到十分低的层次（按 C/C++指针方式）。从本质上讲，它们采取导航的或过程化的查询，这也与前关系型系统（如基于COBOL的分层和网络DBMS）十分相似，而在基于SQL的RDBMS系统中的查询大多是描述性的和声明的。

不能不考虑关系型的厂商（如 Oracle）因此考虑 Oracle 8和它的当前关系-对象型的实现。当用任何RDBMS厂商的产品时，为了估计它的面向对象的能力，必须考虑几件事情，如下所示：

对面向对象概念和结构的信任。

面向对象到关系的映射的概念和结构。

仿真、完全或混合实现。

性能的降低或加强。

使用现有RDBMS子系统的能力。

希望RDBMS具有面向对象的接口的一个主要的原因是，这样做面向对象应用的程序就可以与RDBMS中的面向对象部分直接进行通信，这与在程序代码中必须动态地处理对象-关系型映射（分解和重组）相反。作为一个例子，考虑嵌入式SQL。这里SQL是基于集合的语言，必须与一个基于记录的主机3GL（如C）交互。这是不希望的情况，称为阻抗失配，可以通过诸如游标的结构和诸如预编译的软件（例如 Pro\*C）来进行补救。一个基于面向对象的程序语言的应用与一个RDBMS的交互，情况与此相似。

Oracle 8特别地帮助缓解面向对象的开发和 RDBMS后端的情况，方法是提供一些强有力的内建的面向对象能力，如下所示：

关系作为数据类型。

继承。

集合作为数据类型，包括嵌套（容器）。

用户定义（可扩展的）数据类型。

改进大对象（LOB）。

和如用任何OODBMS或对象-关系DBMS（ORDBMS）一样，使用Oracle 8时，你能利用面向对象模型的优点。面向对象模型是比关系型模型更高层次的抽象。换句话说，你能抽象出事物的模型作为一个整体来比较接近地反映“真实的”世界。例如，在你的OODBMS中，可以具有一个称为CAR的复杂对象，它很可能是RDBMS中的几个表，如MAKE、MODEL、ENGINE、PARTS等等。面向对象模型与关系模型在“自然性”方面的注释是不同。我们来打个比方：

用面向对象技术，就像你在早上醒来穿上你的衣服，在夜晚你简单地脱掉它们以供下次再穿，十分自然。但是用关系型技术，你必须为你的衣服收集所有的纺织物，将它们缝在一起，然后穿上；在夜晚，必须把它们分开，按次序保存它们以供下次再缝和再穿。

尽管这个比喻不完全准确，却有些道理。基本思想是，用RDBMS进行工作时必须连结在前一次通过规范化分开的部分。关系设计者知道这种情况，这就是这类事情作为非规范化而存在的原因。

面向对象技术有哪些优点，特别是与关系型技术有关的并且是通常的优点呢？有些比较重要的已经讨论过了，而这里有一个列表，列出了大多数面向对象的主要强度：

建模：业务级抽象。

重用性：生命周期的代码重用可以在很大程度节约成本。

复杂性：没有数据类型限制；多媒体数据；嵌套。

可扩展性：反映建模的用户定义数据类型。

性能。

正如前面概括的，Oracle 8通过提供特定的能力如继承、集合、容器和用户定义数据模型满足了这些需要中的很大一部分。然而，性能是难以决定的，更多地依赖于整个数据库物理的和逻辑的设计。但是适当地应用分割并配合实验、迭代测试（基准程序）通常伴随性能调整效果，能够帮助你判断是Oracle 8使用面向对象还是只用Oracle 8能够更好地满足你的特定应用的性能需要。

## 12.2 面向对象技术

面向对象技术的一个主要问题是在面向对象团体中总是缺少一个标准定义，也就是什么是对象。相比之下，关系技术实质上对于什么是关系，在关系团体中从没有任何争论。

无论如何，Oracle 8定义对象作为对象类型的实例，一个对象类型作为基本面向对象模型的组成块通常直接指向真实世界的业务项目。因此，类似于表中的行，对象是数据，而对象类型是结构，与其他数据结构（例如记录）很相似。在许多其他面向对象语言和系统中，Oracle对象类型等于对象类。一个Oracle 8对象类型有两部分：

属性——基本的数据类型或其他对象类型，有时称为对象类型的结构化部分。

方法——组成在对象类型上允许的集合操作的 PL/SQL（或 C）子程序；有时称为对象类型的行为部分。

## 多态性和继承

在面向对象技术中，其他的重要概念包括继承和多态性。继承是一个对象包括来自另一个对象类型的属性或方法的能力。方向是从一般到特殊，正像关系型模型中的子类型。一个例子是，你可能有一个雇员对象类型和一个经理对象类型（雇员类型的子类型）。所有雇员有名字、地址、社会保险号等等，还有雇佣日期、薪水、雇佣比例等等。所有前面这些代表一般的或普通的属性，能从雇员对象类型继承到经理对象类型。这意味着有两件事与效率有关：一、所有雇员（包括经理）的共同属性和方法只需要存储一次；二、经理对象类型仅需要存储它自己特有的属性和方法，如 `signature_authority`（属性）或 `rate_an_employee`（方法）。因此继承可以基于属性（结构）或方法（行为）。

刚刚被描述的情况（经理对象模型从雇员对象中继承属性和方法）称之为单继承。有另一种类型的继承称之为多继承。单继承是一个子类最多有一个父类的形式；多继承允许一个更一般的模型：网络（或格子）模型，也就是子类型能够有多个父类，像在真实世界中的情况一样。与雇员到经理的单继承对比，考虑也包含官员的多继承方案：雇员 经理 官员。箭头特指继承方向，从一般（父）到特殊（子）。

多态性像继承一样可以是结构的或行为的。考虑一个没有经理对象类型的雇员对象类型。雇员只简单地是一个雇员，而不是经理。你如何从一个雇员类型获得经理类型呢？事实上，这是一个旧的程序结构，在不同的编程语言中有不同的术语，但在这里为了全面将它描述为变量记录。一个变量记录依赖于被输入的记录类型能够包含一些共同（继承）的数据，也能包含一些附加的域。这种情况对非规范化表是模棱两可的，可能同时包含经理和雇员。方法是怎么样的？与属性类似，方法是否可见依赖于被具体说明的对象（雇员或经理）。

刚才描述了常见的结构多态性是什么。那么行为多态性是什么？在面向对象中，当一种方法被调用时，称为向它发送了一条消息（进行执行）。当你向方法发送一条消息时，它可能有不同的执行，取决于它当前属于的对象类型或被影响的实例。例如，如果你调用一个 `hire` 方法，它构造（创建）一个新的职员或经理。尽管你调用的是同一个 `hire` 方法，这里实际上有构造一个实例（实例化一个对象）的两种不同实现：通过为雇员填入共同雇员属性，或通过为经理填入共同雇员属性和特定的经理属性。面向对象系统能够通过简单 IF/THEN 逻辑在单个程序内或通过定向化实现这个变化的程序，定向是 IF/THEN 逻辑加一个对所有可能程序之外的一个必要程序的调用。多态性也称之为重载。稍后捆绑（运行时捆绑）使多态性在现代语言和系统中成为可能。在面向对象中，每一个对象类型总是至少有一种方法——它自身的构造方法。

另外一个重要的概念是对象 ID（OID），它表示一个在对象创建以后唯一代表该对象的统一标识符。这意味着它在所有的，甚至在 Oracle 之外的面向对象系统和语言中是唯一的。在 Oracle 8 中产生 OID 是 Oracle 公司的一个秘密，但是唯一性好像是以某种方式被链接到 CD-ROM 介质上。对那些熟悉本地网络的以太网的人来说，OID 应该是唯一的，就像以太网卡的以太地址一样，在工厂中被固化。ID 在它自身系统之外也是唯一的概念，与关系模型中

的主键和代理键这些东西是不同的。然而，它在某种程度上像一个分布式数据库中的主键，它们必须在多个系统中是唯一的。

你希望Oracle 8或任何ORDBMS中有的东西是什么？你可能希望一些东西，如组合（集成）多个关系表的对象视图。除了你已经包括的定制的（用户定义）数据外，你可能希望有一些预创建的、综合的、业务的数据类型可以服务于特定的适当范围的需要。客户端对象缓存也可能是需要的，因为它可以提供性能和访问的灵活性。你也不想失去 RDBMS 的强项：一致性、并发性和安全性。

### 12.3 Oracle 8对象选项

那么，通过面向对象技术，Oracle 8给了你什么？Oracle在如下方面扩展它已有的复杂的多用途的RDBMS如下所示：

对象类型——本质上是记录或类。

对象视图——把你的许多规范化的表放在一起形成一个项目。

对象语言——对Oracle SQL和PL/SQL语言的扩展。

对象API——通过Oracle预编译器（例如Pro\*C）支持的对象。

对象可移植性——通过对象类型翻译器（OTT），它可以建立Oracle 8对象类型到C++类的接口这样的东西。

但是，即使有了这些进步，Oracle 8 不支持多继承、多态性或在对象属性（如引用一致性）上的约束。

Oracle 8开放式类型系统（OTS）是一个所有Oracle 8对象类型的知识库，也是其他语言或系统中的外部对象类型的知识库。在开放式类型系统中，有一个数据类型层次，作为它的底层的是内建的Oracle 8数据类型。Oracle 8还增加了大对象（参考第11章“大对象的历史”），形式是BLOB、CLOB、NCLOB和BFILE，还增加了VARRAY（变量数组）和嵌套表形式的集合类型和REF（引用）形式的对象ID。另外，用户定义数据类型能够被创建在任何内建的数据类型和已有的用户定义数据类型上，唯一的例外是它们不能被创建在LONG、LONG RAW、ROWID或%TYPE上。当你在Oracle 8中创建用户定义的对象类型后，可以将它用于多种用途，包括：

作为关系型表中的一列。

作为另一个对象类型的一个属性。

作为关系型表的对象视图的一部分。

作为对象表的基础。

作为PL/SQL变量的基础。

用于管理对象类型的扩展 Oracle SQL，包括以下这些：

CREATE TYPE

ALTER TYPE

DROP TYPE

GRANT/REVOKE TYPE

和表的列一样，属性在一个对象类型中必须唯一，但是在其他对象类型中该属性名可以被再用。与表中的列一样，你总能在一个对象中通过它的名字访问其属性。没有能力进行位

置或偏移量访问。任何给定的对象类型可以是简单的、复合的或者自我引用的。简单对象类型就只是它自身；复合对象类型至少包括一个其他对象类型；自我引用对象类型至少包含一个引用对象类型自身的属性。清单 12-1和清单 12-2给出了两个创建和使用对象类型的例子。

清单 12-1 创建一个雇员对象类型

```
SQL> create type employee_type as object (  
1 ssn number(9),  
2 name varchar2(35),  
3 address varchar2(70),  
4 resume CLOB );
```

清单 12-2 使用一个对象类型作为关系型表中的一列

```
SQL> create table manager_candidates (  
1 position varchar2(40),  
2 vacancy CLOB,  
3 employee employee_type );  
  
SQL> insert into manager_candidates values (  
1 'Technical Manager',empty_clob(),  
2 employee_type(123456789,'John Doe','123 Maple St :  
3 Metropolis : NY : 12762',empty_clob() ) );  
  
SQL> select mc.employee.name from manager_candidates mc  
1 where mc.employee.ssn = 123456789;
```

注意你必须使用一个别名和点符号来查询一个表中的对象属性值。同时还要注意当你进行插入时，为命名为 `employee_type` 的对象类型调用命名为 `emplpy_type` 的构造器方法。所有的构造器具有与它们相关的对象类型相同的名字。清单 12-3给出了另外一个使用对象类型的例子，这次是创建一个对象表。

清单 12-3 创建一个对象表

```
SQL> create table employee of employee_type;
```

创建这个表也为一个对象实例（行）创建一个包含对象ID（OID或REF）的列。这与创建一个关系型表，这个表在一个列中具有一个对象类型形成对照。在关系表中的对象类型没有OID。

## 12.4 REF属性

在Oracle 8对象世界中的关系与在Oracle关系世界中的关系实际没有什么不同：它指明对象之间如何相关。Oracle8仅支持一对一单向的关系，它可能足够也可能不够将你的业务模型化。当你声明一个对象类型中的属性是 REF时，它代表对其他一些对象类型的引用，这是一个最基本的关系。REF代表了一个对象的关系。一个 REF本质上是一个指针，或一个对象的“句柄”。你可以限制一个REF实际指引的范围，在SQL 3中包括范围的概念。SQL 3是对ANSI SQL标准的最新版本，它包含一些面向对象的特性。

像RMT（关系型模型2）中的代理键一样，Oracle 8中的对象引用（REF）是一个系统产生的值，在所有对象中（任何地方）是唯一的，指向一些永久不变的对象。一个 REF实际上引用一个对象——对象表中的一行（实例），而不是真正的对象类型。一个给定对象的 REF是

对象的OID+对象的表ID+对象表的数据库ID的组合。清单 12-4显示如何创建一个包含引用已创建的employee\_type的REF属性的对象类型，并创建基于这一个对象类型的对象表，再插入一行（实例化一个对象）。

清单12-4

```
SQL> create type insured_employee_type as object (  
    1 control_number number(9),  
    2 contract CLOB,  
    3 employee_ref ref employee_type );  
  
SQL> create table insured_employee (  
    1 of insured_employee_type  
    2 scope for (employee_ref) is employee );  
  
SQL> insert into insured_employee  
    1 select control_sequence.nextval,  
    2 empty_clob(),  
    3 ref(e) from employee e  
    4 where e.ssn=123456789;
```

## 12.5 方法

一个对象类型可以有零个或多个成员方法。一个成员方法是一个能够操作任何对象类型（不仅仅是定义它的那个类型）的数据（也就是属性）的子程序。当然，方法简单说就是PL/SQL过程或函数。

正像使用任何PL/SQL封装的子程序一样，应为每一个方法创建一个接口（声明）和一个实现（体）。当访问一个方法时，应使用点表示法（对象.方法）。清单 12-5给出在一个对象类型内创建方法（声明和体）的例子，跟随其后的是调用该方法的一些 PL/SQL代码。

清单12-5

```
SQL> create type auto_type as object (  
    1 vin varchar2(80),  
    2 make varchar2(30),  
    3 model varchar2(40),  
    4 member function getcost  
    5 return number,  
    6 pragma restrict_references  
    7 (getcost, WNPS) );  
  
SQL> create type body auto_type (  
    1 member function getcost  
    2 return number is  
    3 begin  
    4 return (cost);  
    5 end; );  
  
/* create object table and insert some data */  
  
SQL> declare  
    1 a auto_type;  
    2 c number;  
    3 begin  
    4 select value(a)  
    5 into a
```

```
6  from auto a
7  where a.vin=1VWF673FB9093871AF21;
8  c:=a.getcost();
9  /* do something with c */
10 end;
```

注意函数使用pragma指令设置为WNPS。WNPS代表“写无包状态”，它的意思是函数被设置为不能修改自身内的任何参数（否则产生负面影响）。为了比较或排序对象类型，你能定义一个MAP或一个ORDED函数来比较两个对象。当使用这些比较函数时，你指定了对象分类语法（也就是，一个整理的序列），用于条件和分类。请参考Oracle 8或Oracle 8i CD-ROM文献以获得更详细的实现信息。

## 12.6 集合——变量数组和嵌套表

集合是排序的或未排序的一组事物，变量数组是排序的，而嵌套表是未排序的。嵌套表是在最新的ANSI标准中，但变量数组是非标准的。任何主从（也就是一对多）关系能帮助对比这两种类型的集合。用某种顺序列出的主表项目能够使用变量数组（VARRAY），对每一个主表行的每行项目细节可以在变量数组中使用嵌套表。

**警告** 并行查询、并行DML、复制和分布数据库函数都不支持集合。

和许多数组一样，Oracle 8变量数组有一个隐式的顺序。每一个元素在数组中有一个从数组起始位置开始的偏移量，能够通过偏移量直接进行访问（也就是所谓的下标或索引）。变量数组是按行存储的（在表的行中），直到存储单元超过4KB（该点被存储在数据库之外）。和许多数组一样，每一个变量数组有数量和限制。数量是当前有多少元素存储在变量数组中，而限制是能够存储在变量数组中的最大元素数。你能通过在PL/SQL和3GL中使用下标访问单个元素，但是不能在简单的SQL中访问它们。

嵌套表能很好地适合主从（一对多）关系。嵌套表在Oracle 8中实际上是用户定义类型或表类型，能够在一个表作为一列使用，在对象类型作为属性，或者作为PL/SQL变量。嵌套表被使用STORE AS语句存储在主表之外的存储表中。嵌套表有一些优点和缺点。优点包括能够对它们进行索引（与对象表相反）和不需要连结（与簇类似）。一个缺点是，尽管有级联删除形式（主行删除时将删除所有从属行），但是引用完整性约束是不可能的。清单12-6中给出一个创建和使用变量数组和嵌套表的例子。

清单12-6 创建和使用变量数组和嵌套表（表类型）

```
/* VARRAY */

SQL> create type tax_type as object (
    1 year date,
    2 taxes_paid number );

SQL> create type tax_array_type as VARRAY(3)
    1 of tax_type;

SQL> create type client_type as object (
    1 cid number,
    2 name varchar2(35),
    3 address varchar2(80),
    4 taxes tax_array_type );
```

```
SQL> create table clients of client_type;

/* nested TABLE */

SQL> create type child_type as object (
    1 child_id number,
    2 child_name varchar2(35) );

SQL> create type child_nest_type as TABLE
    1 of child_type;

SQL> create table parent (
    1 parent_id number,
    2 parent_name varchar2(35),
    3 parent_address varchar2(80),
    4 children child_nest_type )
    5 NESTED TABLE children store as child_nest_store;
```

总的来说，当你已在程序中使用数组时使用 VARRAY 集合，当你已在程序中使用一个记录（在一个记录里）时使用嵌套表集合。对这些程序数据结构来说，这些 Oracle 8 集合结构是模糊的。

## 12.7 对象视图

关系型表的对象视图提供给用户和开发者许多好处，如建立面向对象的前端应用到关系的后端存储之间的联系、协调多个面向对象的前端模式为一个关系型的后端模型和并行开发新的面向对象的应用和当前的关系型应用。后面这一项是对象视图方便地移植到面向对象技术的一种说法。对于用户，整个系统从表面上看似是基于面向对象的，但是实际上数据是关系型的。然而，这是透明的。当然，一般地说因为对象视图是视图的特例，它们有与一般视图相同的优点：为不同用户和不同目的提供相同数据的不同视图。正确地创建对象视图的一般顺序如下：

创建表。

创建对象类型。

创建对象视图。

创建 INSTEAD OF 触发器。

INSTEAD OF 触发器使你可以插入、更新或删除对象视图所基于的关系型表，而不是直接试图修改对象视图（它不能够这样做）。

清单 12-7 中给出一个创建对象视图的例子。

清单 12-7 创建对象视图的步骤

```
SQL> create table depts (
    1 deptid number,
    2 deptbudget number );

SQL> create table branches (
    1 branchid number,
    2 branchbudget number,
    3 deptid number );

SQL> create type funding_type as object (
    1 deptid number,
    2 deptbudget number,
```

```
3 branchid number,
4 branchbudget number );

SQL> create or replace view funding_objv as

1 select funding_type
2 (deptid,deptbudget,branchid branchbudget) funding,
3 from depts d, branches b
4 where d.deptid = b.deptid;

SQL> create to replace trigger funding_trig INSTEAD OF

1 insert on funding_objv for each row
2 begin
3 insert into depts values
4 (:new.funding.deptid, :new.funding.deptbudget);
5 insert into branches values
6 (:new.funding.branchid, :new.funding.branchbudget,
7 :new.funding.deptid);
8 end;
9 /
```

---