第 11 章

记录类型

记录类型是一种复合的数据结构，也就是说是由多个元素或者成员组成的，每个成员都有自己的值。PL/SQL 编程中的记录类型无论在概念上还是结构上都非常类似于数据库表的行。记录作为一个整体本身并没有值；不过每个单独成员或者字段都有值，记录提供了一种把这些值当做一组进行操作的方法。记录类型可以让程序员生活更简单，通过把原来在字段级别的声明和操作转换成在记录级别进行，我们的代码编写和维护工作都更有效。

11.1 PL/SQL 中的记录

表中的每行都有一个或多个不同数据类型的列。类似的，一个记录也是由一个或者多个字段组成的。一共有 3 种不同方式可以定义一个记录，不过一旦定义了记录，使用记录中的字段或者修改记录中的字段就都要遵循相同的规则。

下面这个代码块演示了根据一个底层数据表直接声明的一个记录类型。假设我定义了一个表用来记录我最喜欢的图书：

```
CREATE TABLE books (
  book_id        INTEGER,
  isbn           VARCHAR2(13),
  title          VARCHAR2(200),
  summary        VARCHAR2(2000),
  author         VARCHAR2(200),
  date_published  DATE,
  page_count      NUMBER
);
```

接着我就可以很容易地根据这个表创建一个记录，并用一个查询来填充记录里的数据，再通过记录的字段来访问每一列：

```
DECLARE
   my_book  books%ROWTYPE;
BEGIN
   SELECT *
     INTO my_book
     FROM books
     WHERE title = 'Oracle PL/SQL Programming, 5th Edition';

   IF my_book.author LIKE '%Feuerstein%'
   THEN
      DBMS_OUTPUT.put_line ('Our newest ISBN is ' || my_book.isbn);
   END IF;
END;
```

我也可以定义自己的记录类型并用作声明记录的基础，比如，假设我只需要图书的作者和标题，与其通过%ROWTYPE 来声明记录，还不如创建一个记录类型：

```
DECLARE
   TYPE author_title_rt IS RECORD (
      author books.author%TYPE
     ,title books.title%TYPE
     );
   l_book_info author_title_rt;
BEGIN
   SELECT author, title INTO l_book_info
```

```
FROM books WHERE isbn = '0-596-00977-1';
```

我们一起看一下使用记录所带来的好处。接着我会检查不同记录定义方法的细节，最后通过一个使用记录的例子来结束。

11.1.1 使用记录的好处

记录作为一种数据结构，为在 PL/SQL 程序中对数据（相对于数据库表而言）的访问和操作提供了一种高级方式。这种方法有许多好处，接下的章节会描述。

数据抽象

在你对某个事物抽象化时，你是在对它进行概括总结，从而可以远离琐碎的细节进而关注的一个更大的视角。当你创建模块时，你是把模块中的每个行为抽象成一个个名字。这些名字（程序的规范）就代表了这些行为。

当你创建了一个记录时，你是把这个记录所代表的主体的不同属性或者字段进行了抽象化，你为这些属性建立起关系，并通过定义一个记录来给这些关系起了个名字。

操作集合

一旦你把数据保存到记录中，你就可以在同一时间内对整块数据进行操作了，而不再是对一个一个属性的操作。这种集合操作增强了记录的抽象程度。更多时候你真正感兴趣的不再是改变一个记录中的单独成员，而是代表了所有这些成员的

对象本身。

假设我的工作是处理企业信息。我其实根本不关心企业的地址信息到底是两行或者三行记录；相反，我更希望在企业自身这一级别进行工作，包括修改、删除或者分析企业的状态。在所有这些处理中，我所谈论的都是数据库中的一整行，而不是某一列。企业的记录把所有信息封装了起来，但又触手可得。这种定位是为了让你把数据看做对象集合，并对这些对象应用规则。

简洁干净的代码

使用记录类型可以有助我们写出更干净更简洁的代码。如果使用的是记录类型，我一定会生成更少量的代码、不惧修改的、更少注释的代码。记录也可以减少变量的肆虐，不再需要那么多个单独的变量，只需要一个记录就可以了。因为代码不再杂乱无章而是赏心悦目了，需要的维护成本也更少。

无论是在最开始的开发阶段还是持续的维护过程，使用 PL/SQL 记录类型都会对你的编程有巨大的积极应用。为了能从记录这种数据结构中获得更多好处，我给自己的代码开发工作制定了如下的指导原则：

创建互相匹配的游标和记录

每当我在程序中创建了一个游标时，我都会相应的创建一个匹配的记录类型（只有对于游标型 FOR 循环是个例外）。对于 FETCH 操作我总是把信息提取到一个记

录中，而不是提取到一个个变量中。即使某些时候，这么做需要一些额外的工作，我对这种方法的优雅感到震惊，并对自己的坚持原则感到赞赏。从 Oracle9i 数据库 R2 版本开始，我甚至可以在 DML 语句中使用记录！

创建基于表的记录

每当我要在程序中保存来自表的数据时，我都会创建一个全新的（或者使用一个预定义的）基于表的记录来保存数据。这样，我只需要声明一个变量就够了。更值得称道的是，每次编译时记录的结构都会自动根据表结构的变动而调整。

把记录作为参数传递

只要可以，我就会把记录作为接口的参数，而不是用若干个变量做参数。这样一来，我对过程的调用也不会随时间发生变化，我的代码更加稳定。

第 15 章会更详细地讨论游标。不在，在下面的许多例子中游标是和记录都是同时出现的。

11.1.2 声明记录

你可以用下面 3 种方法之一来声明记录：

基于表的记录类型

用表名加%ROWTYPE 属性的方法可以声明一个记类型录，该记录类型的每个字段都和表中的一列相互对应——同时具有相同的名字。在下面的例子中，我声明了一个叫做 one_book 的记录，该记录类型和 books 表有相同的结构：

```
DECLARE
one_book books%ROWTYPE;
```

基于游标的记录类型

可以对显示声明的游标或者游标变量加上%ROWTYPE 的方法声明一个基于游标的记录类型，这个记录类型中的每一个字段都对应着游标的 SELECT 语句中的一列或者别名表达式。在下面这个例子中，我先是声明了一个显式游标，然后又声明了一个和它具有相同结构的记录类型：

```
DECLARE
CURSOR my_books_cur IS
   SELECT * FROM books
    WHERE author LIKE '%FEUERSTEIN%';

one_SF_book my_books_cur%ROWTYPE;
```

程序员自定义的记录类型

用 TYPE…RECORD 语句也可以定义记录类型，用这种方式时记录的每个字段都要在 TYPE 语句中明确的定义（包括字段名和数据类型）；自定义记录类型中的字段甚至可以是另一个记录类型。在下面的例子中，我定义了一个记录类型，TYPE

语句中包含了每本书写作生涯的信息以及一个记录类型的 “实例”:

```
DECLARE
TYPE book_info_rt IS RECORD (
   author books.author%TYPE,
   category VARCHAR2(100),
   total_page_count POSITIVE);

steven_as_author book_info_rt;
```

注意当我是根据一个记录类型声明记录时，我并没有使用%ROWTYPE 属性。Book_info_rt 元素已经是一个 TYPE。

%ROWTYPE 声明的通常格式是：

```
record_name [schema_name.]object_name%ROWTYPE
[ DEFAULT|:= compatible_record ];
```

这其中的 *schema_name* 是可选的（如果没有指定，就会使用代码编译时所在的模式进行解析）。其中 *object_name* 可以是一个显式游标、游标变量、表、视图或者同义词。作为可选的我们也可以提供一个缺省值，这个值也要是相同或者兼容的记录类型。

下面就是基于一个游标变量创建记录类型的例子：

```
DECLARE
TYPE book_rc IS REF CURSOR RETURN books%ROWTYPE;
book_cv book_rc;

one_book book_cv%ROWTYPE;
BEGIN
...
```

另一种声明和使用记录的方式是隐式的，主要见于游标型 FOR 循环中。在下面这个代码块中，声明单元并没有定义 book_rec 记录类型；PL/SQL 会自动用循环中查询语句的%ROWTYPE 属性来为我们声明这种记录类型：

```
BEGIN
FOR book_rec IN (SELECT * FROM books)
LOOP
   calculate_total_sales (book_rec);
END LOOP;
END;
```

用 TYPE 语句声明一个记录类型是最复杂也是最有趣的方法，因此我们更详细的探索一下这种方法。

11.1.3 程序员自定义的记录类型

如果我们程序中要创建的数据结构需要和表或者游标的结构相匹配，用基于表或者基于游标的记录类型是最合适的。不过这些记录类型是否覆盖了我们对于复合数据结构的全部需求呢？如果我想要创建的记录类型和表或者游标没有任何关系呢？如果我想创建的记录类型是从几个不同的表或者视图推导出来的呢？我是否有必要为需要的记录结构创建一个“哑元”游标呢？对于这些场合，PL/SQL 提供了程序员自定义的记录类型，这种类型是通过 TYPE…RECORD 语句声明的。

通过使用程序员自定义的记录类型，我们可以整体控制记录中字段的数量、名字和数据类型。要声明一个程序员自定义的记录类型，我们必须执行两个步骤：

1． 声明或者定义一个记录类型，在 TYPE 语句中包含我们想要记录的结构；

2． 基于这个记录类型声明我们自己的相同结构的实际记录。

声明自定义的记录类型

我们用 TYPE 语句声明一个记录类型。TYPE 语句指明了这个新记录结构的名字、组成

记录的成员或字段。定义记录类型的通用语法是：

```
TYPE type_name IS RECORD
   (field_name1 datatype1 [[NOT NULL]:=|DEFAULT default_value],
    field_name2 datatype2 [[NOT NULL]:=|DEFAULT default_value],
    ...
    field_nameN datatypeN [[NOT NULL]:=|DEFAULT default_value]
   );
```

这其中的 *field_nameN* 是记录类型中第 N 个字段的名称，*datatypeN* 是第 *N* 个字段的数据类型。记录类型的字段的数据类型可以是：

- 硬编码的，标量数据类型（VARCHAR2、NUMBER 等）；
- 程序员自定义的 SUBTYPE 子类型；
- 用%TYPE 或者%ROWTYPE 属性声明的锚类型。在后面的例子中，我创建了一个嵌套记录——在另一个记录中的记录类型；
- PL/SQL 的集合类型；记录中的字段可以是一个列表甚至是一个集合；
- REF CURSOR，该字段包含了一个游标变量。

下面是一个记录类型 TYPE 语句的例子：

```
TYPE company_rectype IS RECORD (
    comp# company.company_id%TYPE
  , list_of_names DBMS_SQL.VARCHAR2S
  , dataset SYS_REFCURSOR
  );
```

我们可以在本地的声明单元或者包的规范中声明一个记录类型。用后面这种方法声明的类型可以用于拥有这个包的模式下的或者对这个包有 EXECUTE 权限的模式的任何

PL/SQL 块中。

声明记录

一旦我们创建了属于我们自己的自定义记录类型，我们就可以使用这些类型来声明记录了。实际的记录声明格式如下：

```
record_name record_type;
```

这其中的 record_name 是记录的名称，record_type 是记录类型的名称，这个类型我们已经通过 TYPE…RECORD 语句定义了。

比如，要创建一个客户销售记录，我首先定义一个叫做 customer_sales_rectype 的记录类型，像下面这样：

```
PACKAGE customer_sales_pkg
IS
    TYPE customer_sales_rectype IS RECORD
       (customer_id   customer.customer_id%TYPE,
        customer_name customer.name%TYPE,
        total_sales   NUMBER (15,2)
       );
```

这个记录的结构有 3 个字段组成，包括主键、用户名以及通过计算得到的该用户的销售额总计。利用这个新的记录类型我可以声明一个相同结构的记录：

```
DECLARE
   prev_customer_sales_rec customer_sales_pkg.customer_sales_rectype;
   top_customer_rec customer_sales_pkg.customer_sales_rectype
```

注意在声明一个记录时，我不需要使用%ROWTYPE 或其他关键字来强调这是一个记录声明。只有对于表记录或者游标记录才需要使用%ROWTYPE 属性。

我们也可以把这些类型的记录作为参数传递给过程，只需要在声明过程时把记录类型作为形参的数据类型就可以，像这样：

```
PROCEDURE analyze_cust_sales (
  sales_rec_in IN customer_sales_pkg.customer_sales_rectype)
```

在声明记录的每个字段时除了要指定数据类型之外，我们还可以用 DEFAULT 或者 := 语法来为字段提供缺省值。最后，在一个记录中的每个字段名称必须是唯一的。

程序员自定义记录类型声明的例子

假设我已经像下面这样声明了的子类型、游标、关联数组[1]几种数据结构：

```
SUBTYPE long_line_type IS VARCHAR2(2000);

CURSOR company_sales_cur IS
  SELECT name, SUM (order_amount) total_sales
    FROM company c, orders o
WHERE c.company_id = o.company_id;

TYPE employee_ids_tabletype IS
  TABLE OF employees.employee_id%TYPE
  INDEX BY BINARY_INTEGER;
```

接着我就可以在同一个声明单元中声明下面这些程序员自定义的记录了：

- 一个由 company 表的子集以及代表员工的 PL/SQL 表组成的程序员自定义记录类型。我用%TYPE 属性把记录中的字段和表的字段直接联系在一起，然后又增加了第三个字段，后者实际是一个员工 ID 编号的关联数组。

```
TYPE company_rectype IS RECORD
  (company_id    company.company_id%TYPE,
   company_name  company.name%TYPE,
   new_hires_tab employee_ids_tabletype);
```

[1] 关联数组在过去也叫做“PL/SQL 表”或者“索引表”，在第 12 章中会有详细解释。

- 下面是一个混合型的记录类型，演示了一个记录类型中可以出现的各种字段声明，包括 NOT NULL 约束，子类型的使用、%TYPE 属性的使用，指定缺省值、关联数组、嵌套的记录：

```
TYPE mishmash_rectype IS RECORD
   (emp_number NUMBER(10) NOT NULL := 0,
    paragraph_text long_line_type,
    company_nm company.name%TYPE,
    total_sales company_sales.total_sales%TYPE := 0,
    new_hires_tab employee_ids_tabletype,
    prefers_nonsmoking_fl BOOLEAN := FALSE,
    new_company_rec company_rectype
   );
```

正像你所看到的一样，PL/SQL 在设计我们自己的记录结构时相当有灵活性。我们的记录类型可以代表表、视图或者 PL/SQL 中的 SELECT 语句。字段类型也可以是记录或者关联数组的任意复杂结构。

11.1.4 使用记录类型

不管我们的记录类型是如何定义的（基于表、游标或者明确的用 TYPE 语句创建的），使用记录的方法都是一样的。我们可以在“记录级别”使用记录中的数据，也可以使用记录中的单独的字段。

记录级别的操作

如果我们是在记录级别进行操作，我们可以避免对记录中单独字段的引用。目前 PL/SQL 可以支持的记录级别的操作包括：

- 我们可在把一个记录的内容拷贝到另一个记录，只要这些记录是基于相同的记录

类型或者兼容的%ROWTYPE 记录类型即可（这些记录类型具有相同数量的字段，字段的数据类型相同或者可以隐式的互相转换）；

- 我们可以用赋值操作给记录赋一个 NULL 值；
- 我们可以在参数列表中定义记录类型，并传递记录参数；
- 我们的函数接口可以返回记录类型。

目前还不支持的记录级别操作包括：

- 我们不能用 IS NULL 语法来判断一个记录的所有字段都是 NULL。相反，我们必须对每个字段使用 IS NULL 操作符。
- 我们不能对两个记录作比较——比如，我们不能比较记录（它们的字段值）是相等还是不等，或者一个记录是比另一个记录大或是小。不幸的是，要回答这种问题，我们必须比较每一个字段。我会在“记录的比较”讨论这个主题，并提供一个可以生成比较代码的工具。
- 在 Oracle 9i 数据库 R2 版本之前，我们不能向数据库表插入一个记录类型。相反，我们必须把记录的每一个字段传递给每一列。有关基于记录的 DML 操作的更多内容，见第 14 章。

只要记录的结构兼容，我们就可以使用记录级别的操作。换句话说，记录必须具有相同数量的字段，以及字段的数据类型或者相同或者可以互相转换，并不要求字段必须

是相同的数据类型。假设我创建了下面这个表：

```
CREATE TABLE cust_sales_roundup (
   customer_id NUMBER (5),
   customer_name VARCHAR2 (100),
   total_sales NUMBER (15,2)
   )
```

下面定义的 3 种记录类型的结构就都是相兼容的，我就可以像下面这样混合匹配这些记录中的字段：

```
DECLARE
   cust_sales_roundup_rec cust_sales_roundup%ROWTYP E
   CURSOR cust_sales_cur IS SELECT * FROM cust_sal es_roundup;
   cust_sales_rec cust_sales_cur%ROWTYPE;
   TYPE customer_sales_rectype IS RECORD
      (customer_id NUMBER(5),
       customer_name customer.name%TYPE,
       total_sales NUMBER(15,2)
      );
   preferred_cust_rec customer_sales_rectype;
BEGIN
   -- 把一个记录赋值给另外一个记录
   cust_sales_roundup_rec := cust_sales_rec;
   preferred_cust_rec := cust_sales_rec;
END;
```

让我们再看其他一些关于记录级别操作的例子。

- 在这个例子中，我打算给记录一个缺省值。我们可以在声明记录时用另一个类型兼容的记录赋值来初始化这个记录。接下来的程序中，我把作为传入参数的记录赋值给一个局部变量。这样一来我就可以修改记录的字段值了：

```
PROCEDURE compare_companies
   (prev_company_rec IN company%ROWTYPE)
IS
   curr_company_rec company%ROWTYPE := prev_company_rec;
BEGIN
   ...
```

```
END;
```

- 在下面这个初始化例子中，我创建了一个新的记录类型和记录变量，接着我又创建了第二个记录类型，并把第一个记录类型作为它唯一的一列。最后，我有用之前定义的记录变量对这个新的记录进行了初始化：

```
DECLARE
  TYPE first_rectype IS RECORD (var1 VARCHAR2(100) := 'WHY NOT');
  first_rec first_rectype;
  TYPE second_rectype IS RECORD (nested_rec first_rectype := first_rec);
BEGIN
  ...
END;
```

- 正如你所想的那样，我也可以在执行单元进行赋值操作，在下面的例子中，我声明了两个不同的 rain_forest_history 记录变量，并用之前的历史记录设置当前历史信息：

```
DECLARE
  prev_rain_forest_rec rain_forest_history%ROWTYPE;
  curr_rain_forest_rec rain_forest_history%ROWTYPE;
BEGIN
  ... initialize previous year rain forest data ...
  -- Transfer data from previous to current records.
  curr_rain_forest_rec := prev_rain_forest_rec;
```

- 这种整体赋值的结果就是 current 记录的每个字段值都被设置成 previous 记录的对应字段的值。我也可以用 previous 和 current 记录的每个字段间的直接赋值来完成这个工作，不过这就需要若干个独立的赋值语句，以及要打更多的字；只要可能，就应该尽量使用记录级别的操作。既可以节省时间，也可以让代码更稳定。

- 我也可以把表中的一行直接提取到代码中的一个记录变量。下面是两个例子：

```
DECLARE
  /*
```

```
   || 声明一个游标，然后用这个游标的%ROWTYPE 属性定义一个记录类型。
   */
   CURSOR cust_sales_cur IS
      SELECT customer_id, customer_name, SUM (total_sales) tot_sales
        FROM cust_sales_roundup
       WHERE sold_on < ADD_MONTHS (SYSDATE, -3)
      GROUP BY customer_id, customer_name;
   cust_sales_rec cust_sales_cur%ROWTYPE;
BEGIN
   /*通过对游标的提取操作直接给记录变量赋值*/
   OPEN cust_sales_cur;
   FETCH cust_sales_cur INTO cust_sales_rec;
   CLOSE cust_sales_cur;
```

下面这个代码块中，我用 TYPE 声明了一个程序员自定义的记录类型，用来匹配隐式游标的数据。接着我的 SELECT 直接把数据取到基于此类型的记录中：

```
DECLARE
   TYPE customer_sales_rectype IS RECORD
      (customer_id   customer.customer_id%TYPE,
       customer_name customer.name%TYPE,
       total_sales   NUMBER (15,2)
       );
   top_customer_rec  customer_sales_rectype;
BEGIN
   /* 直接把数据提取到记录变量中: */
   SELECT customer_id, customer_name, SUM (total_sales)
     INTO top_customer_rec
     FROM cust_sales_roundup
    WHERE sold_on < ADD_MONTHS (SYSDATE, -3)
     GROUP BY customer_id, customer_name;
```

- 我可以通过直接赋值操作，把记录的所有字段设置成 NULL 值：

```
/* File on web: record_assign_null.sql */
FUNCTION dept_for_name (
   department_name_in IN departments.department_name%TYPE
)
   RETURN departments%ROWTYPE
IS
   l_return   departments%ROWTYPE;

   FUNCTION is_secret_department (
      department_name_in IN departments.department_name%TYPE
   )
      RETURN BOOLEAN
   IS
```

```
   BEGIN
      RETURN CASE department_name_in
               WHEN 'VICE PRESIDENT' THEN TRUE
               ELSE FALSE
             END;
   END is_secret_department;
BEGIN
   SELECT *
     INTO l_return
     FROM departments
     WHERE department_name = department_name_in;

   IF is_secret_department (department_name_in)
   THEN
      l_return := NULL;
   END IF;
   RETURN l_return;
END dept_for_name;
```

任何时候只要有可能，就应该整体级别的方式来使用记录；把记录作为一个整体对待，而不是一个个独立的字段。作为回报，代码会更容易编写和维护。当然，也会有许多场合我们需要单独操作记录中的字段。让我们看看如何做到这一点。

字段级别的操作

如果我们需要访问记录中的一个字段（或者是读取或者是改变字段值），我们必须使用圆点表示法，就像我们要表示指定的数据库表中的一列那样。这种语法是：

```
[[schema_name.]package_name.]record_name.field_name
```

如果记录不是在我们当前包，而是在另一个不同的包规范中定义的，我们就需要提供包名。如果包不属于代码编译所在的模式，还需要提供模式名。

一旦我们使用圆点表示法来标识某个特定字段，我们该如何引用或者改变这个字段的值遵循 PL/SQL 的通常规则。让我们一起看一些例子。

赋值操作符（:=）可以改变某个特定字段的值，在第一个赋值中，total_sales 被清零。在第二个赋值中，调用了一个函数，这个函数的返回值用来设置 output_generated 的布尔标志（被设置成 TRUE、FALSE 或者 NULL）。

```
   BEGIN
      top_customer_rec.total_sales := 0;
      report_rec.output_generated := check_report_status
(report_rec.report_id);
   END;
```

接下来的这个例子中，我基于表 rain_forest_history 创建了一个记录，然后填入值，再向同一个表中插入记录：

```
   DECLARE
      rain_forest_rec rain_forest_history%ROWTYPE;
   BEGIN
      /* 设置记录的值*/
      rain_forest_rec.country_code  := 1005;
      rain_forest_rec.analysis_date := ADD_MONTHS (TRUNC (SYSDATE), -3);
      rain_forest_rec.size_in_acres := 32;
      rain_forest_rec.species_lost  := 425;

      /*向表中插入记录的值 */
      INSERT INTO rain_forest_history
            (country_code, analysis_date, size_in_acres, species_lost)
      VALUES
         (rain_forest_rec.country_code,
          rain_forest_rec.analysis_date,
          rain_forest_rec.size_in_acres,
          rain_forest_rec.species_lost);
       ...
   END;
```

注意，因为 analysis_date 这个字段是日期类型的，我可以用任何有效的日期表达式给这个字段赋值。对于其他字段也是一样，就是更复杂的结构也是同样。

从 Oracle 9i 数据库 R2 版本开始，我们也可以执行一个记录级别的插入操作，上面那个 INSERT 语句就可以简化成这样：

```
INSERT INTO rain_forest_history
   (country_code, analysis_date, size_in_acres, species_lost)
VALUES rain_forest_rec;
```

记录级别的 DML 操作（包括插入和更新）在第 14 章介绍。

对嵌套记录的字段级别的操作

假设我创建了一个嵌套的记录结构，也就是说“外层”记录的一个字段实际是另外一个记录类型。在下面的例子中，我首先声明了一个包含了电话号码全部元素的记录类型（phone_rectype），接着又声明了另外一个记录类型，这个记录类型（contact_set_rectype）把一个人的全部电话号码统统收集到一个结构中。

```
DECLARE
   TYPE phone_rectype IS RECORD
      (intl_prefix   VARCHAR2(2),
       area_code     VARCHAR2(3),
       exchange      VARCHAR2(3),
       phn_number    VARCHAR2(4),
       extension     VARCHAR2(4)
      );
   -- 每个字段都是一个嵌套的记录...
  TYPE contact_set_rectype IS RECORD
      (day_phone#    phone_rectype,
       eve_phone#    phone_rectype,
       fax_phone#    phone_rectype,
       home_phone#   phone_rectype,
       cell_phone#   phone_rectype
      );
   auth_rep_info_rec contact_set_rectype;
BEGIN
```

由于我仍然是用句点表示法引用嵌套记录中的字段，现在我必须引用嵌套在结构的深层的字段。要达到这个目的，我必须为每一个嵌套的记录结构都使用一个圆点，就像下面这个赋值中显示的，这个例子是用家里电话号码的区号设置传真的区号。

```
auth_rep_info_rec.fax_phone#.area_code :=
   auth_rep_info_rec.home_phone#.area_code;
```

对基于包的记录的字段级别操作

最后，再用一个例子演示如何引用包中的记录（或者基于包的记录类型）。假设我在准备暑期的读书计划（那些懒洋洋地呆在加勒比沙滩上的日子）。我创建了下面这个包规范：

```
CREATE OR REPLACE PACKAGE summer
IS
   TYPE reading_list_rt IS RECORD (
      favorite_author  VARCHAR2 (100),
      title            VARCHAR2 (100),
      finish_by        DATE);

   must_read reading_list_rt;
   wifes_favorite reading_list_rt;
END summer;

CREATE OR REPLACE PACKAGE BODY summer
IS
BEGIN  --包的初始化部分
   must_read.favorite_author := 'Tepper, Sheri S.';
   must_read.title := 'Gate to Women''s Country';
END summer;
```

等这个包在数据库编译通过后，我就可以像下面这样来构造我的阅读列表了：

```
DECLARE
   first_book summer.reading_list_rt;
   second_book summer.reading_list_rt;
BEGIN
   summer.must_read.finish_by := TO_DATE ('01-AUG-2009', 'DD-MON-YYYY');
   first_book := summer.must_read;

   second_book.favorite_author := 'Hobb, Robin';
   second_book.title := 'Assassin''s Apprentice';
   second_book.finish_by := TO_DATE ('01-SEP-2009', 'DD-MON-YYYY');
END;
```

我声明了两个局部图书记录，接着对包中的必读书籍（must_read）设置了“截止”日期（注

意 *package.record.field* 语法），然后把这个包记录赋值给代表暑期第一本书的变量，再对暑假第二本书的每个字段赋值。

注意，当我们在 PL/SQL 中使用 UTL_FILE 内置包进行文件的 I/O 操作时，我们要遵守相同的规则。UTL_FILE.FILE_TYPE 数据类型实际就是一个记录类型的 TYPE 定义。因此，当我们声明一个文件句柄时，我们真正声明的是一个基于包的记录类型：

```
DECLARE
  my_file_id UTL_FILE.FILE_TYPE;
```

11.1.5 记录的比较

我们该如何检查两个记录是否相等（即每个对应字段的值都相同）？如果 PL/SQL 能让我们直接进行比较就好了，就像这样：

```
DECLARE
  first_book summer.reading_list_rt := summer.must_read;
  second_book summer.reading_list_rt := summer.wifes_favorite;
BEGIN
  IF first_book = second_book /*还不支持这种方式！ */
  THEN
    lots_to_talk_about;
  END IF;
END;
```

不幸的是，我们不能这么操作。相反，要想检测两个记录是否相等，我们必须编写代码比较每一个字段。如果一个记录的字段不算多，这种方法还不算麻烦。对与刚才的读书列表记录，我们需要类似这样的代码：

```
DECLARE
  first_book summer.reading_list_rt := summer.must_read;
  second_book summer.reading_list_rt := summer.wifes_favorite;
BEGIN
  IF  first_book.favorite_author = second_book.favorite_author
```

```
      AND first_book.title = second_book.title
      AND first_book.finish_by = second_book.finish_by
    THEN
      lots_to_talk_about;
    END IF;
END;
```

还要注意另外一个复杂性，如果我们的需求要求两个 NULL 的记录也按照相等处理（相等的 NULL），我们必须把之前的比较修改成类似于下面这种样子：

```
(first_book.favorite_author = second_book.favorite_author
  OR( first_book.favorite_author IS NULL AND
      second_book.favorite_author IS NULL))
```

不论从哪个角度来看，这样的代码也是相当冗长乏味。如果我们能够生成这种代码岂不是更好？事实上，也没有那么难——至少如果要比较的记录是用表或者视图的%ROWTYPE 属性定义的。对这种要求，我们可以从 ALL_TAB_COLUMNS 这个字典视图中得到所有字段的名字，然后把格式化后的代码输出到屏幕或者文件中。

不过还好，这些工作我们无须亲力亲为。我们可以下载并运行由 Dan Spencer 提供的“记录相等”生成器。在本书站点的 *gen_record_comparison.pkg* 文件中有这个包。

11.1.6 触发器伪记录

如果是在数据库触发器中操作某个表，数据库为我们提供了两种结构，OLD 和 NEW，这两个是伪记录。这两个结构和用%ROWTYPE 声明的基于表的记录类型有相同的格式：表中的每一列都有一个对应的字段：

OLD

这条伪记录代表的是当前事务开始之前表中记录的值。

NEW

这条伪记录代表的是当前事务结束之后，表中记录的新值。

当我们在触发器中引用 OLD 和 NEW 时，我们必须在这些标识符前面加上一个冒号；不过要是在 WHEN 语句中就不需要冒号。下面是一个例子：

```
TRIGGER check_raise
   AFTER UPDATE OF salary
   ON employee
   FOR EACH ROW
WHEN  (OLD.salary != NEW.salary) OR
      (OLD.salary IS NULL AND NEW.salary IS NOT NULL) OR
      (OLD.salary IS NOT NULL AND NEW.salary IS NULL)
BEGIN
   IF :NEW.salary > 100000 THEN ...
```

第 19 章针对在数据库触发器中该如何使用 OLD、NEW 伪记录提供了更完整的解释，特别还提到了使用 OLD、NEW 时的若干限制。