

Th.S NGUYỄN QUẢNG NINH - NGUYỄN NAM THUẬN
và nhóm tin học thực dụng



GIÁO TRÌNH

**HƯỚNG DẪN LÝ THUYẾT
KÈM THEO BÀI TẬP THỰC HÀNH**

Oracle 11g

DÀNH CHO HỌC SINH, SINH VIÊN

ẤN BẢN 2009

TẬP 1

AN VAN CORP.



NHÀ XUẤT BẢN HỒNG ĐỨC

Giáo trình

**Hướng dẫn Lý thuyết và
kèm theo bài tập thực hành**

ORACLE 11g - *tập 1*

Th.S: NGUYỄN QUẢNG NINH - NGUYỄN NAM THUẬN
và nhóm Tin học thực dụng

GIÁO TRÌNH

Tập 1

Hướng dẫn Lý thuyết
và kèm theo bài tập thực hành
ORACLE 11g

Dành cho học sinh -
sinh viên

Ấn bản 2009

NHÀ XUẤT BẢN HỒNG ĐỨC

LỜI NÓI ĐẦU

"Giáo trình Hướng dẫn Lý thiết và kèm theo bài tập thực hành ORACLE 11g" được biên soạn toàn diện từ đầu đến cuối nhằm mục đích giúp bạn đọc làm quen với ngôn ngữ lập trình PL/SQL, nắm vững các điểm cơ bản của ngôn ngữ và các kỹ thuật lập trình nâng cao để có thể phát triển các ứng dụng PL/SQL mạnh được điều khiển bởi cơ sở dữ liệu, giúp nhanh chóng hoàn thành các dự án phát triển của các cá nhân, công ty và doanh nghiệp.

Sách gồm 2 tập, được thiết kế thành các phần chính sau đây:

Phần I: Các điểm cơ bản về PL/SQL

Phần này giới thiệu các tính năng Oracle Database 10g phiên bản 2 gần đây và các tính năng Oracle Database 11g mới, trình bày các điểm cơ bản của PL/SQL, và giải thích các khái niệm về ngữ nghĩa, các kiểu dữ liệu, các cấu trúc điều khiển và sự quản lý lỗi.

Phần II: Lập trình PL/SQL

Phần này trình bày cách sử dụng các công cụ mạnh để phát triển các ứng dụng cơ sở dữ liệu phù hợp nhu cầu thực tế và mang lại hiệu quả ứng dụng cao, bao gồm các hàm và thủ tục, các tập hợp, các gói, các bộ kích khởi (trigger) và các đối tượng lớn Oracle.

Phần III: Lập trình nâng cao PL/SQL

Với phần này, bạn đọc sẽ được hướng dẫn về SQL động, sự liên kết giữa các phiên làm việc, các thủ tục ngoài, các loại đối tượng, các thư viện Java và sự phát triển ứng dụng Web nhằm đạt được các kỹ thuật lập trình nâng cao với PL/SQL.

Sách được bố cục rõ ràng theo từng chủ điểm cụ thể, nội dung trình bày ngắn gọn, hy vọng sẽ là một tài liệu tham khảo thật sự hữu ích cho bạn đọc, đặc biệt là ở lĩnh vực lập trình cơ sở dữ liệu.

Rất mong đón nhận sự đóng góp ý kiến chân thành từ bạn đọc về một số sai sót khó tránh khỏi trong quá trình biên soạn nhằm làm cho sách hoàn thiện hơn. Xin chân thành cảm ơn.

Tác giả

PHẦN I

Các điểm cơ bản về Oracle PL/SQL

Chương 1: Tổng quan về Oracle PL/SQL

Chương 2: Các điểm cơ bản về PL/SQL

Chương 3: Các điểm cơ bản về ngôn ngữ

Chương 4: Các cấu trúc điều khiển

Chương 5: Quản lý lỗi

CHƯƠNG 1

TỔNG QUAN VỀ ORACLE PL/SQL

Chương này giới thiệu đến bạn Procedure Language/Structured Query Language (PL/SQL). Chương giải thích lịch sử, kiến trúc và cấu trúc khối của PL/SQL, xem lại những tính năng Oracle 10g và thảo luận những tính năng mới của Oracle 11g. Chương được chia thành các phần sau đây:

- Lịch sử và thông tin cơ bản
- Kiến trúc
- Những cấu trúc khối cơ bản
- Những tính năng mới của Oracle 10g
- Những tính năng mới của Oracle 11g

Lịch sử và thông tin cơ bản

PL/SQL đã được Oracle phát triển vào những năm 1980. Ban đầu PL/SQL có những khả năng giới hạn, nhưng điều đó đã thay đổi vào đầu những năm 1990. PL/SQL cung cấp cho chương trình Oracle một môi trường lập trình cài sẵn có khả năng thông dịch và độc lập với hệ điều hành. Các câu lệnh SQL được tích hợp riêng trong ngôn ngữ PL/SQL.

Oracle 8 Database đã đưa các kiểu đối tượng vào cơ sở dữ liệu. Nó đã biến cơ sở dữ liệu Oracle từ một mô hình quan hệ thuần túy trở thành mô hình quan hệ đối tượng (hoặc quan hệ mở rộng). Những kiểu này có giá trị giới hạn dưới dạng các tập hợp biến vô hướng cho đến khi chúng trở nên có khả năng thể hiện cụ thể trong Oracle 9i, Release 2. Khả

năng thể hiện cụ thể (instantiate) các kiểu đối tượng SQL đã làm cho các đối tượng Oracle bên trong tương thích với các kiểu đối tượng C++, Java hoặc C#. Các kiểu đối tượng PL/SQL được thực thi trong SQL và được đề cập trong chương 15.

PL/SQL đã phát triển với sự ra đời của đầy đủ những tính năng lập trình hướng đối tượng trong Oracle 9i, Release 2. PL/SQL không còn là một ngôn ngữ hoàn toàn thủ tục nữa. Bây giờ nó vừa là ngôn ngữ lập trình thủ tục vừa là ngôn ngữ lập trình hướng đối tượng.

Oracle 11g Database cũng phát triển PL/SQL từ ngôn ngữ thông dịch thành một ngôn ngữ biên dịch riêng. Bạn có thể hỏi "Điều đó có loại bỏ lợi ích của một ngôn ngữ độc lập với hệ điều hành hay không?" Lời giải đáp cho câu hỏi đó là không. Bây giờ bạn có thể viết PL/SQL một lần ở dạng độc lập với hệ điều hành, sau đó triển khai nó và để Oracle quản lý việc biên dịch riêng. Oracle 11g tự động hoá tiến trình trên các nền được hỗ trợ.

Các phiên bản PL/SQL

Các phiên bản PL/SQL ban đầu không được xếp trình tự với phiên bản của cơ sở dữ liệu. Ví dụ, PL/SQL 1.0 đính kèm với Oracle 6 Database. PL/SQL 2x đính kèm với Oracle 7.x Databases. Bắt đầu với Oracle 8, các phiên bản PL/SQL tương ứng với các số phiên bản cơ sở dữ liệu như PL/SQL 11.1 trong Oracle 11g Release 1 Database.

Cũng như khả năng gọi từ những chương trình bên ngoài, PL/SQL cũng là cửa ngõ chính đi đến các thư viện ngoài. Nhân thư viện ngoài gây nhầm lẫn vì các thư viện Java cũng được lưu trữ bên trong cơ sở dữ liệu. Oracle gọi các thư viện (library) ngoài là những thủ tục ngoài thông qua PL/SQL bất kể chúng được lưu trữ ở đâu. Các chương trình PL/SQL phục vụ như các wrapper cho những thư viện ngoài. Các wrapper là những giao diện (interface) che giấu sự chuyển đổi kiểu giữa cơ sở dữ liệu và các chương trình bên ngoài.

Bạn có thể mở rộng chức năng của Oracle 11g Database khi sử dụng các hàm và thủ tục lưu trữ trong PL/SQL, C, C++, hoặc Java. Các chương trình Java có thể được lưu trữ trực tiếp trong Oracle 11g Database trong tất cả phiên bản ngoại trừ Oracle Express Edition. Chương 12 trình bày cách xây dựng và chạy các thủ tục (procedure) ngoài. Chương 14 đề cập cách xây dựng và triển khai các thư viện Java bên trong cơ sở dữ liệu.

PL/SQL tiếp tục phát triển và ngày càng trở nên mạnh hơn. Lập trình PL/SQL mang đến những thách thức cho những người mới làm quen với ngôn ngữ bởi vì nó phục vụ quá nhiều vấn đề trong cơ sở dữ liệu Oracle. Khi đã có những kỹ năng về ngôn ngữ, bạn sẽ học cách sử dụng PL/SQL để giải quyết những vấn đề luôn phức tạp hơn.

Kiến trúc

Ngôn ngữ PL/SQL là một công cụ mạnh với nhiều tùy chọn. PL/SQL cho phép viết mã một lần và triển khai nó trong cơ sở dữ liệu gắn dữ liệu nhất. PL/SQL có thể đơn giản hoá việc phát triển ứng dụng, tối ưu hoá việc thực thi và cải thiện việc tận dụng nguồn tài nguyên trong cơ sở dữ liệu.

Ngôn ngữ là một ngôn ngữ lập trình không nhạy kiểu chữ như SQL. Điều này đã dẫn đến vô số hướng dẫn tốt nhất của việc định dạng. Thay vì lặp lại những đối số đó cho kiểu này hay kiểu khác, dường như tốt nhất nên đề nghị bạn tìm một kiểu nhất quán với các chuẩn của tổ chức và áp dụng nó một cách nhất quán. Mã PL/SQL trong sách này sử dụng chữ hoa cho các từ lệnh và chữ thường cho các biến, tên cột và lệnh gọi chương trình lưu trữ.

PL/SQL đã được phát triển bằng cách mô phỏng những khái niệm về lập trình cấu trúc, định kiểu dữ liệu tĩnh, tính module, quản lý ngoại lệ (exception) và xử lý song song (đồng thời) được tìm thấy trong ngôn ngữ lập trình Ada. Ngôn ngữ lập trình Ada, phát triển cho bộ quốc phòng Mỹ, được thiết kế để hỗ trợ các hệ thống nhúng thời gian thực và quan trọng đối với sự an toàn của quân sự chẳng hạn như các hệ thống trong các máy bay và tên lửa. Ngôn ngữ lập trình Ada vay mượn cú pháp quan trọng từ ngôn ngữ lập trình Pascal, bao gồm các toán tử gán và so sánh và các dấu phân cách dấu trích dẫn đơn.

Những lựa chọn này cũng cho phép đưa trực tiếp các câu lệnh SQL vào các khối mã PL/SQL. Chúng quan trọng bởi vì SQL chấp nhận các toán tử Pascal, dấu tách chuỗi và kiểu dữ liệu vô hướng khai báo y như thế. Cả Pascal và Ada đều có các kiểu khai báo vô hướng. Các kiểu dữ liệu khai báo không thay đổi vào thời gian chạy và được gọi là các kiểu dữ liệu mạnh. Các kiểu dữ liệu mạnh thì quan trọng cho việc tích hợp chặt chẽ các ngôn ngữ Oracle SQL và PL/SQL. PL/SQL hỗ trợ các kiểu dữ liệu động bằng cách ánh xạ chúng vào thời gian chạy với các kiểu được định nghĩa trong catalog Oracle 11g Database. Ánh xạ các toán tử và dấu tách chuỗi nghĩa là việc phân tích cú pháp được đơn giản hoá bởi vì những câu lệnh SQL được nhúng riêng trong các đơn vị lập trình PL/SQL.

Ghi chú

Các đối tượng nguyên thủy trong ngôn ngữ lập trình Java mô tả các biến vô hướng (scalar variables) chứa chỉ mỗi lần một thứ.

Đội ngũ phát triển PL/SQL ban đầu đã đưa ra những lựa chọn này một cách cẩn thận. Cơ sở dữ liệu Oracle đã được đền đáp qua nhiều năm nhờ vào những lựa chọn này. Một lựa chọn nổi bật là cho bạn liên kết các biến PL/SQL với catalog cơ sở dữ liệu. Đây là một dạng thừa kế kiểu thời gian chạy. Bạn sử dụng các giả kiểu (pseudotype) %TYPE và %ROWTYPE để thừa kế từ các biến được định kiểu mạnh được định nghĩa trong catalog cơ sở dữ liệu (được đề cập trong các chương 3 và 9).

Neo (anchor) các biến PL/SQL vào các đối tượng catalog cơ sở dữ liệu là một dạng ghép đôi cấu trúc hiệu quả. Nó có thể giảm thiểu số thay đổi mà bạn cần thực hiện đối với những chương trình PL/SQL. Tối thiểu nó giới hạn mức độ thường xuyên bạn tái tạo mã do kết quả của những thay đổi giữa các kiểu cơ sở như thay đổi VARCHAR2 thành DATE. Nó cũng loại bỏ nhu cầu tái định nghĩa các kích cỡ biến. Ví dụ, bạn không cần chỉnh sửa mã khi một bảng (table) thay đổi kích cỡ của một chuỗi có chiều dài khả biến.

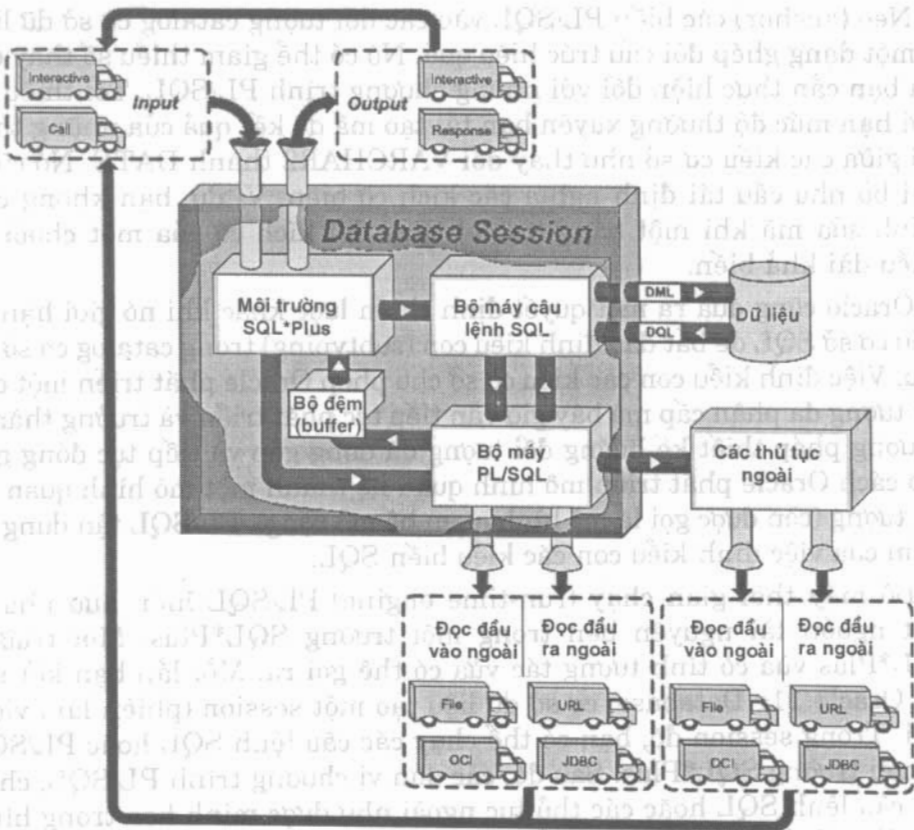
Oracle cũng đưa ra một quyết định chiến lược khác khi nó giới hạn số kiểu cơ sở SQL để bắt đầu định kiểu con (subtyping) trong catalog cơ sở dữ liệu. Việc định kiểu con các kiểu cơ sở cho phép Oracle phát triển một cây đối tượng đa phân cấp mà bây giờ vẫn tiếp tục phát triển và trưởng thành. Phương pháp thiết kế hướng đối tượng đã đóng góp và tiếp tục đóng góp cho cách Oracle phát triển mô hình quan hệ thành một mô hình quan hệ đối tượng (còn được gọi là mô hình quan hệ mở rộng). PL/SQL tận dụng ưu điểm của việc định kiểu con các kiểu biến SQL.

Bộ máy thời gian chạy (run-time engine) PL/SQL hiện hữu như là một nguồn tài nguyên bên trong môi trường SQL*Plus. Môi trường SQL*Plus vừa có tính tương tác vừa có thể gọi ra. Mỗi lần bạn kết nối với Oracle 11g Database, cơ sở dữ liệu tạo một session (phiên làm việc) mới. Trong session đó, bạn có thể chạy các câu lệnh SQL hoặc PL/SQL từ môi trường SQL*Plus. Sau đó, các đơn vị chương trình PL/SQL, chạy các câu lệnh SQL hoặc các thủ tục ngoài như được minh họa trong hình 1.1. Câu lệnh SQL cũng có thể gọi các hàm hoặc thủ tục được lưu trữ PL/SQL. Các câu lệnh SQL tương tác trực tiếp với dữ liệu thật sự.

Các lệnh gọi trực tiếp đến PL/SQL được thực hiện qua Oracle Call Interface (OCI) hoặc Java Database Connectivity (JDBC). Điều này cho phép đẩy mạnh PL/SQL lưu trữ, PL/SQL ngay trong các ứng dụng các cơ sở dữ liệu, quan trọng bởi vì nó cho phép quản lý phạm vi giao dịch (transaction scope) trong các đơn vị chương trình đơn giản đáng kể rất nhiều tác vụ thường được đặt trong lớp trừu tượng dữ liệu của các ứng dụng.

PL/SQL cũng hỗ trợ việc xây dựng các câu lệnh SQL vào thời gian chạy. Các câu lệnh SQL thời gian chạy là SQL động. Bạn có thể sử dụng hai phương pháp cho SQL hoạt động: một là Native Dynamic SQL (NDS) và phương pháp lược là DBMS_SQL package. Oracle 11g Database phân phối các tính năng NDS mới và cải thiện tốc độ thực thi. Với ấn bản này, bạn chỉ cần sử dụng DBMS_SQL package khi nào bạn không biết rõ về số cột mà lệnh gọi SQL động yêu cầu. Chương 11 sẽ đề cập đến SQL động và cả hai phương pháp trên.

Bây giờ bạn có một cái nhìn cấp cao về ngôn ngữ PL/SQL. Phần tiếp theo sẽ đưa ra một cái nhìn tổng quan ngắn gọn về những cấu trúc khối PL/SQL.



Hình 1.1 Kiến trúc xử lý cơ sở dữ liệu.

Các cấu trúc khối cơ bản

PL/SQL là một ngôn ngữ lập trình kết khối. Các đơn vị chương trình có thể là các khối được đặt tên hoặc không được đặt tên. Các khối không được đặt tên gọi là các khối nặc danh và được gọi như vậy qua suốt sách này. Kiểu viết mã PL/SQL khác với kiểu viết mã của các ngôn ngữ lập

trình C, C++ và Java. Ví dụ, các dấu ngoặc cong không phân tách các khối trong PL/SQL.

Các chương trình khối nặc danh có hiệu quả trong một số tình huống. Bạn thường sử dụng các khối nặc danh khi xây dựng các script để seed (gieo hạt) dữ liệu hoặc thực hiện các hoạt động xử lý một lần. Chúng cũng hiệu quả khi bạn muốn xếp lồng hoạt động trong phần thực thi của một khối PL/SQL khác. Cấu trúc khối nặc danh (anonymous-block) cơ bản phải chứa một phần thực thi. Bạn có thể đặt các phần khai báo ngoại lệ và tùy chọn trong các khối nặc danh. Phần sau đây minh họa một nguyên mẫu khối nặc danh:

```
[DECLARE]
declaration_statements
BEGIN
    execution_statements
[EXCEPTION]
exception_handling_statements
END;
/
```

Khối khai báo (declaration) cho phép định nghĩa các kiểu dữ liệu, cấu trúc và biến. Định nghĩa một biến (variable) nghĩa là bạn đặt cho nó một tên và một kiểu dữ liệu (datatype). Bạn cũng có thể định nghĩa một biến bằng cách đặt cho nó một tên, một kiểu dữ liệu và giá trị. Bạn vừa định nghĩa vừa gán một giá trị khi khai báo một biến.

Một số kiểu đối tượng không thể được định nghĩa là các biến có phạm vi cục bộ, nhưng phải được định nghĩa là các kiểu trong catalog cơ sở dữ liệu như thảo luận trong chương 14. Các cấu trúc (Structures) là các biến phức hợp như các tập hợp (collection), cấu trúc bản ghi (record) hoặc cursor tham chiếu hệ thống. Các cấu trúc cũng có thể là các hàm, thủ tục hoặc cursor được đặt tên cục bộ. Các cursor hoạt động hơi giống các hàm. Các cursor có các tên, chữ ký (signature) và một kiểu trả về - các cột kết quả từ một query (mẫu truy vấn) hoặc một câu lệnh SELECT. Từ dành riêng DECLARE bắt đầu khối khai báo và từ dành riêng BEGIN kết thúc nó.

Khối thực thi (execution) cho phép xử lý dữ liệu. Khối thực thi có thể chứa các phép gán biến, phép so sánh, phép toán có điều kiện và các phép lặp lại. Khối thực thi cũng là nơi bạn có thể truy cập các cursor và các đơn vị chương trình được đặt tên khác. Các hàm, thủ tục và một số kiểu đối tượng là các đơn vị chương trình được đặt tên. Bạn cũng có thể xếp lồng các chương trình khối nặc danh bên trong khối thực thi. Từ dành riêng BEGIN bắt đầu khối ngoại lệ (execution) và từ dành riêng

EXCEPTION tùy chọn hoặc END bắt buộc kết thúc nó. Tối thiểu bạn phải có một câu lệnh bên trong một khối thực thi. Câu lệnh khối nặc danh tối thiểu sau đây chứa một câu lệnh NULL:

```
BEGIN
    NULL;
END;
/
```

Câu lệnh này không làm gì cả ngoại trừ cho cụm từ biên dịch hoàn tất mà không có lỗi nào. Sự biên dịch bằng bất kỳ ngôn ngữ bao gồm sự phân tích cú pháp. Việc thiếu một câu lệnh trong khối sẽ đưa ra một lỗi phân tích cú pháp như được đề cập trong chương 5.

Khối xử lý ngoại lệ cho phép quản lý các ngoại lệ (exception). Bạn vừa có thể đón bắt vừa quản lý chúng ở đó. Khối ngoại lệ cho phép xử lý luân phiên; trong nhiều cách nó hành động như một sự kết hợp của khối catch và finally trong ngôn ngữ lập trình Java. Từ dành riêng EXCEPTION bắt đầu phần, và từ dành riêng END kết thúc nó.

••••• Thủ thuật

Bạn có cùng một qui tắc yêu cầu tối thiểu một câu lệnh cho bất kỳ khối trong một khối câu lệnh điều kiện (như một câu lệnh IF) và các vòng lặp (loop).

Các chương trình khối được đặt tên có cấu trúc khối hơi khác bởi vì chúng được lưu trữ trong cơ sở dữ liệu. Chúng cũng có một phần khai báo được gọi là header. Tên, danh sách các tham số hình thức và bất kỳ kiểu trả về của các khối PL/SQL được đặt tên được định nghĩa bởi header. Tên và danh sách các tham số hình thức được gọi là chữ ký của một thường trình con (subroutine). Vùng giữa header và các khối thực thi có chức năng như khối khai báo cho một khối được đặt tên. Qui tắc này tương tự đối với các phần kiểu đối tượng được đề cập trong chương 14.

Ví dụ sau đây minh họa một nguyên mẫu hàm khối được đặt tên:

```
FUNCTION function_name
[( parameter1 [IN][OUT] [NOCOPY] sql_data_type | plsql_data_type
, parameter2 [IN][OUT] [NOCOPY] sql_data_type | plsql_data_type
, parameter(n+1) [IN][OUT] [NOCOPY] sql_data_type | plsql_data_type )]

RETURN [ sql_data_type | plsql_data_type ]
[ AUTHID {DEFINER | CURRENT_USER}]
[ DETERMINISTIC | PARALLEL_ENABLED ]
[ PIPELINED ]
```

```
[ RESULT_CACHE [RELIES ON table_name]] IS
declaration_statements
BEGIN
    execution_statements
[EXCEPTION]
    exception_handling_statements
END;
/
```

Chương 6 thảo luận các hàm chi phối các qui tắc. Các hàm có thể hoạt động như các thường trình con chuyển theo giá trị hoặc chuyển theo tham chiếu. Các thường trình con chuyển theo giá trị định nghĩa các tham số hình thức bằng cách sử dụng chỉ một chế độ IN. Điều này có nghĩa biến được chuyển và không thể được thay đổi trong quá trình thực thi thường trình con. Các thường trình con chuyển theo tham chiếu định nghĩa các tham số hình thức bằng cách sử dụng các chế độ IN và OUT hoặc chỉ OUT.

Oracle 11g tiếp tục chuyển các bản sao của các biến thay vì các tham chiếu đến các biến trừ phi bạn chỉ định một gọi ý NOCOPY. Oracle thực thi các hành vi chuyển theo tham chiếu bằng cách này để bảo đảm tính toàn vẹn của các biến chế độ IN OUT. Mô hình này bảo đảm các biến không thay đổi trừ phi lệnh gọi thường trình con thành công. Bạn có thể ghi đè hành vi mặc định này bằng cách sử dụng một gọi ý NOCOPY. Oracle đề nghị không sử dụng gọi ý NOCOPY bởi vì sử dụng nó có thể làm thay đổi một phần các giá trị tham số thật sự. Cuối cùng cơ sở dữ liệu chọn việc hành động theo một gọi ý và gửi một tham chiếu hay không.

Các hàm có thể truy vấn dữ liệu bằng cách sử dụng những câu lệnh SELECT nhưng không thể thực thi các câu lệnh DML chẳng hạn như INSERT, UPDATE, hoặc DELETE. Tất cả qui tắc khác áp dụng vào các hàm lưu trữ giống như những qui tắc áp dụng vào các khối nặc danh. Các hàm định nghĩa các tham số hình thức hoặc các kiểu trả về vốn sử dụng các kiểu dữ liệu PL/SQL không thể được gọi từ dòng lệnh SQL. Tuy nhiên, bạn có thể gọi các hàm sử dụng các kiểu dữ liệu SQL từ dòng lệnh SQL.

Giá trị mặc định AUTHID là DEFINER, được gọi là các quyền định nghĩa. Các quyền định nghĩa nghĩa là bất kỳ người nào có các đặc quyền thực thi chương trình lưu trữ chạy nó với những đặc quyền giống như tài khoản người dùng đã định nghĩa nó. Lựa chọn CURRENT_USER cho những người có đặc quyền thực thi gọi chương trình được lưu trữ và chạy nó trên chỉ dữ liệu user/schem của mình. Đây được gọi là các quyền viện

dẫn, nó mô tả tiến trình gọi một chương trình nguồn chung dựa trên các tài khoản và dữ liệu cá nhân.

Bạn nên tránh sử dụng DETERMINISTIC khi các hàm phụ thuộc vào trạng thái của các biến cấp Session. DETERMINISTIC là mệnh đề phù hợp nhất với các index dựa trên hàm và các view cụ thể hoá.

Mệnh đề PARALLEL_ENABLE nên được bật cho các hàm mà bạn dự định gọi từ những câu lệnh SQL mà có thể sử dụng các tính năng truy vấn song song. Bạn nên xem kỹ mệnh đề này cho những công dụng lưu trữ dữ liệu (data warehousing).

Mệnh đề PIPELINED cải thiện hiệu suất khi các hàm trả về các tập hợp (collection) như các bảng xếp lồng (nested table) hoặc VARRAYs. Bạn cũng nên lưu ý những cải tiến hiệu suất khi trả về các cursor tham chiếu hệ thống bằng cách sử dụng mệnh đề PIPELINED.

Mệnh đề RESULT_CACHE chỉ định một hàm được lưu trữ chỉ một lần trong SGA và các session chéo có sẵn. Nó mới trong Oracle 11g Database. Các hàm session chéo chỉ làm việc với các tham số hình thức chế độ IN.

Chương 6 trình bày các chi tiết thực thi về những mệnh đề này và đưa ra những ví dụ nhằm minh hoạ cách sử dụng chúng.

Bảng sau đây minh hoạ một nguyên mẫu thủ tục khối được đặt tên:

```

PROCEDURE procedure_name
( parameter1 {IN}[OUT] [NOCOPY] sql_data_type | plsq l_data_type
, parameter2 {IN}[OUT] [NOCOPY] sql_data_type | plsq l_data_type
, parameter(n+1) {IN}[OUT] [NOCOPY] sql_data_type | plsq l_data_type )
[ AUTHID {DEFINER | CURRENT_USER}]
declaration_statements
BEGIN
    execution_statements
[EXCEPTION]
    exception_handling_statements
END;
/

```

Chương 6 thảo luận những thủ tục chi phối các qui tắc. Chúng hoạt động như các hàm trong nhiều cách nhưng không thể trả về một kiểu dữ liệu. Điều này có nghĩa rằng bạn không thể sử dụng chúng làm các toán hạng thích hợp. Không giống như các hàm, các thủ tục phải được gọi bởi các khối PL/SQL. Các thủ tục (Procedures) vừa có thể truy vấn dữ liệu vừa xử lý dữ liệu. Các thủ tục cũng là các thương trình con nền tảng để

chuyển các giá trị qua lại những ngôn ngữ bên ngoài chẳng hạn như C, C++, Java và PHP.

Phần này đã trình bày và thảo luận cấu trúc cơ bản của những đơn vị chương trình PL/SQL. Các phần tiếp theo sẽ xem lại những tính năng gần đây trong Oracle 10g Database và những tính năng mới trong Oracle 11g Database.

Các tính năng mới của Oracle 10g

Một số thay đổi đã được đưa vào Oracle 10g Database. Không phải tất cả đều đã có sẵn khi ấn bản trước được viết bởi vì chúng đã không được phân phối cho đến lần tung ra thứ hai của cơ sở dữ liệu.

Những tính năng PL/SQL mới được đưa vào Oracle 11g bao gồm

- Các gói (package) cài sẵn
- Các cảnh báo thời gian biên dịch
- Biên dịch có điều kiện
- Các hành vi kiểu dữ liệu số
- Một trình biên dịch (compiler) PL/SQL được tối ưu hoá
- Các biểu thức thông thường
- Các lựa chọn trích dẫn
- Các toán tử tập hợp
- Các lỗi truy vết ngăn xếp
- Các chương trình lưu trữ PL/SQL bao bọc

Các mục nhỏ đề cập những tính năng gần đây được đưa vào Oracle 10g. Những mục này cũng tham khảo chéo các tính năng liên quan đến Oracle 11g Database mà sẽ được đề cập ở phần sau trong chương này.

Các gói cài sẵn

Bắt đầu với Oracle 10g phiên bản 2, bạn có thể truy cập một số gói (package) cài sẵn mới hoặc cải tiến. Ba gói đáng được đề cập ở đây là

- DBMS_SCHEDULER. Thay thế DBMS_JOB cài sẵn và cung cấp chức năng mới để lập thời biểu và thực thi các công việc theo lô (batch job).
- DBMS_CRYPTO. Bây giờ có khả năng mã hoá và giải mã những đối tượng lớn và hỗ trợ sự toàn cầu hoá qua nhiều tập hợp ký tự.
- DBMS_MONITOR. Cung cấp một API hỗ trợ việc truy vết và thu thập thống kê bằng các session.

Các cảnh báo thời gian biên dịch

Bắt đầu với Oracle 10g, phiên bản 1, bạn có thể nhận thấy rõ hiệu suất của các chương trình PL/SQL bằng cách bật tham số PLSQL_WARNINGS trong các trường hợp phát triển. Bạn có thể thiết lập điều này cho một session hoặc cơ sở dữ liệu. Cái trước là những thói quen được đề nghị do hao phí được tạo ra trên cơ sở dữ liệu. Bạn xác lập tham số này bằng cách sử dụng lệnh sau đây:

```
ALTER SESSION SET plsql_warnings = 'enable:all';
```

Biên dịch có điều kiện

Bắt đầu với Oracle 10g phiên bản 2, bạn có thể sử dụng sự biên dịch có điều kiện (conditional compilation). Biên dịch có điều kiện cho phép đưa vào logic gỡ rối hoặc logic chuyên dụng vốn chạy chỉ khi các biến session được thiết lập. Lệnh sau đây thiết lập một biến thời gian biên dịch PL/SQL DEBUG bằng 1:

```
ALTER SESSION SET PLSQL_CCFLAGS = 'debug:1';
```

Lệnh này xác lập biến thời gian biên dịch PL/SQL DEBUG bằng 1. Bạn nên chú ý rằng cờ thời gian biên dịch không nhạy kiểu chữ. Bạn cũng có thể xác lập các biến thời gian biên dịch sang true hoặc false để chúng hành động như các biến Boolean. Khi muốn xác lập nhiều cờ biên dịch có điều kiện, bạn cần sử dụng cú pháp sau đây:

```
ALTER SESSION SET PLSQL_CCFLAGS = 'name1:value1 [, name(n+1):value(n+1)]';
```

Các tham số biên dịch có điều kiện được lưu trữ dưới dạng các cặp tên và giá trị trong tham số cơ sở dữ liệu PL/SQL_CCFLAG. Chương trình sau đây sử dụng các chỉ lệnh \$IF, \$THEN, \$ELSE, \$ELSIF, \$ERROR, và \$END tạo một khối mã biên dịch có điều kiện:

```
BEGIN
    $IF $$DEBUG = 1 $THEN
        dbms_output.put_line('Debug Level 1 Enabled.');
```

```
    $END
END;
```

```
/
```

Các khối mã có điều kiện khác với các khối mã if-then-else chuẩn. Đáng chú ý nhất chỉ lệnh \$END đóng khối thay vì END_IF và dấu chấm phẩy như đề cập trong chương 4. Bạn cũng nên chú ý ký hiệu biểu thị một biến thời gian biên dịch có điều kiện PL/SQL.

Các qui tắc chi phối việc biên dịch có điều kiện được thiết lập bằng bộ phân tích cú pháp (parser) SQL. Bạn không thể sử dụng việc biên dịch có điều kiện trong các kiểu đối tượng SQL. Giới hạn này cũng áp dụng vào các bảng xếp lồng (nested table) và VARRAYS (các bảng vô hướng). Biên dịch có điều kiện khác nhau trong các hàm và thủ tục. Hành vi thay đổi việc hàm hoặc thủ tục có một danh sách tham số hình thức hay không. Bạn có thể sử dụng sự biên dịch có điều kiện sau dấu ngoặc mở của một danh sách tham số hình thức như sau:

```
CREATE OR REPLACE FUNCTION conditional_type
( magic_number $IF $$DEBUG = 1 $THEN SIMPLE_NUMBER $ELSE NUMBER
$END )
RETURN NUMBER IS
BEGIN
    RETURN magic_number;
END;
/
```

Hoặc, bạn có thể sử dụng chúng sau từ khoá AS hay IS trong các hàm hoặc thủ tục không tham số. Chúng cũng có thể được sử dụng cả bên trong danh sách tham số hình thức và sau AS hay IS trong các hàm hoặc thủ tục tham số.

Biên dịch có điều kiện chỉ xảy ra sau từ khoá BEGIN trong các trigger và các đơn vị chương trình khối nặc danh. Chú ý bạn không thể đóng gói một placeholder hoặc một biến liên kết bên trong một khối biên dịch có điều kiện.

Chương 4 chứa các ví dụ sử dụng những kỹ thuật biên dịch có điều kiện.

Màn hình vi kiểu dữ liệu số

Bắt đầu với Oracle 10g phiên bản 1, bây giờ cơ sở dữ liệu sử dụng số học máy cho BINARY_INTEGER, INTEGER, INT, NATURAL, NATURALN, PLS_INTEGER, POSTIVE, POSITIVEN và SIGNTYPE. Điều này có nghĩa bây giờ chúng sử dụng cùng một cách giải quyết như kiểu dữ liệu BINARY_INTEGER. Trong các phiên bản trước của cơ sở dữ liệu, những kiểu dữ liệu này làm việc giống như kiểu dữ liệu NUMBER và chúng sử dụng cùng một thư viện toán học C như kiểu dữ liệu NUMBER. Các phiên bản mới của những kiểu dữ liệu này có thể được so sánh với sự vô hạn hoặc NaN (not a number - không phải một số).

Một khuyết điểm của sự thay đổi này là bây giờ chúng sử dụng tính chính xác số, không phải tính chính xác thập phân. Các ứng dụng tài chính nên tiếp tục sử dụng kiểu dữ liệu NUMBER vì lý do đó.

Một `BINARY_FLOAT` độ chính xác đơn và một `BINARY_DOUBLE` độ chính xác kép cũng được cung cấp trong Oracle 10g Database. Chúng lý tưởng cho các phép toán hoặc phép tính khoa học.

Trình biên dịch PL/SQL được tối ưu hoá

Bắt đầu với Oracle 10g phiên bản 1, bây giờ cơ sở dữ liệu tối ưu hoá việc biên dịch PL/SQL. Điều này được thiết lập theo mặc định và áp dụng vào cả p-code được biên dịch và mã PL/SQL được biên dịch riêng. Bạn huỷ xác lập hoặc chỉnh sửa tính tháo vát của bộ tối ưu hoá bằng cách xác lập lại tham số `PLSQL_OPTIMIZE_LEVEL`. Bảng 1.1 trình bày rõ ba giá trị có thể có cho tham số.

Bạn có thể tắt chế độ tối ưu hoá session bằng cách sử dụng

```
ALTER SESSION SET plsql_optimize_level = 0;
```

Bạn cũng có thể xác lập mức độ tối ưu hoá cho một thủ tục. Nguyên mẫu là:

```
ALTER PROCEDURE some_procedure COMPILE plsql_optimize_level = 1;
```

Sau khi bạn đã xác lập mức độ tối ưu hoá, bạn có thể sử dụng mệnh đề `RESUE SETTINGS` để tái sử dụng xác lập trước như

```
ALTER PROCEDURE some_procedure COMPILE REUSE SETTINGS;
```

Trong khi điều này mang tính thông tin, nói chung bạn nên để nó như mặc định. Mã tối ưu luôn chạy nhanh hơn mã không tối ưu.

Ghi chú

`PLSQL_OPTIMIZE_LEVEL` phải được xác lập tại 2 trở lên để việc inlining chương trình con tự động xảy ra trong cơ sở dữ liệu Oracle 10g hoặc 11g.

Bảng 1.1 Các giá trị `PLSQL_OPTIMIZE_LEVEL` có sẵn

Mức độ tối ưu	Ý nghĩa tối ưu
0	Không tối ưu
1	Tối ưu tương đối, có thể loại bỏ mã hoặc các ngoại lệ không được sử dụng.
2 (mặc định)	Tối ưu mạnh, có thể tái sắp xếp dòng mã nguồn.

Các biểu thức thông thường

Bắt đầu với Oracle 10g phiên bản 1, bây giờ cơ sở dữ liệu hỗ trợ một tập hợp các hàm biểu thức thông thường. Bạn có thể truy cập chúng như nhau trong câu lệnh SQL hoặc các đơn vị chương trình PL/SQL. Chúng là:

- **REGEXP_LINE.** Hàm này tìm một chuỗi để tìm một mẫu biểu thức thông thường tương hợp
- **REGEXP_INSTR.** Hàm này tìm vị trí bắt đầu của một mẫu biểu thức thông thường tương hợp.
- **REGEXP_SUBSTR.** Hàm này tìm một chuỗi con sử dụng một mẫu biểu thức thông thường tương hợp.
- **REGEXP_REPLACE.** Hàm này thay thế một chuỗi con sử dụng một mẫu biểu thức thông thường tương hợp.

Lựa chọn trích dẫn

Bắt đầu với Oracle 10g phiên bản 1, bây giờ cơ sở dữ liệu cho bạn thay thế dấu trích dẫn đơn quen thuộc bằng một ký hiệu trích dẫn khác. Điều này hữu dụng khi bạn có một chùm các dấu phẩy trên (') trong một chuỗi mà riêng lẻ từng dấu đòi hỏi phải có một dấu trích dẫn đơn khác. Cách cũ như sau:

```
SELECT 'It's a bird, no plane, no it can't be ice cream!' AS phrase FROM dual;
```

Cách mới là

```
SELECT q'(It's a bird, no plane, no it can't be ice cream!)' AS phrase FROM dual;
```

Cả hai cách này mang lại kết quả sau đây:

```
PHRASE
```

```
It's a bird, no plane, no it can't be ice cream!
```

Có những cơ hội sử dụng cú pháp mới hơn và tiết kiệm thời gian, nhưng cách cũ cũng tiếp tục có hiệu quả. Cách cũ được hiểu và có thể mang chuyển rộng hơn.

Các toán tử tập hợp

Bắt đầu với Oracle 10g phiên bản 1, bây giờ cơ sở dữ liệu hỗ trợ các toán tử tập hợp cho các bảng xếp lồng (nested table). Những toán tử này bao gồm **MULTISET EXCEPT**, **MULTISET INTERSECT**, **MULTISET UNION** và **MULTISET UNION DISTINCT**. **MULTISET UNION** thực thi như toán tử **UNION ALL** quen thuộc. Nó trả về hai bản sao của mọi thứ trong giao giữa hai tập hợp và một bản sao của các phần bù tương đối. **MULTISET UNION DISTINCT** làm việc như toán tử **UNION**. Nó trả về một bản sao của mọi thứ bằng cách thực thi một phép toán phân loại gia tăng. Chương 7 đề cập đến những toán tử này.

Các lỗi truy vết ngăn xếp

Bắt đầu với Oracle 10g phiên bản 1, cuối cùng bạn có thể định dạng các vết ngăn xếp (stack trace). Các vết ngăn xếp tạo một danh sách các lỗi từ lệnh gọi ban đầu đến vị trí nơi lỗi được đưa ra. Bạn sử dụng hàm `DBMS_UTILITY.FORMAT_ERROR_BACKTRACE` để tạo một ngăn xếp. Bạn cũng có thể gọi `FORMAT_CALL_STACK` hoặc `FORMAT_ERROR_STACK` từ cùng một gói để làm việc với những ngoại lệ được đưa ra.

Sau đây là một ví dụ đơn giản:

```

DECLARE
    local_exception EXCEPTION;
    FUNCTION nested_local_function
    RETURN BOOLEAN IS
        retval BOOLEAN := FALSE;
    BEGIN
        RAISE local_exception;
        RETURN retval;
    END;
BEGIN
    IF nested_local_function THEN
        dbms_output.put_line('No raised exception');
    END IF;
EXCEPTION
    WHEN others THEN
        dbms_output.put_line('DBMS_UTILITY.FORMAT_CALL_STACK');
        dbms_output.put_line('-----');
        dbms_output.put_line(dbms_utility.format_call_stack);
        dbms_output.put_line('DBMS_UTILITY.FORMAT_ERROR_BACKTRACE');
        dbms_output.put_line('-----');
        dbms_output.put_line(dbms_utility.format_error_backtrace);
        dbms_output.put_line('DBMS_UTILITY.FORMAT_ERROR_STACK');
        dbms_output.put_line('-----');
        dbms_output.put_line(dbms_utility.format_error_stack);
END;
/

```

Script này tạo kết quả sau đây:

```
DBMS_UTILITY.FORMAT_CALL_STACK
-----
----- PL/SQL Call Stack -----
object      line object
handle      number
name
20909240     18 anonymous block
DBMS_UTILITY.FORMAT_ERROR_BACKTRACE
-----

ORA-06512: at line 7
ORA-06512: at line 11
DBMS_UTILITY.FORMAT_ERROR_STACK
-----

ORA-06510: PL/SQL: unhandled user-defined exception
```

Có thể bạn sẽ thấy `FORMAT_ERROR_BACKTRACE` hữu dụng nhất. Nó bắt giữ dòng nơi lỗi đầu tiên xảy ra ở trên cùng và sau đó di chuyển lùi lại qua các lệnh gọi cho đến khi nó đi đến lệnh gọi ban đầu. Các số dòng và tên chương trình hiển thị cùng với nhau khi các khối được đặt tên liên quan trong một ngăn xếp sự kiện (event stack). Chương 5 trình bày thêm chi tiết về việc quản lý lỗi.

Các chương trình lưu trữ PL/SQL bao bọc

Bắt đầu với Oracle 10g phiên bản 2, bây giờ cơ sở dữ liệu hỗ trợ khả năng bao bọc hoặc obfuscate (làm xáo trộn) các chương trình lưu trữ PL/SQL. Điều này được thực hiện bằng cách sử dụng thủ tục `CREATE_WRAPPED` của gói `DBMS_DDL`. Bạn sử dụng nó như sau:

```
BEGIN
  dbms_ddl.create_wrapped(
    'CREATE OR REPLACE FUNCTION hello_world RETURN STRING AS '
    | 'BEGIN '
    | ' RETURN "Hello World!"; '
    | 'END;');
END;
/
```

Sau khi tạo thủ tục, bạn có thể truy vấn nó bằng cách sử dụng định dạng cột SQL*Plus và mẫu truy vấn sau đây:

```
SQL> COLUMN message FORMAT A20 HEADING "Message"
SQL> SELECT hello_world AS message FROM dual;
Message
```

```
-----
Hello World!
```

Bạn có thể mô tả hàm để kiểm tra chữ ký và kiểu trả về của nó:

```
SQL> DESCRIBE hello_world

FUNCTION hello_world RETURNS VARCHAR2
```

Bất kỳ nỗ lực nhằm kiểm tra các thao tác chi tiết của nó sẽ dẫn đến một kết quả xáo trộn. Bạn có thể test điều này bằng cách truy vấn phần thực thi hàm lưu trữ trong cột TEXT của bảng USER_SOURCE như sau:

```
SQL> COLUMN text FORMAT A80 HEADING "Source Text"
SQL> SET PAGESIZE 49999

SQL> SELECT text FROM user_source WHERE name = 'HELLO_WORLD';
```

Kết quả sau đây được trả về:

```
FUNCTION hello_world wrapped
a000000
369
abcd
. . . et cetera . . .
```

Đây là một tiện ích rất hữu dụng để che giấu các chi tiết thực thi khỏi những cặp mắt tò mò.

Những tính năng mới của Oracle 11g

Những tính năng PL/SQL mới được đưa vào Oracle 11g bao gồm

- Inlining chương trình con tự động
- Một câu lệnh continue
- Một cache kết quả hàm PL/SQL session chéo
- Các cải tiến SQL động
- Các lệnh gọi SQL ký hiệu hỗn hợp, được đặt tên, và theo vị trí
- Một bộ chứa nối kết (connection pool) đa tiến trình
- PL/SQL Hierarchical Profile
- Bây giờ PL/SQL Native Compiler tạo mã riêng

- PL/Scope
- Các cải tiến biểu thức thông thường
- Một kiểu dữ liệu SIMPLE_INTEGER
- Các lệnh gọi trình tự trực tiếp trong các câu lệnh SQL

Những cải tiến này được xem lại ngắn gọn trong các mục tiếp theo. Chương 3 đề cập đến kiểu dữ liệu SIMPLE_INTEGER. Chương 4 đề cập đến câu lệnh continue. Chương 6 trình bày cache kết quả hàm PL/SQL session chéo và các lệnh gọi ký hiệu hỗn hợp, định danh và vị trí. Inlining chương trình con tự động và PL/SQL Native Compiler được đề cập trong chương 9. Chương 16 đề cập đến việc phát triển ứng dụng web và bộ chứa nối kết đa tiến trình.

Inlining chương trình con tự động

Inlining một chương trình con sẽ thay thế lệnh gọi đến chương trình con bên ngoài bằng một bản sao của chương trình con. Điều này hầu như luôn nâng cao hiệu suất chương trình. Bạn có thể ra lệnh trình biên dịch inlining các chương trình con bằng cách sử dụng chỉ lệnh biên dịch PRAGMA INLINE trong PL/SQL bắt đầu với Oracle 11g Database. Bạn phải xác lập PRAGMA khi tham số PLSQL_OPTIMIZE_LEVEL được xác lập sang 2.

Ví dụ, bạn có một hàm lưu trữ ADD_NUMBERS trong một schema, bạn có thể ra lệnh một đơn vị chương trình PL/SQL inline lệnh gọi đến hàm ADD_NUMBERS. Điều này rất hữu dụng khi bạn gọi hàm ADD_NUMBERS trong một vòng lặp như trong ví dụ này:

```
CREATE OR REPLACE PROCEDURE inline_demo
( a NUMBER
, b NUMBER ) IS
  PRAGMA INLINE(add_numbers,'YES');
BEGIN
  FOR i IN 1..10000 LOOP
    dbms_output.put_line(add_function(8,3));
  END LOOP;
END;
/
```

Cơ sở dữ liệu tự động hoá các lựa chọn inlining khi bạn xác lập tham số PLSQL_OPTIMIZE_LEVEL sang 3. Nói chung, điều này giúp bạn khỏi phải nhận dạng chúng khi việc inline các lệnh gọi hàm thì thích hợp. Tuy nhiên, đây chỉ là những đề nghị cho trình biên dịch. Bạn nên để bộ máy (engine) tối ưu hoá mã trong quá trình biên dịch.

Câu lệnh Continue

Câu lệnh CONTINUE cuối cùng đã được thêm vào ngôn ngữ PL/SQL. Một số có những cảm xúc hỗn hợp. Có những ý kiến cho rằng câu lệnh continue dẫn đến việc lập trình kém tối ưu hơn, nhưng nói chung nó đơn giản hoá các cấu trúc vòng lặp.

Câu lệnh CONTINUE báo hiệu một sự kết thúc tức thì một sự lặp lại vòng lặp và quay trở về câu lệnh đầu tiên trong vòng lặp. Khởi nặc danh sau đây minh hoạ việc sử dụng một câu lệnh continue khi index vòng lặp là một số chẵn:

```
BEGIN
    FOR i IN 1..5 LOOP
        dbms_output.put_line('First statement, index is ['|| i ||'].');
        IF MOD(i,2) = 0 THEN
            CONTINUE;
        END IF;
        dbms_output.put_line('Second statement, index is ['|| i ||'].');
    END LOOP;
END;
/
```

Hàm MOD trả về một zero khi chia bất kỳ số chẵn, do đó dòng thứ hai không bao giờ được in ra bởi vì câu lệnh CONTINUE huỷ bỏ phần còn lại của vòng lặp. Chương 4 sẽ đề cập thêm chi tiết về việc sử dụng lệnh này.

Cache kết quả hàm PL/SQL session chéo

Cache kết quả hàm PL/SQL session chéo là một cơ cấu nhằm chia sẻ các hàm được truy cập gần đây trong SGA giữa các session. Trước Oracle 11g Database, mỗi lệnh gọi đến một hàm có các tham số thật sự giống nhau hoặc các giá trị thời gian chạy (run-time) đã được lưu trữ một lần trên mỗi session. Giải pháp duy nhất cho chức năng đó đòi hỏi bạn viết mã các phương thức truy cập.

Bạn chỉ định một trong hai phương thức sau đây để lưu trữ kết quả:

RESULT_CACHE clause

hoặc

RESULT_CACHE RELIES_ON (table_name)

Mệnh đề `RELIES_ON` đặt một giới hạn trên kết quả lưu trữ. Bất kỳ thay đổi đối với bảng tham chiếu sẽ vô hiệu hoá hàm cũng như bất kỳ hàm, thủ tục hoặc view vốn phụ thuộc vào hàm.

Hao phí khi gọi hàm lần đầu tiên không khác gì so với hao phí do gọi một kết quả không được lưu trữ. Tương tự, cache sẽ ra khỏi SGA khi nó không còn được gọi tích cực bởi các session nữa.

Các cải tiến SQL động

SQL động vẫn có hai loại trong Oracle 11g Database. Bạn có Native Dynamic SQL còn được gọi là NDS và gói cài sẵn `DBMS_SQL`. Cả hai đã được cải tiến trong phiên bản này.

Native Dynamic SQL

Trong Oracle 11g, bây giờ Native dynamic SQL hỗ trợ các câu lệnh động lớn hơn 32 KB bằng cách chấp nhận CLOB. Bạn truy cập chúng thay cho một câu lệnh SQL bằng cách sử dụng cú pháp sau đây:

```
OPEN cursor_name FOR dynamic_string;
```

Chuỗi động có thể là CHAR, VARCHAR2 hoặc CLOB. Nó không thể là Unicode NCHAR hoặc NVARCHAR2. Điều này loại bỏ giới hạn trước vốn đã hạn chế kích cỡ của các chuỗi được tạo động.

Gói cài sẵn DBMS_SQL

Một số thay đổi đã cải thiện tiện ích của một gói `DBMS_SQL`. Bắt đầu Oracle 11g, bây giờ bạn sử dụng tất cả kiểu dữ liệu được hỗ trợ bởi NDS. Bạn cũng có thể sử dụng thủ tục `PARSE` để làm việc với các câu lệnh lớn hơn 32KB. Điều này được thực hiện bằng cách sử dụng một kiểu dữ liệu CLOB. CLOB thay thế giải pháp trước vốn đã sử dụng một bảng các kiểu dữ liệu VARCHAR2 (thường là VARCHAR2A hoặc VARCHAR2S). Thật may thay, gói `DBMS_SQL` tiếp tục hỗ trợ giải pháp, nhưng bạn nên xem xét di chuyển đến một giải pháp tốt hơn.

`DBMS_SQL` bổ sung hai hàm mới: hàm `TO_REFCURSOR` và `TO_CURSOR_NUMBER`. Chúng cho bạn chuyển các cursor tham chiếu đến các cursor và ngược lại. Hiển nhiên cũng có một số lời khuyên về cách sử dụng các hàm này. Bạn phải mở cursor hoặc cursor tham chiếu hệ thống trước khi sử dụng chúng và sau khi chạy chúng bạn không thể truy cập các cấu trúc cũ của chúng. Về cơ bản, mã gán lại tham chiếu tương tác từ cursor sang cursor tham chiếu hệ thống hoặc từ cursor tham chiếu hệ thống sang cursor.

Sau cùng nhưng chắc chắn không kém phần quan trọng, bây giờ bạn có thể thực hiện các thao tác liên kết hàng loạt trên các kiểu tập hợp do người dùng định nghĩa. Các kiểu tập hợp có thể là các mảng vô hướng

(scalar array). Trước đó bạn bị giới hạn chỉ trong các kiểu được định nghĩa bởi thông số gói DBMS_SQL.

Các lệnh gọi ký hiệu tên hỗn hợp và vị trí

Oracle 11g Database mang lại những thay đổi trong cách làm việc của ký hiệu tên và ký hiệu vị trí trong SQL và PL/SQL. Bây giờ chúng thật sự làm việc theo cùng một cách trong cả SQL và PL/SQL. Điều này sửa chữa một tật xấu đã có từ lâu trong cơ sở dữ liệu Oracle.

Các lệnh gọi PL/SQL

Trước đó bạn đã có hai lựa chọn. Bạn có thể liệt kê tất cả tham số theo thứ tự vị trí của chúng hoặc gửi một số đến tất cả tham số bằng tham chiếu định danh. Bây giờ bạn có thể sử dụng tham chiếu vị trí, tham chiếu định danh hoặc hỗn hợp cả hai. Hàm sau đây sẽ cho bạn thử nghiệm các phương pháp khác nhau. Hàm chấp nhận ba tham số tùy chọn và trả về tổng của ba số.

```
CREATE OR REPLACE FUNCTION add_three_numbers
    ( a NUMBER := 0, b NUMBER := 0, c NUMBER := 0 ) RETURN NUMBER IS
BEGIN
    RETURN a + b + c;
END;
/
```

Ba phần con đầu tiên hướng dẫn cách bạn gọi bằng cách sử dụng ký hiệu vị trí, ký hiệu định danh và ký hiệu hỗn hợp. Trong những ký hiệu này, bạn cung cấp các tham số thật sự cho từng tham số hình thức được định nghĩa bởi chữ ký hàm.

Bạn cũng có thể loại một hoặc nhiều giá trị bởi vì tất cả tham số hình thức được định nghĩa là tùy chọn, nghĩa là chúng có những giá trị mặc định. Điều này được thực hiện trong phần "ký hiệu loại trừ".

Ký hiệu vị trí

Bạn có thể gọi hàm sử dụng ký hiệu vị trí bằng

```
BEGIN
    dbms_output.put_line(add_three_numbers(3,4,5));
END;
/
```

Ký hiệu định danh

Bạn gọi hàm sử dụng ký hiệu định danh bằng

```
BEGIN
    dbms_output.put_line(add_three_numbers(c => 4,b => 5,c => 3));
END;
/
```

Ký hiệu hỗn hợp

Bạn gọi hàm sử dụng hỗn hợp cả ký hiệu vị trí lẫn ký hiệu định danh bằng

```
BEGIN
    dbms_output.put_line(add_three_numbers(3,c => 4,b => 5));
END;
/
```

Có một giới hạn về ký hiệu hỗn hợp. Tất cả tham số thật sự của ký hiệu vị trí phải xuất hiện trước tiên và theo cùng một thứ tự như chúng được định nghĩa bằng chữ ký hàm. Bạn không thể cung cấp một giá trị vị trí sau một giá trị định danh.

Ký hiệu loại trừ

Như được đề cập, bạn cũng có thể loại trừ một hoặc nhiều tham số thật sự khi các tham số hình thức được định nghĩa là tùy chọn. Tất cả tham số trong hàm ADO_THREE_NUMBERS thì tùy chọn. Ví dụ sau đây chuyển một giá trị đến tham số thứ nhất theo tham chiếu vị trí và tham số thứ hai theo tham chiếu định danh:

```
BEGIN
    dbms_output.put_line(add_three_numbers(3,c => 4));
END;
/
```

Khi bạn chọn không cung cấp một tham số thật sự, như thể bạn đang chuyển một giá trị rỗng. Đây được gọi là ký hiệu loại trừ. Trong nhiều năm, đây đã là đề nghị liệt kê các biến tùy chọn sau cùng trong các chữ ký hàm và thủ tục. Bây giờ bạn có thể loại trừ một hoặc hai nhưng không phải tất cả tham số tùy chọn. Đây là một cải tiến tuyệt vời, nhưng phải chú ý cách bạn khai thác nó như thế nào.

Ký hiệu lệnh gọi SQL

Trước đó bạn chỉ có một lựa chọn. Bạn đã phải liệt kê tất cả tham số theo thứ tự vị trí của chúng bởi vì bạn không thể sử dụng tham chiếu định danh trong SQL. Điều này được sửa chữa trong Oracle 11g. Bây giờ bạn có thể gọi chúng như bạn gọi từ một khối PL/SQL. Dòng sau đây minh họa ký hiệu hỗn hợp trong một lệnh gọi SQL:

```
SELECT add_three_numbers(3,c => 4,b => 5) FROM dual;
```

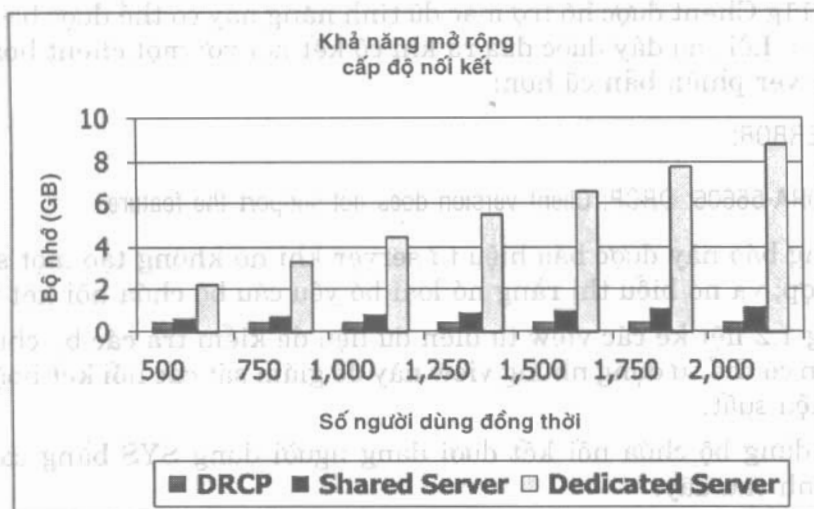
Như trong các phiên bản trước, bạn chỉ có thể gọi các hàm có các biến chỉ chế độ IN từ các câu lệnh SQL. Bạn không thể gọi một hàm từ SQL khi bất kỳ tham số hình thức của nó được định nghĩa là các biến chỉ chế độ IN OUT hoặc OUT. Đây là do bạn phải chuyển một tham chiếu biến khi một tham số có một chế độ OUT. Các hàm trả một tham chiếu trở về các biến chế độ OUT được chuyển dưới dạng các tham số thật sự.

Bộ chứa nối kết đa tiến trình

Enterprise JavaBeans (EJB) trở nên tốt hơn với việc tung ra bộ chứa nối kết đa tiến trình trong Oracle 11g Database. Nó chính thức là Database Resident Connection Pooling (DRCP). Tính năng này cho bạn quản lý một bộ chứa nối kết phía server có khả năng mở rộng cấp độ nhiều hơn. Trước phiên bản này, bạn tận dụng các tiến trình server chia sẻ hoặc một Java Serylet đa chuỗi liên kết.

Bộ chứa nối kết đa tiến trình giảm đáng kể footprint bộ nhớ trên tầng cơ sở dữ liệu và tăng khả năng mở rộng cấp độ của tầng giữa và tầng cơ sở dữ liệu. Một nối kết cơ sở dữ liệu chuẩn đòi hỏi 4.4 MB bộ nhớ thật; 4 MB được cấp phát cho nối kết vật lý và 400 KB cho user session (phiên làm việc người dùng). Do đó, 500 nối kết đồng thời chuyên dụng đòi hỏi khoảng 2.2 GB bộ nhớ thật. Một mô hình server chia sẻ có tính mở rộng cấp độ hơn và đòi hỏi chỉ 600 MB bộ nhớ thật cho cùng một số người dùng đồng thời. 80% bộ nhớ đó được quản lý trong Shared Global Area (SGA) của Oracle. Database Resident Connection Pooling mở rộng cấp độ tốt hơn và đòi hỏi chỉ 400 MB bộ nhớ thật. Rõ ràng đối với các ứng dụng trên web, DRCP là giải pháp ưu tiên đặc biệt khi sử dụng các nối kết thường trực OC18.

Hành vi của những mô hình này ấn định khả năng mở rộng cấp độ tương ứng của chúng. Hình 1.2 minh họa bằng đồ thị việc sử dụng bộ nhớ cho ba mô hình từ 500 đến 2000 người dùng đồng thời.



Hình 1.2 Khả năng mở rộng cấp độ nối kết

Tính năng mới được phân phối bởi gói cài sẵn DBMS_CONNECTION_POOL mới. Gói này cho bạn bắt đầu, dừng và cấu hình các tham số bộ chứa nối kết chẳng hạn như giới hạn kích cỡ và thời gian. Bạn bắt đầu bộ chứa nối kết dưới dạng SYS bằng cách sử dụng lệnh sau đây:

```
SQL> EXECUTE DBMS_CONNECTION_POOL.START_POOL();
```

Bạn phải cho phép file tnsnames.ora hỗ trợ nối kết với bộ chứa chia sẻ. Đoạn sau đây bật một bộ mô tả nối kết bộ chứa chia sẻ (shared pool connect descriptor), miễn là bạn thay thế đúng một hostname, domain và số cổng lắng nghe Oracle:

```
ORCLCP =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)
      (HOST = hostname.domain)
      (PORT = port_number)
    )
    (CONNECT_DATA = (SERVER = POOLED)
      (SERVICE_NAME = orcl)
    )
  )
```

Khoá SERVER trong bộ mô tả nối kết (connect descriptor) hướng các nối kết sang dịch vụ hợp nhất nối kết (connection pooling). Bạn có thể chỉ sử dụng bộ chứa nối kết khi bạn có một Oracle 11g Database hoặc

Oracle 11g Client được hỗ trợ mặc dù tính năng này có thể được backport tiếp theo. Lỗi sau đây được đưa ra khi cố kết nối với một client hoặc thư viện server phiên bản cũ hơn:

ERROR:

ORA-56606: DRCP: Client version does not support the feature

Thông báo này được báo hiệu từ server khi nó không tạo một socket thích hợp; và nó biểu thị rằng nó loại bỏ yêu cầu bộ chứa nối kết từ xa.

Bảng 1.2 liệt kê các view từ điển dữ liệu để kiểm tra các bộ chứa nối kết. Bạn có thể sử dụng những view này để giám sát các nối kết hoặc đặc điểm hiệu suất.

Bạn dùng bộ chứa nối kết dưới dạng người dùng SYS bằng cách sử dụng lệnh sau đây:

```
SQL> EXECUTE DBMS_CONNECTION_POOL.STOP_POOL();
```

View	Mô tả
DBA_CPOOL_INFO	Các nối kết trạng thái (status) tối đa (maximum) và tối thiểu (minimum) và các timeout rảnh có sẵn trong view này cho mỗi bộ chứa nối kết.
V\$CPOOL_STAT	Số yêu cầu session, số lần một session vốn khớp với một yêu cầu được tìm thấy trong bộ chứa và tổng thời gian chờ trên mỗi yêu cầu session có sẵn trong view này.
V\$CPOOL_CC_STATS	Số liệu thống kê cấp nối kết cho mỗi instance của bộ chứa nối kết.

Bảng 1.2 Các view từ điển dữ liệu hợp nhất nối kết.

Dường như phiên bản ban đầu sẽ chỉ hỗ trợ một bộ chứa nối kết. Tên bộ chứa nối kết là SYS_DEFAULT_CONNECTION_POOL. Bạn cũng có ba thủ tục khác trong gói DBMS_CONNECTION_POOL để quản lý bộ chứa nối kết: các thủ tục ALTER_PARAM(), CONFIGURE_POOL(), và RESTORE_DEFAULTS(). Bạn thay đổi một tham số bộ chứa nối kết bằng thủ tục ALTER_PARAM(). Khi bạn muốn thay đổi nhiều tham số thành tất cả tham số, bạn sử dụng thủ tục CONFIGURE_POOL(). Thủ tục RESTORE_DEFAULTS() xác lập lại tất cả bộ chứa nối kết sang những tham số mặc định của chúng.

Tính năng Oracle 11g Database mới này chắc chắn nâng cao khả năng mở rộng cấp độ của các ứng dụng web. Nó là một tính năng quan trọng trao quyền tính năng nối kết thường trực được đưa vào trong các thư viện OC18 trong cơ sở dữ liệu Oracle 10g phiên bản 2.

PL/SQL Hierarchical Profile

Hierarchical profile được đưa vào Oracle 11g Database cho bạn thấy tất cả thành phần trong một ứng dụng thực thi như thế nào. Điều này khác với một profile không phân cấp (phẳng) vốn đơn giản ghi chép thời gian đã dành ra cho mỗi module. Các hierarchical profile đi theo chu kỳ thực thi từ chương trình chứa xuống đến chương trình con thấp nhất.

PL/SQL Hierarchical Profile làm những điều sau đây:

- Nó báo cáo profile thực thi động của chương trình PL/SQL vốn được tổ chức bằng các lệnh gọi chương trình con.
- Nó phân chia các thời gian thực thi SQL và PL/SQL và báo cáo chúng một cách riêng biệt.
- Nó không đòi hỏi chuẩn bị nguồn đặc biệt hoặc thời gian biên dịch như PRAGMA được bắt buộc để đề nghị inlining.
- Nó lưu trữ kết quả trong một tập hợp bằng cơ sở dữ liệu mà bạn có thể sử dụng để phát triển các công cụ lập báo cáo hoặc sử dụng công cụ dòng lệnh plshprof để tạo các report HTML đơn giản.
- Bảng tổng kết cấp chương trình con bằng cách thông tin về số lệnh gọi chương trình con, thời gian bỏ ra trong các chương trình con hoặc các cây con của chúng và thông tin chi tiết giữa các module.

PL/SQL Native Compiler tạo mã riêng

Việc biên dịch PL/SQL Native thay đổi trong Oracle 11g Database. Không giống như các phiên bản trước, trong đó PL/SQL được biên dịch trước tiên thành mã C rồi mới được biên dịch sau, bây giờ bạn có thể biên dịch trực tiếp. Tốc độ thực thi của mã cuối cùng trong một số trường hợp nhanh gấp hai lần hoặc có độ lớn hơn.

Oracle đề nghị bạn chạy tất cả PL/SQL trong chế độ NATIVE hoặc INTERPRETED. Chế độ INTERPRETED là chế độ mặc định của cơ sở dữ liệu và các module PL/SQL được lưu trữ dưới dạng text rõ ràng hoặc text bao bọc. Bạn có thể xem các chương trình lưu trữ bằng cách truy vấn các bảng tự điển dữ liệu ALL_SOURCE, DBA_SOURCE hoặc USER_SOURCE. Mã chế độ NATIVE được biên dịch thành một dạng trung gian trước khi được rút giảm thành mã dành riêng cho máy. Một bản sao của mã cũng được lưu trữ trong từ điển dữ liệu trong khi thư viện được đặt trong một thư mục bên ngoài. Bạn ánh xạ thư mục vật lý sang thư mục ảo được định nghĩa bởi tham số cơ sở dữ liệu PLSQL_NATIVE_LIBRARY_DIR.

Mã được biên dịch riêng có lợi khi thời gian chạy PL/SQL chậm. Điều này có thể xảy ra với mã đòi hỏi nhiều khả năng điện toán nhưng nói chung thời gian trễ thực thi xảy ra do việc xử lý câu lệnh SQL. Bạn nên sử dụng PL/SQL Hierarchical Profile mới để quyết định xem hỗ trợ nỗ lực chuyển đổi của bạn có một lợi ích đáng kể nào hay không.

PL/Scope

PL/Scope là một công cụ được điều khiển bằng trình biên dịch, cho phép kiểm tra các định danh (identifier) và các hành vi tương ứng của chúng trong Oracle 11g Database. Nó được tắt theo mặc định. Bạn có thể mở nó cho cơ sở dữ liệu hoặc session. Bạn nên xem xét mở nó cho session chứ không phải cơ sở dữ liệu. Bạn mở nó bằng cách sử dụng lệnh sau đây:

```
ALTER SESSION SET PLScope_SETTINGS = 'IDENTIFIERS:ALL';
```

Tiện ích PL/Scope không thu thập số liệu thống kê cho đến khi bạn mở nó. Số liệu thống kê cho bạn kiểm tra các chương trình sử dụng các định danh như thế nào. Nó nên được thêm vào kho vũ khí các công cụ tinh chỉnh của bạn.

Kiểu dữ liệu SIMPLE_INTEGER

Oracle 11g Database giới thiệu SIMPLE_INTEGER. Nó là một kiểu con của BINARY_INTEGER và nó có cùng một dãy giá trị. Không giống như BINARY_INTEGER, SIMPLE_INTEGER loại trừ giá trị rỗng (null) và sự tràn (overflow) bị cắt bớt.

Bạn nên sử dụng kiểu SIMPLE_INTEGER nếu bạn chọn biên dịch riêng mã bởi vì nó cải thiện hiệu suất đáng kể trong mã biên dịch.

Phần này đã xem lại các tính năng mới của Oracle 11g Database hoặc hướng bạn đến những chương khác của sách để bạn tìm hiểu thêm.

Các lệnh gọi dãy trực tiếp trong các câu lệnh SQL

Sau cùng, Oracle 11g cho phép gọi một dãy với nextval hoặc currval bên trong các lệnh SQL, nghĩa là bạn có thể dùng như sau:

```
SELECT some_sequence.nextval
       INTO some_local_variable
       FROM dual;
```


Tóm tắt

Chương này đã trình bày lịch sử, tiện ích, các điểm cơ bản về viết mã và những tính năng gần đây được thêm vào ngôn ngữ lập trình PL/SQL. Chương cũng giải thích sự quan trọng của PL/SQL và cách nó có thể tận dụng sự đầu tư của bạn vào cơ sở dữ liệu Oracle 11g như thế nào. Bạn sẽ thấy rằng một sự kết hợp SQL và PL/SQL có thể đơn giản hoá các dự án phát triển ứng dụng bên ngoài trong những ngôn ngữ như Java và PHP.

CHƯƠNG 2

CÁC ĐIỂM CƠ BẢN VỀ PL/SQL

Nơi khởi đầu chung là tóm tắt các thành phần ngôn ngữ. Chương này khảo sát những tính năng PL/SQL. Các chương tiếp theo triển khai các chi tiết nhằm giải thích tại sao ngôn ngữ PL/SQL là một công cụ mạnh với nhiều tùy chọn.

Nhằm giới thiệu những điểm cơ bản về PL/SQL, chương này giới thiệu và thảo luận ngắn gọn

- Cấu trúc khối Oracle PL/SQL
- Các biến, phép gán và toán tử
- Các cấu trúc điều khiển
- Các cấu trúc có điều kiện
- Các cấu trúc lặp lại
- Các hàm lưu trữ, thủ tục và gói (package)
- Phạm vi giao tác
- Các trigger cơ sở dữ liệu

PL/SQL là một ngôn ngữ lập trình không nhạy kiểu chữ như SQL. Tuy ngôn ngữ không nhạy kiểu chữ, nhưng có nhiều qui ước áp dụng cho cách viết mã. Hầu hết chọn kết hợp chữ hoa, chữ thường, kiểu chữ tiêu đề hoặc kiểu chữ hỗn hợp. Trong số những ý kiến này không có phương pháp chuẩn nào để tuân theo.

Sử dụng chuẩn PL/SQL cho sách này

Mã PL/SQL trong sách này sử dụng chữ hoả cho các từ lệnh và chữ thường cho các biến, tên cột và lệnh gọi chương trình lưu trữ.

Cấu trúc khối Oracle PL/SQL

PL/SQL đã được phát triển bằng cách mô phỏng những khái niệm về lập trình cấu trúc, định kiểu dữ liệu tĩnh, tính module và quản lý ngoại lệ. Nó mở rộng ngôn ngữ lập trình ADA. ADA mở rộng ngôn ngữ lập trình Pascal bao gồm các toán tử gán và so sánh và các dấu phân cách chuỗi dấu ngoặc đơn.

PL/SQL hỗ trợ hai loại chương trình: một là chương trình khối nặc danh (anonymous-block) và chương trình kia là chương trình khối được đặt tên. Cả hai loại chương trình có các phần hoặc khối xử lý khai báo (declaration), thực thi (execution), và ngoại lệ (exception). Các khối nặc danh hỗ trợ việc tạo script hàng loạt và các khối được đặt tên phân phối các đơn vị lập trình lưu trữ.

Nguyên mẫu cơ bản cho chương trình PL/SQL khối nặc danh là

```
[DECLARE]
    declaration_statements
BEGIN
    execution_statements
[EXCEPTION]
    exception_handling_statements
END;
/
```

Như được trình bày trong nguyên mẫu, PL/SQL đòi hỏi chỉ phần thực thi cho một chương trình khối nặc danh. Phần thực thi bắt đầu với một câu lệnh BEGIN và dừng tại đầu khối EXCEPTION tùy chọn hoặc câu lệnh END của chương trình. Một dấu chấm phẩy kết thúc khối PL/SQL nặc danh và dấu gạch chéo tiến (/) thực thi khối.

Các phần khai báo có thể chứa các định nghĩa biến và các phần khai báo, các định nghĩa kiểu PL/SQL do người dùng định nghĩa, các định nghĩa cursor, định nghĩa cursor tham chiếu và các định nghĩa hàm cục bộ hoặc thủ tục. Các phần thực thi có thể chứa các phép gán biến, phần khởi tạo đối tượng, cấu trúc có điều kiện, cấu trúc lặp lại, các khối PL/SQL xếp lồng hoặc các lệnh gọi đến các khối PL/SQL được đặt tên cục bộ hoặc lưu trữ. Các phần ngoại lệ chứa các cụm từ xử lý lỗi mà có thể sử dụng tất cả các mục giống như phần thực thi.

Khối PL/SQL đơn giản nhất không làm gì cả. Bạn phải có tối thiểu một câu lệnh bên trong bất kỳ khối thực thi ngay cả nếu đó là một câu lệnh NULL. Dấu gạch chéo (/) thực thi một khối PL/SQL rỗng danh. Dòng mã sau đây minh họa chương trình khối rỗng danh cơ bản nhất.

```
BEGIN
    NULL;
END;
/
```

Bạn phải bật biến SQL*Plus SERVEROUTPUT để in nội dung sang console. hello_world.sql in một thông báo sang console:

```
-- This is found in hello_world.sql on the publisher's web site.
SET SERVEROUTPUT ON SIZE 1000000
BEGIN
    dbms_output.put_line('Hello World.');
```

END;

/

Biến môi trường SQL*Plus SERVEROUTPUT mở một bộ đệm ra (output buffer) và hàm DBMS_OUTPUT.PUT_LINE() in một dòng kết quả. Tất cả phần khai báo, câu lệnh và khối được kết thúc bằng một dấu chấm phẩy.

Ghi chú

Mọi khối PL/SQL phải chứa một thứ gì đó, tối thiểu là một câu lệnh NULL, nếu không nó sẽ không biên dịch thời gian chạy còn được gọi là phân tích cú pháp (parsing).

SQL*Plus hỗ trợ việc sử dụng các biến thay thế trong console tương tác bắt đầu bằng một dấu ampersand (&). Các biến thay thế là chuỗi hoặc số có chiều dài khả biến. Bạn không nên gán các giá trị động trong khối khai báo như các biến thay thế.

Chương trình sau đây định nghĩa một biến và gán cho nó một giá trị:

```
— This is found in substitution.sql on the publisher's web site.
DECLARE
    my_var VARCHAR2(30);
BEGIN
    my_var := '&input';
    dbms_output.put_line('Hello ' || my_var );
```

```
END;
```

```
/
```

Toán tử gán trong PL/SQL là một dấu hai chấm cộng với một dấu bằng (`:=`). Các trực kiện PL/SQL được phân cách bằng các dấu ngoặc đơn. Các trực kiện ngày tháng, số và chuỗi được đề cập trong chương 3.

Bạn chạy các khối nặc danh bằng cách gọi chúng từ Oracle SQL*Plus. Ký hiệu `@` trong Oracle SQL*Plus tải và thực thi một file script. Phần mở rộng file mặc định là `.sql`, nhưng bạn có thể ghi đè nó bằng một phần mở rộng khác. Điều này có nghĩa là bạn có thể gọi một tên file không có phần mở rộng `.sql` của nó.

Dòng mã sau đây minh họa việc gọi file `substitution.sql`:

```
SQL> @substitution.sql
Enter value for input: Henry Wadsworth Longfellow
old   3:  my_var VARCHAR2(30) := '&input';
new   3:  my_var VARCHAR2(30) := 'Henry Wadsworth Longfellow';
Hello Henry Wadsworth Longfellow
PL/SQL procedure successfully completed.
```

Dòng bắt đầu với `old` chỉ định nơi chương trình chấp nhận một sự thay thế và `new` chỉ định sự thay thế thời gian thực. Gán một trực kiện chuỗi quá lớn cho biến sẽ kích khởi một ngoại lệ. Các khối ngoại lệ quản lý các lỗi được đưa ra. Bộ xử lý ngoại lệ (exception handler) chung quản lý bất kỳ lỗi nào xảy ra.

Bạn sử dụng một khối `WHEN` để đón bắt mọi ngoại lệ được đưa ra. Với bộ xử lý lỗi chung `_OTHERS`.

••••• Thủ thuật

Bạn có thể bỏ qua việc hiện lên phần thay thế bằng cách tắt SQL*Plus `VERIFY`.

Chương trình `exception.sql` sau đây minh họa cách một khối ngoại lệ quản lý một lỗi như thế nào khi chuỗi quá dài cho biến:

```
-- This is found in exception.sql on the publisher's web site.
DECLARE
    my_var VARCHAR2(10);
BEGIN
    my_var := '&input';
    dbms_output.put_line('Hello '||my_var );
EXCEPTION
    WHEN others THEN
```

```

dbms_output.put_line(SQLERRM);
END;
/

```

Khối nặc danh đã thay đổi định nghĩa của chuỗi từ 30 ký tự thành 10 ký tự. Tên của nhà thơ (poet) bây giờ quá dài không thể nằm vừa trong biến đích. Việc gán biến sẽ đưa ra một ngoại lệ. Kết quả trên console thể hiện ngoại lệ được xử lý và được đưa ra.

```

SQL> @exception.sql
Enter value for input: Henry Wadsworth Longfellow
old 4: my_var := '&input';
new 4: my_var := 'Henry Wadsworth Longfellow';
ORA-06502: PL/SQL: numeric or value error: character string buffer too small
PL/SQL procedure successfully completed.

```

Bạn cũng có thể có: (a) các chương trình khối nặc danh xếp lồng trong phần thực thi của một khối nặc danh; (b) các chương trình khối được đặt tên trong phần khai báo mà lần lượt có thể chứa cùng một loại chương trình xếp lồng; và (c) các lệnh gọi đến các chương trình khối được định danh lưu trữ.

Khối lập trình tận cùng bên ngoài điều khiển toàn bộ dòng chảy chương trình trong khi các khối xếp lồng chương trình điều khiển dòng chảy lập trình thứ cấp của chúng. Mỗi đơn vị lập trình khối nặc danh hoặc khối được đặt tên có thể chứa một phần ngoại lệ. Khi một phương thức xử lý ngoại lệ cục bộ không quản lý một ngoại lệ, nó ném ngoại lệ cho một khối chứa cho đến khi nó tiến đến môi trường SQL*Plus.

Quản lý ngăn xếp lỗi (error stack) giống nhau cho dù các lỗi được đưa ra từ các khối PL/SQL cục bộ hay các khối được đặt tên. Các lỗi được đưa ra và được đặt trong một hàng đợi vào trước ra sau còn gọi là ngăn xếp (stack).

Bạn đã khai thác cấu trúc cơ bản của các chương trình khối PL/SQL và việc quản lý ngăn xếp lỗi. Cấu trúc khối là kiến thức cơ bản để làm việc trong PL/SQL.

Các biến, phép gán và toán tử

Các kiểu dữ liệu trong PL/SQL bao gồm tất cả dữ liệu SQL và kiểu con. Chương 3 đề cập các kiểu dành riêng cho PL/SQL. PL/SQL cũng hỗ trợ các biến vô hướng và biến tổng hợp. Các biến vô hướng (scalar variables) chứa chỉ một thứ trong khi các biến tổng hợp (composite variables) chứa nhiều thứ. Các chương trình trước đã minh họa cách khai báo và gán giá trị vào các biến vô hướng như thế nào.

Các tên biến bắt đầu với các mẫu tự và có thể chứa các ký tự trong bảng chữ cái, các số thứ tự (0 đến 9), các ký hiệu \$ và #. Các biến chỉ có phạm vi cục bộ. Điều này có nghĩa chúng chỉ có sẵn trong phạm vi của một khối PL/SQL nào đó. Các ngoại lệ trong qui tắc đó là các khối nặc danh xếp lồng. Các khối nặc danh xếp lồng hoạt động bên trong khối định nghĩa. Do đó, chúng có thể truy cập các biến từ khối chứa. Nghĩa là, trừ phi bạn đã khai báo cùng một tên biến với một thứ khác nào đó bên trong khối nặc danh xếp lồng.

Một phần khai báo của biến số không có phép gán tường minh làm cho giá trị ban đầu của nó trở nên rỗng. Nguyên mẫu cho thấy bạn có thể gán một giá trị sau đó trong khối thực thi:

```
DECLARE
    variable_name NUMBER;
BEGIN
    variable_name := 1;
END;
/
```

Phép gán tường minh gán một biến với một giá trị không rỗng. Bạn có thể sử dụng giá trị mặc định hoặc gán một giá trị mới trong khối thực thi. Cả hai được minh họa tiếp theo. Bạn có thể sử dụng một toán tử gán hoặc từ dành riêng DEFAULT thay thế cho nhau để gán các giá trị ban đầu. Dòng mã sau đây minh họa một nguyên mẫu:

```
DECLARE
    variable_name NUMBER [:= | DEFAULT ] 1;
BEGIN
    variable_name := 1;
END;
/
```

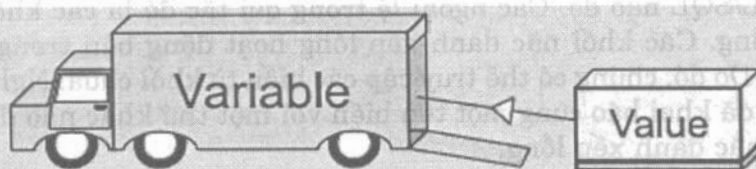
Mô hình và ngôn ngữ gán

Tất cả ngôn ngữ lập trình gán các giá trị vào các biến. Chúng thường gán một giá trị vào biến nằm bên trái.

Nguyên mẫu cho việc gán chung trong bất kỳ ngôn ngữ lập trình là

left_operand assignment_operator right_operand statement_terminator

Nguyên mẫu này gán toán hạng phải vào toán hạng trái như được trình bày ở đây:



Bạn thực thi nó trong PL/SQL như sau:

```
left_operand := right_operand;
```

Toán hạng trái phải luôn là một biến. Toán hạng phải có thể là một giá trị, biến hoặc hàm. Các hàm phải trả về một giá trị khi chúng là các toán hạng phải. Điều này tiện lợi trong PL/SQL bởi vì tất cả hàm trả về các giá trị. Các hàm trong ngữ cảnh này còn được gọi là các biểu thức (expression).

Thủ thuật là chỉ các hàm trả về một kiểu dữ liệu SQL có thể được gọi trong các câu lệnh SQL. Các hàm trả về một kiểu dữ liệu PL/SQL chỉ làm việc bên trong các khối PL/SQL.

Oracle 11g thực thi nhiều thao tác cast ngầm định. Chúng không tuân theo qui tắc lập trình chung: cast tường minh khi không mất độ chính xác. Điều này có nghĩa bạn có thể gán một số phức như 4.67 vào một số nguyên và mất đi phần 0.67 của số. Tương tự có một loạt các hàm để cho bạn cast một cách tường minh khi rủi ro mức độ chính xác cao hơn. Bạn nên chọn cẩn thận khi nào bạn downcast các biến một cách tường minh.

Cũng có một số kiểu dữ liệu dành riêng cho sản phẩm. Chúng hỗ trợ các sản phẩm thành phần khác nhau trong Oracle 11g. Bạn có thể tìm thấy các kiểu dữ liệu này trong Oracle Database PL/SQL Packages and Type Reference.

Toán tử gán không phải là toán tử đơn chiếc trong ngôn ngữ lập trình PL/SQL. Chương 3 đề cập đến tất cả toán tử so sánh, ghép, logic và toán học. Tóm lại, bạn sử dụng

- Ký hiệu bằng (=) để kiểm tra các giá trị tương hợp.
- Lớn hơn hoặc nhỏ hơn chuẩn có hoặc không có một thành phần bằng (>, >=, <, hoặc <=) dưới dạng các toán tử so sánh để kiểm tra tìm các bất đẳng thức.

Các toán tử so sánh phủ định (<>, !=, ~= hoặc ^=) để kiểm tra tìm các giá trị không tương hợp.

Bạn định nghĩa các câu lệnh CURSOR trong phần khai báo. Các câu lệnh CURSOR cho bạn mang dữ liệu từ các bảng (table) và view vào những chương trình PL/SQL. Một câu lệnh CURSOR có thể không có hoặc có nhiều tham số hình thức. Các tham số CURSOR là các biến chuyển theo giá trị (pass-by-value) hoặc các biến chỉ chế độ IN. Chương 4 đề cập đến các câu lệnh CURSOR.

Bây giờ bạn đã xem lại các biến, phép gán và toán tử. Bạn cũng đã được trình bày các kiểu do người dùng định nghĩa dành riêng cho PL/SQL.

Các cấu trúc điều khiển

Các cấu trúc điều khiển làm hai điều. Chúng kiểm tra một điều kiện logic và phân nhánh việc thực thi chương trình hoặc chúng lặp lại trên một điều kiện cho đến khi nó được đáp ứng hoặc được ra lệnh thoát. Phần các cấu trúc có điều kiện đề cập đến các câu lệnh if, elsif, else và case. Phần sau "Các cấu trúc lặp lại" đề cập đến việc lặp lại bằng các cấu trúc for và while.

Các cấu trúc có điều kiện

Các câu lệnh có điều kiện kiểm tra việc một giá trị có đáp ứng điều kiện trước khi tiến hành hay không. Có hai loại cấu trúc có điều kiện trong PL/SQL. Một là câu lệnh IF và còn lại là câu lệnh CASE. Câu lệnh IF có hai loại con if-then-else và if-then-elsif-then-else. elsif không phải là lỗi gõ nhập mà là từ dành riêng chính xác trong PL/SQL. Đây là một sự thừa kế khác từ Pascal và ADA.

Câu lệnh IF

Tất cả câu lệnh IF là các khối trong PL/SQL và kết thúc bằng cụm từ END IF. Câu lệnh CASE cũng là các khối kết thúc bằng cụm từ END CASE. Các dấu chấm phẩy đi theo các cụm từ kết thúc và kết thúc tất cả khối trong PL/SQL. Sau đây là nguyên mẫu cơ bản cho một khối PL/SQL if-then-else:

```
IF [NOT] left_operand1 = right_operand1 [[AND]OR]
   [NOT] left_operand2 = right_operand2 [[AND]OR]
   [NOT] boolean_operand ]] THEN
    NULL;
ELSE
    NULL;
END IF;
```

Nguyên mẫu khối if-then-else ở trên sử dụng một phép so sánh đẳng thức nhưng bạn có thể thay thế bất kỳ toán tử so sánh cho ký hiệu bằng. Bạn cũng có thể lượng giá một hoặc nhiều điều kiện bằng cách sử dụng AND hoặc OR để nối các câu lệnh. Sau đó các kết quả áp dụng vào tổ hợp các biểu thức. Bạn có thể phủ định các kết quả đơn hoặc được kết hợp bằng toán tử NOT.

Các toán tử logic hỗ trợ các thao tác hợp nhất và bao hàm. Một toán tử hợp nhất END nghĩa là cả hai câu lệnh phải lượng giá là true hoặc false. Một toán tử bao hàm OR nghĩa cái này hoặc cái kia phải là true (đúng). Các toán tử bao hàm ngưng xử lý khi một câu lệnh lượng giá là true. Các toán tử hợp nhất kiểm tra nhằm bảo đảm rằng tất cả câu lệnh lượng giá là true.

Các biến BOOLEAN là những phép so sánh của chính chúng. Những toán hạng khác có thể là bất kỳ kiểu dữ liệu hợp lệ vốn làm việc với toán tử so sánh thích hợp nhưng hãy nhớ các biến phải được khởi tạo. Các sự cố xảy ra khi bạn không khởi tạo hoặc xử lý các biến không được khởi tạo trong các câu lệnh.

••••• Thủ thuật

Bạn có thể kiểm tra xem một giá trị BOOLEAN là true hay không bằng cách sử dụng một toán tử so sánh và hằng (như `some_boolean = TRUE`), nhưng nó không phải là cách tốt nhất để sử dụng một biến Boolean trong một phép toán so sánh.

Ví dụ, khi bạn sử dụng một câu lệnh IF để lượng giá một BOOLEAN không được khởi tạo là true, nó thất bại và xử lý khối ELSE. Tuy nhiên, khi bạn sử dụng một câu lệnh IF NOT để lượng giá một BOOLEAN không được khởi tạo là false, nó cũng thất bại, và xử lý khối ELSE. Điều này xảy ra bởi vì một biến BOOLEAN không được khởi tạo không true hoặc false.

Giải pháp cho vấn đề này là sử dụng hàm SQL NVL (). Nó cho bạn thay thế một giá trị cho bất kỳ biến giá trị rỗng (null). Hàm NVL () đòi hỏi hai tham số: tham số thứ nhất là một biến và tham số thứ hai là một trực kiện (literal) mà có thể là một giá trị số, chuỗi, hoặc giá trị hằng. Hai tham số phải chia sẻ cùng một kiểu dữ liệu. Bạn có thể truy cập riêng tất cả hàm SQL chuẩn trong những chương trình PL/SQL. Chương trình sau đây minh họa cách bạn sử dụng NVL () trên một biến BOOLEAN không được khởi tạo:

```
-- This is found in if_then.sql on the publisher's web site.
```

```
DECLARE
```

```
-- Define a Boolean variable.
```

```

my_var BOOLEAN;
BEGIN
  -- Use an NVL function to substitute a value for evaluation.
  IF NOT NVL(my_var,FALSE) THEN
    dbms_output.put_line('This should happen!');
  ELSE
    dbms_output.put_line('This can't happen!');
  END IF;
END;
/

```

Câu lệnh IF NOT trả về false khi biến BOOLEAN không được khởi tạo. Chương trình trước tìm giá trị hàm NVL () để là false, hoặc NOT true và nó in thông báo sau đây:

This should

Câu lệnh if-then-elsif-then-else làm việc như câu lệnh if-then-else, nhưng cho bạn thực hiện nhiều lượng giá có điều trong cùng một câu lệnh IF. Sau đây là nguyên mẫu cơ bản cho một khối PL/SQL if-then-elsif-then-else.

```

IF [NOT] left_operand1 > right_operand2 [AND|OR]
  NULL;
ELSIF [NOT] left_operand1 = right_operand1 [[AND|OR]
  [NOT] left_operand2 = right_operand2 [[AND|OR]
  [NOT] boolean_operand ]] THEN
  NULL;
ELSE
  NULL;
END IF;

```

Câu lệnh CASE

Câu lệnh điều kiện còn lại là câu lệnh CASE. Câu lệnh CASE làm việc như tiến trình if-then-elsif-then-else. Có hai loại câu lệnh CASE: một loại là CASE đơn giản và loại kia là CASE tìm kiếm. Câu lệnh CASE đơn giản lấy một biến làm biểu thức và sau đó lượng giá nó dựa vào danh sách các kết quả vô hướng tương tự. Câu lệnh CASE tìm kiếm lấy một biến BOOLEAN làm biểu thức và sau đó so sánh trạng thái Boolean của các kết quả mệnh đề WHEN dưới dạng một biểu thức.

Sau đây là nguyên mẫu của câu lệnh CASE:

```

CASE [ TRUE | [selector_variable]]
  WHEN [criterion1 | expression1] THEN
    criterion1_statements;
  WHEN [criterion2 | expression2] THEN
    criterion2_statements;
  WHEN [criterion(n+1) | expression(n+1)] THEN
    criterion(n+1)_statements;
  ELSE
    block_statements;
END CASE;

```

Chương trình kế tiếp minh hoạ một câu lệnh CASE tìm kiếm:

```

BEGIN
  CASE TRUE
  WHEN (1 > 3) THEN
    dbms_output.put_line('One is greater than three.');
```

WHEN (3 < 5) THEN

```

    dbms_output.put_line('Three is less than five.');
```

WHEN (1 = 2) THEN

```

    dbms_output.put_line('One equals two.');
```

ELSE

```

    dbms_output.put_line('Nothing worked.');
```

END CASE;

END;

/

••••• Thủ thuật

Bạn có thể bỏ qua TRUE (bởi vì nó là selector mặc định), nhưng không. Việc đặt nó vào sẽ tăng thêm tính rõ ràng.

Chương trình lượng giá các kết quả mệnh đề WHEN dưới dạng các biểu thức, cho thấy 3 nhỏ hơn 5. Sau đó nó in.

Three is less than five

Bạn có thể tìm hiểu thêm về những câu lệnh CASE trong chương 4. Mục nhỏ này đã minh họa các biểu thức có điều kiện có sẵn cho bạn trong PL/SQL. Nó cũng gợi ý một lựa chọn thay thế cho các biến không được khởi tạo.

Các cấu trúc lặp lại

PL/SQL hỗ trợ các vòng lặp FOR, SIMPLE và WHILE. Không có cú pháp cho khối vòng lặp repeat until nhưng bạn vẫn có thể thực thi một khối vòng lặp. Các vòng lặp thường làm việc cùng với các cursor nhưng có thể giải quyết những vấn đề khác như tìm kiếm hoặc quản lý các tập hợp Oracle.

Các vòng lặp FOR

PL/SQL hỗ trợ các vòng lặp FOR số và cursor. Vòng lặp FOR số (numeric) lặp lại qua một dãy được định nghĩa trong khi vòng FOR cursor lặp lại qua các hàng được trả về bởi một cursor câu lệnh SELECT. Các vòng lặp FOR quản lý cách chúng bắt đầu và kết thúc một cách ngầm định như thế nào. Bạn có thể ghi đè cụm từ implicit END LOOP bằng cách sử dụng câu lệnh CONTINUE hoặc EXIT tương minh để lần lượt bỏ qua một phép lặp lại hoặc buộc thoát sớm khỏi vòng lặp.

Các vòng lặp FOR số tiến hành hai hành động ngầm định. Chúng tự động khai báo và quản lý index vòng lặp riêng của chúng và chúng tạo và quản lý sự thoát của chúng ra khỏi vòng lặp. Một vòng lặp FOR số có nguyên mẫu sau đây:

```
FOR i IN starting_number..ending_number LOOP
    statement;
END LOOP;
```

Starting_number và ending_number phải là các số nguyên. Index vòng lặp là biến i và phạm vi index vòng lặp giới hạn chỉ trong vòng lặp FOR. Biến index là một số kiểu dữ liệu PLS_INTEGER. Khi trước đó bạn đã định nghĩa hoặc khai báo một biến i, vòng lặp số sẽ bỏ qua biến được định phạm vi bên ngoài và tạo một biến được định phạm vi cục bộ mới.

Chương trình mẫu sau đây in các giá trị index từ 1 đến 10:

```
BEGIN
    FOR i IN 1..10 LOOP
        dbms_output.put_line('The index value is [||i||]');
    END LOOP;
END;
/
```

Vòng lặp FOR cursor đòi hỏi một CURSOR được định nghĩa cục bộ. Bạn không thể sử dụng một vòng lặp FOR cursor để lặp lại qua một cursor tham chiếu (REF CURSOR) bởi vì các cursor tham chiếu chỉ có thể được truyền ngang bằng cách sử dụng các cấu trúc vòng lặp tương minh như các vòng lặp simple và while. Vòng lặp FOR cursor cũng có thể sử dụng một câu lệnh SELECT thay cho một cursor được định nghĩa cục bộ và có nguyên mẫu sau đây:

```
FOR i IN {cursor_name}[(parameter1,parameter(n+1))] | (sql_statement) LOOP
    statement;
END LOOP;
```

cursor_name có thể có một danh sách tham số tùy chọn được đặt trong các dấu ngoặc đơn. Một cursor_name không có các tham số tùy chọn không đòi hỏi các dấu ngoặc đơn. Bạn sử dụng một cursor tương minh khi gọi một cursor_name và một cursor ngầm định khi bạn cung cấp một câu lệnh SELECT.

Dòng mã sau đây minh họa cách viết một cursor tương minh trong vòng lặp FOR và sử dụng dữ liệu được seed (gieo hạt giống) bởi các script có thể download:

```
DECLARE
    CURSOR c IS SELECT item_title FROM item;
BEGIN
    FOR i IN c LOOP
        dbms_output.put_line('The title is [' || i.item_title || ']');
    END LOOP;
END;
/
```

Dòng mã sau đây minh họa cách bạn viết một cursor ngầm định trong một vòng lặp FOR và sử dụng dữ liệu được seed bởi các script có thể download:

```
BEGIN
    FOR i IN (SELECT item_title FROM item) LOOP
        dbms_output.put_line('The title is [' || i.item_title || ']');
    END LOOP;
END;
/
```

Biến index không phải là một số PLS_INTEGER trong vòng lặp FOR cursor. Nó là một tham chiếu dẫn sang cấu trúc record được trả về bởi cursor. Bạn kết hợp biến index cursor và tên cột bằng một dấu chấm còn

gọi là component selector. Trong trường hợp này, index cursor là thành phần (component). Component selector cho bạn chọn một cột từ hàng được trả về bởi cursor.

statement phải là một câu lệnh **SELECT** hợp lệ, nhưng bạn có thể tham chiếu động các tên biến được định phạm vi cục bộ mà không có bất kỳ cú pháp đặc biệt trong tất cả mệnh đề ngoại trừ mệnh đề **FROM**. Trừ phi bạn ghi đề tiêu chuẩn exit (thoát), vòng lặp **FOR** cursor sẽ chạy qua tất cả hàng được trả về bởi cursor hoặc câu lệnh.

Các vòng lặp simple

Các vòng lặp simple (đơn giản) là những cấu trúc tường minh. Chúng đòi hỏi bạn quản lý cả index vòng lặp và tiêu chuẩn thoát. Các vòng lặp đơn giản thường được sử dụng cùng với các câu lệnh cursor được định nghĩa cục bộ và các cursor tham chiếu (**REF CURSOR**).

Oracle cung cấp 6 thuộc tính cursor nhằm giúp bạn quản lý các hoạt động trong những vòng lặp. Bốn thuộc tính cursor là **%FOUND**, **%NOTFOUND**, **%ISOPEN** và **%ROWCOUNT**. Hai loại kia hỗ trợ các thao tác hàng loạt. Chúng được đề cập trong chương 4. Các vòng đơn giản có nhiều công dụng khác nhau. Sau đây là một nguyên mẫu trong một vòng lặp đơn giản sử dụng cursor tường minh:

```
OPEN cursor_name [(parameter1,parameter(n+1))];
LOOP
    FETCH cursor_name
    INTO row_structure_variable | column_variable1 [,column_variable(n+1)];
    EXIT WHEN cursor_name%NOTFOUND;
    statement;
END LOOP;
CLOSE cursor_name;
```

Nguyên mẫu minh họa rằng bạn **OPEN** một **CURSOR** trước khi bắt đầu vòng lặp đơn giản và sau đó bạn **FETCH** một hàng. Trong khi hàng được trả về bạn xử lý chúng, nhưng khi một **FETCH** không trả về một hàng, bạn thoát vòng lặp. Đặt một câu lệnh **EXIT WHEN** dưới dạng câu lệnh cuối cùng trong vòng lặp khi bạn muốn hành vi repeat until loop thường xử lý các câu lệnh trong một vòng lặp tối thiểu một lần bất kể **CURSOR** có trả về các record hay không.

Dòng mã sau đây mô phỏng vòng lặp **FOR** cursor trên bảng **ITEM**.

```
DECLARE
    title item.item_title%TYPE;
    CURSOR c IS SELECT item_title FROM item;
```

```

BEGIN
    OPEN c;
    LOOP
        FETCH c INTO title;
        EXIT WHEN c%NOTFOUND;
        dbms_output.put_line('The title is [' || title || '']);
    END LOOP;
    CLOSE c;
END;
/

```

Các vòng lặp WHILE

Vòng lặp WHILE khác với các vòng lặp đơn giản bởi vì nó bảo vệ lối vào vòng lặp chứ không phải lối thoát. Nó xác lập việc bảo vệ lối vào dưới dạng một biểu thức trước điều kiện. Vòng lặp chỉ được đưa vào khi điều kiện bảo vệ được đáp ứng. Cú pháp cơ bản là

```

OPEN cursor_name [(parameter1,parameter(n+1))];
WHILE condition LOOP
    FETCH cursor_name
    INTO row_structure_variable | column_variable1 [,column_variable(n+1)];
    EXIT WHEN cursor_name%NOTFOUND;
    statement;
END LOOP;
CLOSE cursor_name;

```

Khi điều kiện kiểm tra để tìm một CURSOR được mở thì điều kiện WHILE là cursor_name%ISOPEN. Có nhiều giá trị điều kiện mà bạn có thể sử dụng trong các vòng lặp WHILE. Mã sau đây minh họa cách bạn có thể sử dụng một biến %ISOPEN cursor làm điều kiện bảo vệ khi đi vào:

```

DECLARE
    title item.item_title%TYPE;
    CURSOR c IS SELECT item_title FROM item;
BEGIN
    OPEN c;
    WHILE c%ISOPEN LOOP
        FETCH c INTO title;
        IF c%NOTFOUND THEN

```



```

        CLOSE c;
    END IF;
    dbms_output.put_line('The title-is [' || title || '']);
END LOOP;
END;
/

```

Câu lệnh WHILE chỉ true cho đến khi câu lệnh IF đóng cursor bên trong vòng lặp. Bạn nên chú ý rằng các chỉ lệnh lặp lại đứng sau câu lệnh IF.

Phần này đã minh họa cách bạn có thể sử dụng các cấu trúc tạo vòng lặp ngầm định và tường minh. Nó cũng giới thiệu về việc quản lý câu lệnh CURSOR trong phần thực thi của các chương trình PL/SQL. Chương 4 đề cập đến các câu lệnh CONTINUE và GOTO.

Các hàm lưu trữ, thủ tục và gói (package)

Các đơn vị lập trình lưu trữ PL/SQL thường là các hàm, thủ tục, gói (package) và trigger. Bạn cũng có thể lưu trữ các kiểu đối tượng nhưng điều đó được thảo luận trong chương 14.

Oracle duy trì một danh sách duy nhất các tên đối tượng lưu trữ cho các table, view, trình tự (sequence), chương trình lưu trữ và kiểu (type). Danh sách này được gọi là một namespace. Các hàm, thủ tục, package và đối tượng nằm trong namespace này. Một namespace khác lưu trữ các trigger.

Các hàm lưu trữ, thủ tục và package cung cấp một cách dễ che giấu các chi tiết thực thi trong một đơn vị chương trình. Chúng cũng cho bạn bao bọc phần thực thi khỏi những cặp mắt tò mò trên tầng server.

Các hàm lưu trữ

Các hàm lưu trữ (stored functions) là những cấu trúc tiện lợi bởi vì bạn có thể gọi trực tiếp chúng từ những câu lệnh SQL hoặc chương trình PL/SQL. Tất cả hàm lưu trữ, trả về một giá trị. Bạn cũng có thể sử dụng chúng làm các toán hạng phải bởi vì chúng trả về một giá trị. Các hàm được định nghĩa trong các khối phần khai báo cục bộ hoặc cơ sở dữ liệu. Bạn thường xuyên thực thi chúng bên trong các gói lưu trữ (stored packages).

Nguyên mẫu trong một hàm lưu trữ là

```

FUNCTION function_name
[( parameter1 [IN][OUT] [NOCOPY] sql_datatype | plsql_datatype
[, parameter2 [IN][OUT] [NOCOPY] sql_datatype | plsql_datatype

```

```
[, parameter(n+1) [IN][OUT] [NOCOPY] sql_datatype | plsql_datatype )]]]
RETURN { sql_data_type | plsql_data_type }
[ AUTHID [ DEFINER | CURRENT_USER ] ]
[ DETERMINISTIC | PARALLEL_ENABLED ]
[ PIPELINED ]
[ RESULT_CACHE [ RELIES ON table_name ] ] IS
declaration_statements
BEGIN
    execution_statements
    RETURN variable;
[EXCEPTION]
    exception_handling_statements
END [function_name];
/
```

Các hàm có thể được sử dụng làm các toán hạng phải trong các phép gán PL/SQL. Bạn cũng có thể gọi trực tiếp chúng từ những câu lệnh SQL miễn là chúng trả về một kiểu dữ liệu SQL. Các thủ tục không thể là các toán hạng phải. Bạn cũng không thể gọi chúng từ những câu lệnh SQL.

Bạn có thể truy vấn một hàm vốn trả về một kiểu dữ liệu SQL bằng cách sử dụng nguyên mẫu sau đây từ DUAL giả table:

```
SELECT some_function[(actual_parameter1, actual_parameter2)]
FROM dual;
```

Bạn không còn bị giới hạn chỉ chuyển các tham số thật sự theo thứ tự vị trí cho các câu lệnh SQL nữa. Điều này có nghĩa bạn có thể sử dụng ký hiệu định danh SQL trong PL/SQL. Chương 6 đề cập đến tính cách làm việc của ký hiệu định danh, ký hiệu vị trí và ký hiệu hỗn hợp.

Sau đây là một nguyên mẫu ký hiệu định danh cho cùng một mẫu truy vấn của một hàm PL/SQL từ DUAL giả table:

```
SELECT some_function[(formal_parameter => actual_parameter2)]
FROM dual;
```

Các lệnh gọi vị trí định danh làm việc tốt nhất khi những giá trị mặc định hiện hữu cho các tham số khác. Không có nhiều mục đích trong việc gọi chỉ một số tham số khi lệnh gọi thất bại. Các tham số hình thức là những tham số tùy chọn. Các lệnh gọi vị trí định danh làm việc tốt nhất với các hàm hoặc những thủ tục có các tham số tùy chọn.

Bạn cũng có thể sử dụng câu lệnh CALL để bắt giữ giá trị được trả về từ một hàm vào một biến liên kết. Nguyên mẫu cho câu lệnh CALL như sau:

```
CALL some_function[(actual_parameter1, actual_parameter2)]
INTO some_session_bind_variable;
```

Sau đây là một trường hợp mẫu đơn giản ghép hai chuỗi thành một chuỗi:

```
-- This is found in join_strings.sql on the publisher's web site.
CREATE OR REPLACE FUNCTION join_strings
( string1 VARCHAR2
, string2 VARCHAR2 ) RETURN VARCHAR2 IS
BEGIN
    RETURN string1 || ' ' || string2 || ' ';
END;
/
```

Bây giờ bạn có thể truy vấn hàm từ SQL:

```
SELECT join_strings('Hello','World') FROM dual;
```

Tương tự bạn có thể định nghĩa một biến liên kết cấp session và sau đó sử dụng câu lệnh CALL để đặt biến vào một biến liên kết cấp session:

```
VARIABLE session_var VARCHAR2(30)

CALL join_strings('Hello','World') INTO :session_var;
```

Câu lệnh CALL sử dụng một mệnh đề INTO khi làm việc với các hàm lưu trữ. Bạn phân phối mệnh đề INTO khi làm việc với các thủ tục lưu trữ.

Chọn biến liên kết từ table giả DUAL như sau:

```
SELECT :session_var FROM dual;
```

bạn sẽ thấy

```
Hello World
```

Các hàm mang đến nhiều sức mạnh cho các nhà phát triển cơ sở dữ liệu. Chúng có thể gọi được trong các câu lệnh SQL và khối PL/SQL.

Các thủ tục

Các thủ tục không thể là các toán hạng phải. Bạn cũng không thể sử dụng chúng trong các câu lệnh SQL. Bạn di chuyển dữ liệu vào và ra các thủ tục lưu trữ PL/SQL thông qua danh sách tham số hình thức của chúng. Như với các hàm lưu trữ, bạn cũng có thể định nghĩa các chương trình khối định danh cục bộ trong phần khai báo của các thủ tục.

Nguyên mẫu cho một thủ tục lưu trữ là:

PROCEDURE *procedure_name*

```

([ parameter1 [IN][OUT] [NOCOPY] sql_datatype | plsql_datatype
[, parameter2 [IN][OUT] [NOCOPY] sql_datatype | plsql_datatype
[, parameter(n+1) [IN][OUT] [NOCOPY] sql_datatype | plsql_datatype ])]
[ AUTHID DEFINER | CURRENT_USER ] IS
declaration_statements
BEGIN
    execution_statements
[EXCEPTION]
exception_handling_statements
END [procedure_name];
/

```

Bạn có thể định nghĩa các thủ tục có hoặc không có các tham số hình thức. Các tham số hình thức trong các thủ tục có thể là các biến chuyển theo giá trị (pass-by-value) hoặc chuyển theo tham chiếu (pass-by-reference) trong những thủ tục lưu trữ. Các biến chuyển theo tham chiếu có cả hai chế độ IN và OUT. Như trong trường hợp của các hàm, khi bạn không cung cấp một chế độ tham số, việc tạo thủ tục giả định bạn muốn chế độ là một pass-by-value. Các thủ tục không thể được sử dụng làm các toán hạng phải trong các phép gán PL/SQL cũng không được gọi trực tiếp từ những câu lệnh SQL. Dòng mã sau đây thực thi một thủ tục lưu trữ sử dụng ngữ nghĩa chuyển theo tham chiếu để đóng chuỗi trong các dấu ngoặc vuông:

-- This is found in *format_string.sql* on the publisher's web site.

```

CREATE OR REPLACE PROCEDURE format_string
( string_in IN OUT VARCHAR2 ) IS
BEGIN
    string_in := '[' || string_in || ']';
END;
/

```

Bạn cũng có thể sử dụng câu lệnh CALL để gọi và chuyển các biến vào và ra một thủ tục. Như ví dụ hàm trước đó, ví dụ này sử dụng câu lệnh CALL và biến liên kết:

```

VARIABLE session_var VARCHAR2(30)
CALL join_strings('Hello','World') INTO :session_var;
CALL format_string(:session_var);

```

Câu lệnh CALL đầu tiên gọi hàm được giới thiệu trước đó và tập hợp lại biến :session var. Bạn nên chú ý rằng câu lệnh CALL thứ hai không sử dụng một mệnh đề INTO khi chuyển một biến vào và ra một thủ tục lưu trữ. Điều này khác với cách nó làm việc với các hàm lưu trữ.

Bạn cũng có thể sử dụng câu lệnh EXECUTE với các thủ tục lưu trữ. Dòng sau đây làm việc chính xác như câu lệnh CALL:

```
EXECUTE format_string (:session_var);
```

Khi bạn chọn biến liên kết từ table giả DUAL

```
SELECT :session var FROM dual;
```

bạn sẽ thấy

```
[Hello World.]
```

Trừ phi bạn đã chạy cả hai mẫu, nghĩa là bạn sẽ thấy các dấu ngoặc kép:

```
[[Hello World.]]
```

Các thủ tục cho bạn khả năng sử dụng các tham số hình thức chuyển theo giá trị hoặc chuyển theo tham chiếu. Như bạn sẽ thấy trong các chương 6 và 16, các thủ tục lưu trữ cho bạn trao đổi các giá trị với những ứng dụng bên ngoài.

Các package

Các package (gói) là xương sống của những chương trình lưu trữ trong Oracle 11g. Chúng hoạt động như các thư viện (library) và chúng gồm các hàm và thủ tục. Không giống như các hàm và thủ tục độc lập, các package cho bạn tạo các hàm quá tải và thủ tục. Chương 9 đề cập đến những tính năng này của các package.

Các package có một thông số kỹ thuật được xuất bản. Thông số kỹ thuật tránh những giới hạn của bộ phân tích cú pháp (parser) bởi vì tất cả hàm và thủ tục được xuất bản. Xuất bản giống như tham chiếu về phía trước cho các hàm và thủ tục cục bộ. Các phần thân package chứa các chi tiết ẩn của các hàm và các thủ tục thay vì chữ ký được định nghĩa của chúng.

Các phần thân package phải phản ánh các chữ ký hàm và thủ tục được cung cấp trong các thông số package. Các phần thân package cũng có thể chứa các kiểu được định nghĩa cục bộ, hàm và thủ tục. Những cấu trúc này chỉ có sẵn bên trong phần thân package. Chúng mô phỏng khái niệm về các biến truy cập riêng (private) trong các ngôn ngữ lập trình hiện đại khác như C++ và Java.

Phạm vi giao tác

Phạm vi giao tác (transaction scope) là một chuỗi thực thi - một tiến trình. Bạn thiết lập một session khi bạn kết nối với cơ sở dữ liệu Oracle 11g. Session cho bạn xác lập các biến môi trường như SERVEROUTPUT cho bạn in từ những chương trình PL/SQL. Những gì bạn làm trong suốt session chỉ nhìn thấy được cho đến khi bạn xác nhận thi hành (commit) công việc. Sau khi chắc chắn thay đổi, những session khác có thể thấy những thay đổi mà bạn đã thực hiện.

Trong một session, bạn có thể chạy một hoặc nhiều chương trình PL/SQL. Chúng thực thi một cách nối tiếp hoặc theo trình tự. Chương trình đầu tiên có thể thay đổi dữ liệu hoặc môi trường trước khi chương trình thứ hai chạy... Điều này đúng bởi vì session là giao tác chính. Tất cả hoạt động có thể phụ thuộc vào tất cả hoạt động trước. Bạn có thể commit công việc, làm cho tất cả thay đổi trở nên vĩnh viễn hoặc phục hồi trở lại (roll back) để loại bỏ công việc, bác bỏ tất cả hoặc một số thay đổi.

Sức mạnh để điều khiển session là bằng ba lệnh. Trước đây chúng được gọi là các lệnh ngôn ngữ điều khiển giao tác (TCL). Một số dữ liệu bây giờ nói về chúng dưới dạng các lệnh ngôn ngữ điều khiển dữ liệu (DCL). Sách này sử dụng DCL để tượng trưng cho ba lệnh này. Vấn đề là cố làm rõ ràng nhóm lệnh này từ Tcl của Berkeley. Các lệnh là

- **Câu lệnh COMMIT.** Commit tất cả thay đổi DML được thực hiện từ đầu session hoặc kể từ câu lệnh ROLLBACK cuối cùng.
- **Câu lệnh SAVEPOINT.** Phân chia hai giai đoạn. Một giai đoạn được định nghĩa bằng các giao tác giữa hai điểm thời gian tương đối. Một SAVEPOINT phân giới hai thời kỳ.
- **Câu lệnh ROLLBACK.** Undo (phục hồi) tất cả thay đổi từ thời điểm này trở đi đến một thời kỳ hoặc SAVEPOINT được đặt tên hoặc bây giờ đến đầu một session SQL*Plus.

Những lệnh này cho bạn điều khiển điều gì xảy ra trong các thường trình session và chương trình. Đầu một session vừa là đầu của một thời kỳ vừa là một câu lệnh SAVEPOINT ngầm định. Tương tự, cuối một session là cuối một thời kỳ và câu lệnh COMMIT ngầm định.

Bạn quản lý phạm vi giao tác như thế nào khác nhau giữa một phạm vi giao tác và nhiều phạm vi giao tác. Bạn tạo nhiều phạm vi giao tác khi một hàm hoặc thủ tục được chỉ định là một đơn vị chương trình lưu trữ nặc danh.

Phạm vi giao tác đơn

Một vấn đề công việc phổ biến bao gồm bảo đảm phạm vi trình tự của hai hoặc nhiều câu lệnh DML. Ý kiến là chúng đều phải thành công hoặc thất bại. Thành công một phần không phải là một tùy chọn. Các lệnh

DCL cho bạn bảo đảm hành vi của các hoạt động trình tự trong một phạm vi giao tác đơn.

Chương trình sau đây sử dụng các lệnh DCL để bảo đảm cả hai câu lệnh INSERT thành công hay thất bại:

-- This is found in transaction_scope.sql on the publisher's web site.

```
BEGIN
    -- Set savepoint.
    SAVEPOINT new_member;
    -- First insert.
    INSERT INTO member VALUES
    ( member_s1.nextval, 1005,'D921-71998','4444-3333-3333-4444', 1006
    , 2, SYSDATE, 2, SYSDATE);
    -- Second insert.
    INSERT INTO contact VALUES
    ( contact_s1.nextval, member_s1.currval + 1, 1003
    ,'Bodwin','Jordan',
    , 2, SYSDATE, 2, SYSDATE);
    -- Print success message and commit records.
    dbms_output.put_line('Both succeeded.');
```

COMMIT;

```
EXCEPTION
    WHEN others THEN
        -- Roll back to savepoint, and raise exception message.
        ROLLBACK TO new_member;
        dbms_output.put_line(SQLERRM);
END;
```

/

Câu lệnh INSERT thứ hai thất bại bởi vì ràng buộc khoá ngoại (foreign key) trên member_id trong table member không được đáp ứng. Sự thất bại kích khởi một ngoại lệ Oracle và dịch chuyển sự điều khiển sang khối ngoại lệ. Điều đầu tiên mà khối ngoại lệ làm là phục hồi trở lại câu lệnh SAVEPOINT ban đầu được xác lập bởi chương trình khối nặc danh.

Nhiều phạm vi giao tác

Một số vấn đề công việc đòi hỏi các chương trình làm việc độc lập. Các chương trình độc lập chạy trong các phạm vi giao tác riêng biệt. Khi

bạn gọi một đơn vị chương trình nặc danh, nó chạy trong một phạm vi giao tác khác.

Bạn có thể sử dụng các chương trình nặc danh với chỉ lệnh tiền biên dịch AUTONOMOUS_TRANSACTION. Một chỉ lệnh tiền biên dịch là một PRAGMA và xác lập một hành vi riêng biệt như phạm vi giao tác độc lập. Chỉ các loại chương trình sau đây có thể được chỉ định là các thường trình nặc danh:

- Các khối nặc danh cấp cao nhất (không xếp lồng)
- Các thường trình con gói độc lập cục bộ - các hàm và thủ tục
- Các phương thức của kiểu đối tượng SQL
- Các trigger cơ sở dữ liệu

Phạm vi giao tác bắt đầu được gọi là thường trình chính. Nó gọi một thường trình nặc danh mà sau đó sản sinh ra phạm vi giao tác riêng của nó. Một thất bại trong thường trình chính sau khi gọi một chương trình nặc danh chỉ có thể phục hồi trở lại các thay đổi được thực hiện trong phạm vi giao tác chính. Phạm vi giao tác nặc danh có thể thành công hoặc thất bại độc lập với thường trình chính. Tuy nhiên, thường trình chính cũng có thể thất bại khi một ngoại lệ được đưa ra trong một giao tác nặc danh.

Chương 5 trình bày một ví dụ về loại hoạt động song song này. Câu lệnh INSERT thất bại do các hoạt động trong trigger cơ sở dữ liệu nặc danh. Khi sự kiện (event) kích khởi trigger nặc danh, nó ghi nỗ lực sang một bảng lỗi (error table), commit việc ghi và sau đó đưa ra một ngoại lệ. Ngoại lệ trigger khiến cho câu lệnh INSERT gốc thất bại.

Nhiều chương trình phạm vi giao tác phức tạp. Bạn nên bảo đảm những lợi ích lớn hơn rủi ro khi sử dụng nhiều giải pháp phạm vi giao tác.

Các trigger cơ sở dữ liệu

Các trigger cơ sở dữ liệu là những chương trình lưu trữ chuyên dụng được kích khởi bởi các event (sự kiện) trong cơ sở dữ liệu. Chúng chạy giữa khi bạn thực thi một lệnh và khi bạn thực hiện hành động quản lý cơ sở dữ liệu. Bởi vì chúng nằm ở giữa, bạn không thể sử dụng SQL Data Control Language trong các trigger: SAVEPOINT, ROLLBACK, hoặc COMMIT. Bạn có thể định nghĩa năm loại trigger trong họ sản phẩm Oracle Database 11g:

- *Các trigger Data Definition Language (DDL)*. Những trigger này kích khởi khi bạn tạo, thay đổi, đổi tên hoặc loại bỏ các đối tượng trong một schema cơ sở dữ liệu. Chúng hữu dụng để giám sát các thói quen lập trình kém chẳng hạn như khi các chương trình tạo (create) và loại bỏ (drop) các bảng tạm thời thay vì sử dụng tập hợp Oracle

trong bộ nhớ. Các bảng tạm thời có thể phân đoạn không gian đĩa và theo thời gian giảm đi hiệu suất cơ sở dữ liệu.

- *Các trigger Data Manipulation Language (DML) hoặc các trigger cấp hàng.* Những trigger này kích khởi khi bạn chèn (insert), cập nhật (update), hoặc xoá (delete) dữ liệu ra khỏi một bảng. Bạn có thể sử dụng các loại trigger này để kiểm toán (audit), kiểm tra, lưu và thay thế các giá trị trước khi chúng được thay đổi. Việc đánh số tự động các khoá chính (primary) giả số thường được thực hiện bằng cách sử dụng một trigger DML.
- *Các trigger phức hợp.* Những trigger này hành động như các trigger câu lệnh và trigger cấp hàng khi bạn chèn, cập nhật hoặc xoá dữ liệu ra khỏi một bảng. Những trigger này cho bạn thu thập thông tin tại bốn thời điểm: (a) trước câu lệnh kích khởi; (b) trước mỗi thay đổi hàng từ câu lệnh kích khởi; (c) sau mỗi thay đổi hàng từ câu lệnh kích khởi; (d) sau câu lệnh kích khởi. Bạn có thể sử dụng những loại trigger này để kiểm toán, kiểm tra, lưu và thay thế các giá trị trước khi chúng được thay đổi khi bạn cần thực hiện hành động tại cấp câu lệnh và cấp event hàng.
- *Các trigger Instead of.* Những trigger này cho phép bạn ngưng thi hành một câu lệnh DML và tái hướng dẫn câu lệnh DML. Các trigger INSTEAD OF thường được sử dụng để quản lý cách bạn ghi sang các view vốn vô hiệu hoá việc ghi trực tiếp bởi vì chúng không phải là các view có thể cập nhật. Các trigger INSTEAD OF áp dụng các qui tắc kinh doanh và trực tiếp chèn, cập nhật hoặc xoá các hàng trong các bảng thích hợp liên quan đến các view có thể cập nhật này.
- *Các trigger sự kiện hệ thống hoặc cơ sở dữ liệu.* Những trigger này kích khởi khi một hoạt động hệ thống xảy ra trong cơ sở dữ liệu như các trigger sự kiện logon (đăng nhập) và logoff (đăng xuất) được sử dụng trong chương 13. Những trigger này cho phép theo dõi các sự kiện hệ thống và ánh xạ chúng sang các user.

Tất cả năm loại trigger trên sẽ được đề cập trong chương 10.

Tóm tắt

Chương này đã xem lại những điểm cơ bản của Procedural Language / Structured Query Language (PL/SQL) và giải thích cách bắt đầu với những kỹ năng PL/SQL.

CHƯƠNG 3

CÁC ĐIỂM CƠ BẢN VỀ NGÔN NGỮ

Chương này dựa vào phần thảo luận cấu trúc PL/SQL trong chương 1. Chương giải thích các khối tạo của ngôn ngữ và cách định nghĩa và khai báo các biến, mô tả cách gán các giá trị và các biến và minh hoạ các khái niệm kiểu dữ liệu. Chương được chia thành các phần sau đây:

- Các đơn vị ký tự và đơn vị từ vựng
- Các cấu trúc khối
- Các kiểu biến
- Các kiểu dữ liệu vô hướng
- Các đối tượng lớn
- Các kiểu dữ liệu tổng hợp
- Các cursor tham chiếu hệ thống
- Phạm vi biến

Các đơn vị ký tự và đơn vị từ vựng

Các đơn vị từ vựng là những khối tạo cơ bản trong các ngôn ngữ lập trình. Chúng xây dựng các chương trình PL/SQL. Bạn phát triển các đơn vị từ vựng bằng cách kết hợp các ký tự và symbol hợp lệ. Các đơn vị từ vựng có thể là các dấu tách (delimiter), định danh (identifier), trực kiện (literal) hoặc chú giải (comment). Các định danh bao gồm các từ dành riêng và từ khoá cũng như các tên thường trình con và tên biến.

Các dấu tách

Các dấu tách từ vựng là các symbol hoặc tập hợp symbol. Chúng có thể có chức năng như là các dấu tách hoặc cung cấp những chức năng khác trong các ngôn ngữ lập trình. Những chức năng khác của các dấu tách từ vựng là điều khiển các phép gán, kết hợp, ghép, so sánh, toán học và câu lệnh.

Ví dụ phổ biến nhất về một dấu tách là dấu tách chuỗi ký tự. Trong PL/SQL, bạn tách các trục kiện chuỗi bằng cách sử dụng một tập hợp dấu phết (apostrophe) hoặc các dấu trích dẫn đơn. Bảng 3.1 trình bày đầy đủ tập hợp dấu tách và đưa ra một số ví dụ về cách sử dụng dấu tách trong ngôn ngữ. Các ví dụ bao gồm những kỹ thuật và khái niệm tạo mã được giải thích chi tiết hơn trong phần sau của sách này.

Bảng 3.1 Các dấu tách PL/SQL

Symbol	Loại	Mô tả
:=	Phép gán (Assignment)	<p>Toán tử gán là một dấu hai chấm theo sau ngay là một dấu bằng. Nó là toán tử gán duy nhất trong ngôn ngữ. Bạn gán một toán hạng phải vào một toán hạng trái như</p> <p>a :=b + c;</p> <p>Điều này cộng các số trong các biến b và c và sau đó gán kết quả vào biến a. Phép cộng xảy ra trước phép gán do thứ tự ưu tiên của toán tử, được đề cập sau trong chương này.</p>
:	Kết hợp (Association)	<p>Chỉ báo biến chủ đứng trước một tên định hợp lệ và ấn định định danh đó là biến cấp session. Các biến cấp session còn được gọi là các biến liên kết (bind variables). Bạn sử dụng SQL*Plus để định nghĩa một biến cấp session. Chỉ các kiểu dữ liệu CHAR, CLOB, NCHAR, NCLOB, NUMBER, NVARCHAR2, REF_CURSOR và VARCHAR2 có sẵn cho các biến session. Bạn định nghĩa một biến session bằng cách sử dụng một nguyên mẫu như:</p> <p>VARIABLE variable_name datatype_name</p>

Điều này thực thi nguyên mẫu bằng cách tạo một chuỗi chiều dài khả biến cấp session: `SQL > VARIABLE my_string VARCHAR2 (30)`

Sau đó bạn gán một giá trị bằng cách sử dụng một chương trình PL/SQL khối nặc danh như:

```
BEGIN
:my_string := 'A string literal.';
END;
/
```

Sau đó bạn có thể truy vấn kết quả từ giả table:

```
SELECT :my_string FROM dual;
```

Hoặc, bạn có thể tái sử dụng biến trong một chương trình khối PL/SQL khác bởi vì biến đạt được một phạm vi cấp session. Một chương trình khối nặc danh tiếp theo trong một script sau đó có thể in giá trị trong biến session:

```
BEGIN
dbms_output.put_line(:my_string);
END;
/
```

Đây là một cách linh hoạt để trao đổi các biến giữa nhiều câu lệnh và khối PL/SQL trong một file script. Bạn cũng sử dụng chỉ báo biến chủ làm một placeholder trong các câu lệnh SQL động. Chương 11 giải thích đầy đủ chi tiết về cách sử dụng các placeholder.

& **Kết hợp**

Chỉ báo thay thế cho bạn chuyển các tham số thật sự vào các chương trình PL/SQL khối nặc danh. Bạn không bao giờ gán các biến thay thế bên trong các khối khai báo, bởi vì các lỗi gán không đưa ra một lỗi mà bạn có thể đón bắt trong khối ngoại lệ. Bạn nên thực hiện các phép gán biến

thay thế trong khối thực thi. Dòng sau đây minh họa việc gán một biến thay thế chuỗi vào một biến cục bộ trong khối thực thi:

```
a := '&string_in';
```

% Kết hợp

Chỉ báo thuộc tính cho bạn liên kết một thuộc tính cột, hàng hoặc cursor của catalog cơ sở dữ liệu. Bạn neo (anchor) một kiểu dữ liệu biến khi bạn liên kết một biến với một đối tượng catalog, như một table hoặc cột. Phần sau "Các kiểu biến" trong chương kiểm tra cách neo các biến sang các mục catalog cơ sở dữ liệu bằng toán tử này. Chương 4 trình bày cách tận dụng các thuộc tính cursor. Chương 9 đề cập cách sử dụng các thuộc tính %TYPE và %RONTYPE.

= > Kết hợp

Toán tử kết hợp là một sự kết hợp của một dấu bằng và một dấu lớn hơn. Nó được sử dụng trong hàm ký hiệu tên và các lệnh gọi thủ tục. Chương 6 đề cập cách sử dụng toán tử kết hợp.

. Kết hợp

Component selector là một dấu chấm và nó gắn kết các tham chiếu lại với nhau, ví dụ một schema và một table, một package và một hàm hoặc một đối tượng và một phương thức thành viên (member method). Các component selector có thể được sử dụng để liên kết các cursor và thuộc tính cursor (cột). Sau đây là một số ví dụ nguyên mẫu:

```
schema_name.table_name
```

```
package_name.function_name
```

```
object_name.member_method_name
```

```
cursor_name.cursor_attribute
```

```
bject_name.nested_object_name.object_attribute
```

		Những ví dụ này được tham chiếu trong các chương tiếp theo qua suốt sách này.
@	Kết hợp	Chỉ báo truy cập từ xa cho bạn truy cập một cơ sở dữ liệu từ xa thông qua các liên kết cơ sở dữ liệu.
	Ghép	Toán tử ghép được hình thành bằng cách kết hợp hai đường thẳng đứng vuông góc. Bạn sử dụng nó để gắn kết các chuỗi lại với nhau như được minh họa: a := 'Glued' ' ' 'together. ';
=	So sánh	Dấu bằng là toán tử so sánh. Nó kiểm tra để tìm sự bằng nhau của giá trị và thực hiện việc chuyển đổi kiểu ở nơi có thể một cách ngầm định. (Một biểu đồ thể hiện các chuyển đổi ngầm định được trình bày trong phần sau "Các kiểu biến" của chương này). Không có toán tử so sánh định danh bởi vì PL/SQL là một ngôn ngữ được định kiểu mạnh. Các phép toán so sánh PL/SQL tương đương các phép so sánh định danh bởi vì bạn chỉ có thể so sánh các giá trị thường trình kiểu tương tự.
-	So sánh	Ký hiệu toán tử phải định là một dấu trừ, nó đổi một số từ giá trị dương sang giá trị âm và ngược lại.
<>	So sánh	Có ba toán tử so sánh không bằng, tất cả cùng thực hiện những hành vi như nhau: Bạn sử dụng toán tử nào là tùy thuộc vào nhu cầu của công ty bạn.
!=		
>=	So sánh	Toán tử lớn hơn là một toán tử so sánh không bằng. Nó so sánh toán hạng trái có lớn hơn toán phải không.

<	So sánh	Toán tử nhỏ hơn là một toán tử so sánh không bằng. Nó so sánh toán hạng trái có nhỏ hơn toán hạng phải không.
>=	So sánh	Đây cũng là một toán tử so sánh không bằng. Nó so sánh toán hạng trái lớn hơn hay bằng toán hạng phải.
<=	So sánh	Đây cũng là một toán tử so sánh không bằng. Nó so sánh toán hạng trái nhỏ hơn hay bằng toán hạng phải.
,	Dấu tách (delimiter)	Dấu tách chuỗi ký tự là một dấu ngoặc đơn. Nó cho bạn định nghĩa một giá trị trực kiện chuỗi. Bạn có thể gán một trực kiện chuỗi vào một biến bằng a := 'A string literal.'
(Dấu tách	Điều này tạo một trực kiện chuỗi tập hợp ký tự giữa các dấu tách chuỗi ký tự và gán nó vào biến a. Dấu tách biểu thức hoặc danh sách mở là một dấu ngoặc đơn mở. Bạn có thể đặt một danh sách các trực kiện số hoặc chuỗi tách nhau bằng dấu phẩy hoặc các định danh bên trong một tập hợp dấu ngoặc đơn. Bạn sử dụng các dấu ngoặc đơn để bao bọc các tham số hình thức và tham số thật sự sang các thường trình con hoặc để tạo các danh sách cho các phép lượng giá so sánh. Bạn cũng có thể ghi đè thứ tự ưu tiên bằng cách đặt các phép toán trong các dấu ngoặc đơn. Việc đặt các phép toán trong các dấu ngoặc đơn cho bạn ghi đè thứ tự ưu tiên tự nhiên trong ngôn ngữ.
)	Dấu tách	Dấu tách biểu thức hoặc danh sách đóng là một dấu ngoặc đơn đóng. Xem mục dấu tách biểu thức hoặc danh sách mở để biết thêm thông tin.

'	Dấu tách	Dấu tách mục là một dấu phẩy và tách các mục trong các danh sách.
<<	Dấu tách	Guillemet mở (một từ tiếng Pháp phát âm là gee mey) là dấu tách mở cho các nhãn trong PL/SQL. Các nhãn (label) là bất kỳ định danh hợp lệ trong ngôn ngữ lập trình. Người lập trình Perl và PHP nên biết những nhãn này không làm việc như các thẻ tài liệu HERE. Chương 4 thảo luận các nhãn.
>>	Dấu tách	Guillemet đóng là dấu tách đóng cho các nhãn trong PL/SQL.
--	Dấu tách	Hai dấu gạch găn kề là một toán tử chú giải đơn. Mọi thứ nằm bên phải chú giải đơn được xem là text và không được phân tích cú pháp như là một phần của chương trình PL/SQL. Một ví dụ về chú giải một dòng là: --This is a single line comment
/*	Dấu tách	Đây là dấu tách chú giải nhiều dòng mở. Nó ra lệnh bộ phân tích cú pháp (parser) cho đến dấu tách chú giải nhiều dòng đóng dưới dạng text. Một ví dụ về chú giải nhiều dòng là: /* This is line one. This is line two.*/ Có nhiều gợi ý về cách sử dụng các chú giải (comment). Bạn nên chọn một cách phù hợp với những mục đích của tổ chức và trung thành với nó.
*/	Dấu tách	Đây là dấu tách chú giải nhiều dòng đóng. Nó cho bộ phân tích cú pháp biết rằng chú giải text hoàn tất và mọi thứ sau nó nên được phân tích cú pháp như là một phần của đơn vị chương trình. Xem mục chú giải nhiều dòng mở để biết thêm thông tin.

" Dấu tách

Dấu tách định danh được trích dẫn là một dấu ngoặc kép. Nó cho bạn truy cập các table được tạo bằng cách nhảy kiểu chữ từ catalog cơ sở dữ liệu. Điều này được bắt buộc khi bạn đã tạo các đối tượng catalog cơ sở dữ liệu bằng cách nhảy kiểu chữ. Bạn có thể làm điều này từ Organizer 10g trở lên.

Ví dụ, bạn tạo một table hoặc cột nhảy kiểu chữ bằng cách sử dụng các dấu tách định danh được trích dẫn:

```
CREATE TABLE "Demo"
("Demo_ID" NUMBER
, demo_value VARCHAR2(10));
```

Bạn chèn một hàng bằng cách sử dụng cú pháp tách bằng dấu trích dẫn sau đây:

```
INSERT INTO "Demo1" VALUES
(1,'One Line ONLY.');
```

Như cú pháp SQL, PL/SQL đòi hỏi bạn sử dụng dấu tách định danh được trích dẫn để tìm đối tượng catalog cơ sở dữ liệu như

```
BEGIN
FOR i IN (SELECT "Demo_ID",
demo_value
FROM "Demo") LOOP
dbms_output.put_line(i."Demo_ID");
dbms_output.put_line(i.demo_value);
END LOOP;
END;
/
```

Ngoài định danh trích dẫn trong các câu lệnh SQL nhúng, bạn phải tham chiếu bất kỳ tên cột bằng cách sử dụng cú pháp tách bằng dấu trích dẫn. Điều này được thực hiện trong dòng kết quả đầu tiên nơi index vòng

lập (i) theo sau là component selector (.) và một định danh tách nhau bằng dấu trích dẫn ("Demo ID"). Bạn nên chú ý không bắt buộc các dấu trích dẫn để truy cập cột không nhảy kiểu chữ. Nếu bạn quên đặt một tên (định danh) cột nhảy kiểu chữ trong dấu ngoặc, chương trình trả về một lỗi PLS-00302 nói rằng định danh không được khai báo.

Bạn cũng có thể sử dụng dấu tách định danh được trích dẫn để tạo các định danh có bao gồm các ký hiệu dành riêng như một định danh "X + Y".

+ Toán học

Toán tử cộng cho bạn cộng các toán hạng trái và phải và trả về một kết quả.

/ Phép toán

Toán tử chia cho bạn chia toán hạng trái cho toán hạng phải và trả về một kết quả.

** Toán học

Toán tử số mũ nâng toán hạng trái lên thành lũy thừa được ấn định bằng một toán hạng phải. Toán tử đạt được thứ tự ưu tiên cao nhất cho các toán tử trong ngôn ngữ. Vì điều đó, một số mũ phân số phải được đặt trong các dấu ngoặc đơn (còn được gọi là các dấu tách biểu thức hoặc danh sách) để chỉ định thứ tự phép toán. Nếu không có các dấu ngoặc đơn, toán hạng trái được nâng lên thành lũy thừa của tử số và kết quả được chia cho mẫu số của một số mũ phân. Bạn nâng 3 lên thành lũy thừa 3 và gán kết quả 27 vào một biến a bằng cách sử dụng cú pháp sau đây:

```
a := 3**3;
```

Bạn nâng 8 lên thành lũy thừa phân số 1/3 và gán kết quả 2 vào biến a bằng cách sử dụng cú pháp sau đây:

```
a := B(1/3);
```

		Các dấu ngoặc đơn bảo đảm phép chia xảy ra trước tiên. Các phép toán số mũ được quyền ưu tiên trên các phép toán khác mà không cần đến nhóm ngoặc đơn.
*	Toán học	Toán tử nhân cho bạn nhân toán hạng trái với toán hạng phải và trả về một kết quả.
-	Toán học	Phép toán trừ cho bạn lấy toán hạng trái trừ cho toán hạng phải và trả về một kết quả.
;	Câu lệnh	Dấu kết thúc câu lệnh là một dấu chấm phẩy, bạn phải đóng bất kỳ câu lệnh hoặc đơn vị khối bằng dấu kết thúc câu lệnh.

Các định danh

Các định danh (identifier) là các từ. Chúng có thể là các từ dành riêng, các định danh ẩn định sẵn trích dẫn, biến do người dùng định nghĩa, thường trình con hoặc các kiểu do người dùng định nghĩa.

Các từ dành riêng và từ khoá

Cả các từ dành riêng và từ khoá là các đơn vị từ vựng cung cấp những công cụ cơ bản để sử dụng các chương trình. Ví dụ, bạn sử dụng từ dành riêng NOT làm phép phủ định trong các phép toán so sánh và NULL để tượng trưng cho giá trị rỗng hoặc câu lệnh. Bạn không thể sử dụng những từ này khi định nghĩa các chương trình và kiểu dữ liệu riêng của bạn.

Các định danh định nghĩa sẵn

Oracle 11g cung cấp một gói (package) STANDARD và nó cho phép truy cập toàn bộ đến package này thông qua một cấp phát công cộng (public grant). Gói STANDARD định nghĩa các hàm cài sẵn. Nó cũng chứa các kiểu định nghĩa cho các kiểu dữ liệu chuẩn và các lỗi.

Bạn nên cẩn thận không ghi đè bất kỳ định danh định nghĩa sẵn bằng cách tạo các định danh do người dùng định nghĩa có các tên giống nhau. Điều này xảy ra bất cứ lúc nào bạn định nghĩa một biến vốn sao chép một thành phần từ gói STANDARD tương tự như bạn có thể định nghĩa một biến trong một khối PL/SQL xếp lồng vốn ghi đè tên biến khối chứa.

Các định danh trích dẫn

Oracle 11g cho bạn khả năng sử dụng các dấu tách định danh trích dẫn (quoted identifier) để xây dựng các định danh mà không được phép bằng cách khác do việc tái sử dụng ký hiệu. Các định danh trích dẫn có thể bao gồm bất kỳ các ký tự có thể in kể cả các khoảng trống. Tuy nhiên, bạn không thể nhúng các dấu ngoặc kép bên trong các định danh. Tất cả tối đa của một định danh trích dẫn là 30 ký tự.

Bạn cũng có thể sử dụng các định danh trích dẫn để tận dụng các từ dành riêng và từ khoá. Điều này được phép, nhưng bị ngăn cản quyết liệt bởi Oracle. Ví dụ, chương trình sau đây tạo một định danh trích dẫn "End" vốn là từ dành riêng không nhạy kiểu chữ:

```
DECLARE
    "End" NUMBER := 1;
BEGIN
    dbms_output.put_line('A quoted identifier End ['||'"End"'||']');
END;
/
```

Lần nữa, điều này có thể nhưng bạn nên tránh nó.

Các biến do người dùng định nghĩa, các thường trình con và kiểu dữ liệu

Bạn tạo các định danh khi định nghĩa các thành phần chương trình. Các kiểu dữ liệu do người dùng định nghĩa có thể được định nghĩa trong SQL dưới dạng các kiểu dữ liệu cấp schema (trong các khối PL/SQL). Các định dạng do người dùng định nghĩa phải ít hơn 30 ký tự và bắt đầu với mẫu tự; chúng có thể bao gồm \$, # hoặc _. Chúng không thể chứa dấu chấm câu, các khoảng trống hoặc dấu nối.

Các định danh khối nặc danh chỉ có thể truy cập bên trong một khối hoặc khối xếp lồng. Khi bạn định nghĩa các định danh trong các hàm và thủ tục, tương tự chúng chỉ truy cập kết hợp bên trong các khối được đặt tên. Các thông số package cho bạn định nghĩa các kiểu dữ liệu cấp package có sẵn trong schema. Chúng cũng có sẵn trong các schema khác khi bạn cấp phát các đặc quyền thực thi trên chúng cho schema khác. Bạn tham chiếu chúng bằng cách sử dụng component selector để kết nối package và các tên kiểu dữ liệu. Chương 9 thảo luận các package PL/SQL.

Các trực kiện

Một trực kiện (literal) là một ký tự, chuỗi hoặc giá trị Boolean tường minh. Các giá trị trực kiện không được tượng trưng bởi các định danh.

Các trực kiện chuỗi cũng có thể tượng trưng cho các trực kiện ngày tháng hoặc thời gian.

Các trực kiện ký tự

Các trực kiện ký tự được định nghĩa bằng cách đặt bất kỳ ký tự trong một tập hợp dấu phết (apostrophe). Các giá trị trực kiện nhạy kiểu chữ trong khi ngôn ngữ lập trình thì không nhạy kiểu chữ. Điều này phản ánh hành vi của SQL và dữ liệu được lưu trữ trong cơ sở dữ liệu dưới dạng dữ liệu ký tự hoặc chuỗi (kiểu dữ liệu VARCHAR2 là kiểu thường được sử dụng nhiều nhất).

Bạn gán một trực kiện ký tự vào một biến sử dụng cú pháp sau đây:

```
a := 'a' ;
```

Các trực kiện chuỗi

Các trực kiện chuỗi được định nghĩa như các trực kiện ký tự, sử dụng các dấu ngoặc đơn. Các trực kiện chuỗi có thể chứa bất kỳ số ký tự lên đến giá trị tối đa cho kiểu dữ liệu. Bạn thường sử dụng kiểu dữ liệu VARCHAR2 hoặc một trong các kiểu con (subtype) của nó.

Bạn gán một trực kiện chuỗi vào một biến bằng cách sử dụng cú pháp sau đây:

```
a := 'some string' ;
```

Bạn cũng có thể gán một trực kiện có các dấu ngoặc kép bên trong nó bằng cách sử dụng cú pháp sau đây:

```
a := 'some "quoted" string' ;
```

Các dấu ngoặc kép được xem là các ký tự bình thường khi được nhúng trong các dấu ngoặc đơn.

Các trực kiện số

Các trực kiện số được định nghĩa như các số trong hầu hết các ngôn ngữ lập trình. Việc gán trực kiện số chung bằng cách sử dụng cú pháp sau đây:

```
a :=2525;
```

Bạn có khả năng gán một số lớn với cú pháp số mũ sau đây:

```
n := 2525E8; — This assigns 252,500,000,000 to the variable.
```

Bạn có thể cố gắng gán một số bên ngoài dãy của một kiểu dữ liệu ngoại lệ tràn số hoặc tràn dưới số được đưa ra khi số nằm bên ngoài dãy của kiểu dữ liệu.

Bạn cũng có thể gán một float hoặc một double bằng cách sử dụng cú pháp tương ứng:

`d := 2.0d;` — This assigns a double of 2.

`f := 2.0f;` — This assigns a float of 2

Các trục kiện Boolean

Các trục kiện Boolean có thể là TRUE, FALSE, hoặc NULL. Trạng thái ba giá trị này của các biến Boolean làm chương trình xử lý không đúng một điều khiển `not true` hoặc `not false` bất cứ lúc nào biến là NULL. Chương 4 đề cập cách quản lý các câu lệnh có điều kiện để bảo vệ các kết quả dự tính.

Bạn có thể thực hiện bất kỳ phép gán sau đây vào một biến BOOLEAN được khai báo trước đó:

`b := TRUE;` -- This assigns a true state.

`b := FALSE;` -- This assigns a false state.

`b := NULL;` -- This assigns a null or default state.

••••• Thủ thuật

Bạn nên gán một giá trị ban đầu TRUE hoặc FALSE vào tất cả biến Boolean, nghĩa là luôn định nghĩa trạng thái ban đầu của chúng một cách tường minh. Bạn cũng nên xem xét các cột Boolean dưới dạng không ràng buộc rỗng.

Các trục kiện ngày tháng và thời gian

Các trục kiện ngày tháng có một sự chuyển đổi ngầm định từ trục kiện chuỗi ánh xạ sang mặt nạ định dạng (format mask). Các mặt nạ định dạng mặc định cho các ngày tháng là DD-MON-RR hoặc DD-MON-YYYY. DD tượng trưng cho một ngày hai chữ số, MON tượng trưng cho một tháng ba ký tự, RR tượng trưng cho một năm tương đối hai chữ số và YYYY tượng trưng cho một năm tuyệt đối bốn chữ số. Các năm tương đối được tính bằng cách đếm 50 năm tiến hoặc lùi từ đồng hồ hệ thống hiện hành. Bạn gán một ngày tháng tương đối hoặc tuyệt đối như sau vào các biến kiểu dữ liệu DATE được khai báo trước đó:

`relative_date := '01-JUN-07';` — This assigns 01-JUN-2007.

`absolute_date := '01-JUN-1907';` — This assigns 01-JUN-1907.

Việc gán ngầm định thất bại khi bạn thử những mặt nạ định dạng khác như MON-DD-YYYY. Bạn có thể gán các trục kiện ngày tháng một cách tường minh bằng cách sử dụng hàm `TO_DATE()` hoặc `CAST ()` chỉ

hàm `TO_DATE ()` độc quyền của Oracle cho bạn áp dụng một mặt nạ định dạng ngoại trừ mặt nạ định dạng mặc định. Các loại cú pháp cho hàm `TO_DATE ()` là:

```
date_1 := TO_DATE('01-JUN-07'); — Default format mask.
```

```
date_2 := TO_DATE('JUN-01-07','MON-DD-YY'); — Override format mask.
```

Hàm `CAST ()` có thể sử dụng một trong hai mặt nạ định dạng mặc định được thảo luận trước đó như sau:

```
date_1 := CAST('01-JUN-07' AS DATE); — Relative format mask.
```

```
date_2 := CAST('01-JUN-2007' AS DATE); — Absolute format mask.
```

Bạn có thể sử dụng hàm `TO_CHAR (date_variable 'MON-DD-YYYY')` để xem ngày tháng xác định đầy đủ. Những hành vi này trong PL/SQL phản ánh những hành vi trong Oracle SQL.

Các chú giải

Bạn có thể nhập các chú giải (comments) một dòng hoặc nhiều dòng trong PL/SQL. Bạn sử dụng các dấu gạch để nhập các chú giải một dòng và các dấu tách `/*` và `*/` để nhập một chú giải nhiều dòng. Một chú giải một dòng là

```
This is a single-line comment.
```

Một chú giải nhiều dòng là

```
/* This is a multiple-line comment.
```

```
Style and indentation should follow your company standards. */
```

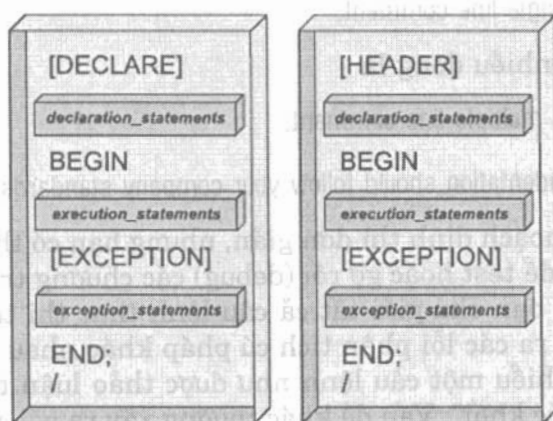
Các chú giải hoạch định thì đơn giản, nhưng bạn có thể gây ra các lỗi khi chú giải mã để test hoặc gỡ rối (debug) các chương trình. Vấn đề lớn nhất xảy ra khi bạn chú giải tất cả câu lệnh thực thi từ một khối mã. Điều này sẽ đưa ra các lỗi phân tích cú pháp khác nhau bởi vì mọi khối mã phải có tối thiểu một câu lệnh như được thảo luận trong phần tiếp theo "Các cấu trúc khối". Vấn đề khác thường xảy ra với các chú giải một dòng nảy sinh do đặt chúng trước một dấu kết thúc câu lệnh (một dấu chấm phẩy) hoặc một từ khoá khối kết thúc. Điều này cũng đưa ra một lỗi phân tích cú pháp khi bạn cố chạy hoặc biên dịch đơn vị chương trình.

Các cấu trúc khối

PL/SQL là một ngôn ngữ lập trình kết khối. Các điều kiện chương trình có thể là các khối được đặt tên hoặc không được đặt tên. Các khối không được đặt tên được gọi là các khối nặc danh và được ghi nhãn như vậy qua suốt sách. Kiểu viết mã PL/SQL khác với các ngôn ngữ lập trình C, C++ và Java. Ví dụ, các dấu ngoặc cong không phân cách các khối trong PL/SQL.

Các chương trình khối nặc danh hiệu quả trong một số tình huống. Bạn thường sử dụng các khối nặc danh khi xây dựng các script để seed dữ liệu hoặc thực hiện các hoạt động xử lý một lần duy nhất. Chúng cũng hiệu quả nếu bạn muốn xếp lồng (nest) hoạt động trong phần thực thi của một khối PL/SQL khác. Cấu trúc khối nặc danh cơ bản phải chứa một phần thực thi. Bạn cũng có thể đặt một phần khai báo tùy chọn và phần ngoại lệ trong các khối nặc danh. Hình 3.1 minh họa các nguyên mẫu khối nặc danh và khối được đặt tên.

Khối khai báo cho bạn khai báo các kiểu dữ liệu, cấu trúc và biến. Khai báo một biến nghĩa là bạn cho nó một tên và kiểu dữ liệu. Bạn cũng có thể định nghĩa một biến bằng cách cho nó một tên, một kiểu dữ liệu và một giá trị. Bạn vừa khai báo vừa gán một giá trị khi định nghĩa một biến. Hình 3.2 minh họa một khái niệm về việc gán một giá trị đơn vào một biến. Các biến vô hướng chứa chỉ mỗi lần một thứ.



Hình 3.1 Cấu trúc khối PL/SQL



Hình 3.2 Gán biến vô hướng

Bạn định nghĩa một biến bằng cách khai báo biến (cung cấp một tên biến và một kiểu dữ liệu) và khởi tạo nó bằng cách gán một giá trị như một trực diện ngày tháng, chuỗi hoặc số. Nguyên mẫu định nghĩa chung là

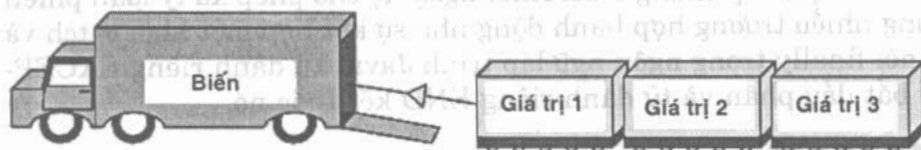
```
variable_name datatype_name := literal_value;
```

Một số loại đối tượng không thể được khai báo là các biến có phạm vi cục bộ và phải được khai báo dưới dạng các kiểu trong catalog cơ sở dữ liệu như được thảo luận trong chương 14. Các cấu trúc (Structure) là các biến phức hợp như các tập hợp (collection), cấu trúc record hoặc các cursor tham chiếu hệ thống. Các cấu trúc cũng có thể là các hàm được đặt tên cục bộ, các thủ tục hoặc cursor.

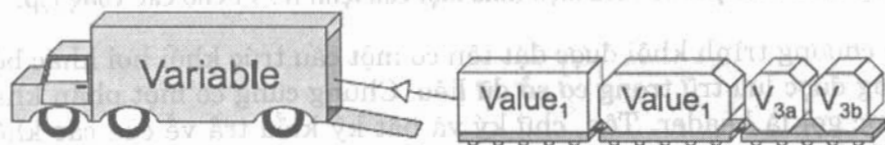
Các cursor hoạt động như các hàm nhỏ. Các cursor có các tên, chữ ký (signature) và một kiểu trả về. Chữ ký là danh sách các tham số hình thức được chấp nhận bởi cursor. Các cột kết quả từ một query hoặc câu lệnh SELECT tạo một cấu trúc cursor dưới dạng kiểu trả về.

Các biến tổng hợp như các biến vô hướng tuân theo các qui tắc định nghĩa tương tự. Sự khác biệt là bạn gán một giá trị vào một biến. Hình 3.3 minh họa ý tưởng tập hợp lại một mảng giá trị bằng cách tải một tập hợp giá trị tương tự. Các phép gán tổng hợp hơi phức tạp hơn nguyên mẫu chung cho các biến vô hướng, được thảo luận trong phần sau "Các kiểu dữ liệu tổng hợp" của chương này.

Một số biến tổng hợp là những cấu trúc chứa những thứ khác nhau như phần tử của một tập address book (sổ địa chỉ). Một cấu trúc giống như một hàng trong một bảng cơ sở dữ liệu. Hình 3.4 minh họa ý tưởng về việc tập hợp lại một cấu trúc - một tập hợp các biến khác nhau.



Hình 3.3 Gán biến tổng hợp: các tập hợp.



Hình 3.4 Gán biến tổng hợp: các cấu trúc

Bạn sử dụng từ dành riêng DECLARE để bắt đầu một khối khai báo và từ dành riêng BEGIN để kết thúc một khối nặc danh. Header của các khối được đặt tên bắt đầu khối khai báo cho các đơn vị lập trình lưu trữ. Như các chương trình khối nặc danh, từ dành riêng BEGIN kết thúc phần khai báo cho các khối được đặt tên. Khối khai báo là nơi bạn khai báo và khởi tạo các biến; nó có thể bao gồm các khối được đặt tên cục bộ.

Khối thực thi cho bạn xử lý dữ liệu. Khối thực thi có thể chứa các phép gán biến, phép so sánh, phép toán điều kiện và phép lặp lại. Khối thực thi cũng là nơi bạn truy cập các cursor và các đơn vị chương trình được đặt tên khác. Các hàm, thủ tục, và một số loại đối tượng là các đơn vị chương trình được đặt tên. Bạn cũng có thể xếp lồng các chương trình khối nặc danh bên trong khối thực thi. BEGIN bắt đầu khối thực thi và EXCEPTION hoặc END kết thúc nó. Dấu chấm phẩy kết thúc khối.

```
BEGIN
    NULL;
END;
/
```

Khối này không thực thi gì cả ngoại trừ cho phép giai đoạn biên dịch hoàn tất mà không gặp lỗi nào cả. Việc biên dịch trong bất kỳ ngôn ngữ đều bao gồm sự phân tích cú pháp. Nếu thiếu một câu lệnh trong khối sẽ đưa ra một lỗi phân tích cú pháp như được đề cập trong chương 5. Bạn nên chú ý dấu gạch chéo tiến (/) điều phối chương trình PL/SQL để thực thi.

Khối xử lý ngoại lệ cho bạn quản lý các ngoại lệ. Bạn vẫn có thể vừa đón bắt vừa quản lý chúng ở đó. Khối ngoại lệ cho phép xử lý luân phiên và trong nhiều trường hợp hành động như sự kết hợp một khối catch và một khối finally trong ngôn ngữ lập trình Java. Từ dành riêng EXCEPTION bắt đầu phần và từ dành riêng END kết thúc nó.

••••• Thủ thuật

Bạn có cùng một qui tắc đòi hỏi tối thiểu một câu lệnh cho bất kỳ khối trong một khối câu lệnh có điều kiện (như một câu lệnh IF) và cho các vòng lặp.

Các chương trình khối được đặt tên có một cấu trúc khối hơi khác bởi vì chúng được lưu trữ trong cơ sở dữ liệu. Chúng cũng có một phần khai báo được gọi là header. Tên, chữ ký và bất kỳ kiểu trả về của các khối PL/SQL được đặt tên được định nghĩa bởi header. Vùng giữa header và các khối thực thi hành động như khối khai báo cho một khối được đặt tên. Qui tắc này cũng tương tự đối với các phần thân kiểu đối tượng được đề cập trong chương 14.

Dòng mã sau đây minh họa một nguyên mẫu hàm khối được đặt tên:

```
FUNCTION function_name
  [( parameter1 [IN][OUT] [NOCOPY] sql_data_type | plsql_data_type
    , parameter2 [IN][OUT] [NOCOPY] sql_data_type | plsql_data_type
    , parameter(n+1) [IN][OUT] [NOCOPY] sql_data_type | plsql_data_type )]
RETURN [ sql_data_type | plsql_data_type ]
  [ AUTHID (DEFINER | CURRENT_USER)]
  [ DETERMINISTIC | PARALLEL_ENABLED ]
  [ PIPELINED ]
  [ RESULT_CACHE [RELIES ON table_name]] IS
  declaration_statements
BEGIN
    execution_statements
  [EXCEPTION]
    exception_handling_statements
END;
/
```

Chương 6 thảo luận các hàm chi phối các qui tắc. Các hàm có thể hành động dưới dạng các thường trình con chuyển theo giá trị hoặc chuyển theo tham chiếu. Những thường trình con chuyển theo giá trị định nghĩa các tham số hình thức sử dụng chỉ chế độ IN. Điều này có nghĩa biến được chuyển vào không thể thay đổi trong quá trình thực thi thường trình con. Các thường trình con chuyển theo tham chiếu định nghĩa các tham số hình thức sử dụng các chế độ chỉ IN và OUT, hoặc OUT.

Oracle tiếp tục chuyển các bản sao của các biến thay vì tham chiếu đến các biến trừ phi bạn chỉ định một gợi ý NOCOPY. Oracle thực thi các hành vi chuyển theo tham chiếu bằng cách này để bảo đảm tính toàn vẹn của các biến của chế độ IN OUT. Mô hình này bảo đảm các biến không thay đổi trừ phi một lệnh gọi chương trình con hoàn tất thành công. Bạn có thể ghi đè hành vi mặc định này bằng cách sử dụng một NOCOPY.

Oracle đề nghị không sử dụng gợi ý NOCOPY bởi vì việc sử dụng nó có thể dẫn đến việc thay đổi một phần các giá trị tham số thực sự. Cuối cùng cơ sở dữ liệu duy trì quyền hành động trên hoặc bỏ qua gợi ý NOCOPY.

Các hàm có thể truy vấn dữ liệu bằng cách sử dụng các câu lệnh SELECT và có thể thực thi các câu lệnh DML, chẳng hạn như INSERT,

UPDATE, hoặc DELETE. Tất cả quy tắc khác áp dụng vào các hàm lưu trữ giống như các quy tắc áp dụng vào các khối nặc danh. Các hàm định nghĩa các tham số hình thức hoặc các kiểu trả về sử dụng các kiểu PL/SQL không thể được gọi từ dòng lệnh SQL. Tuy nhiên, bạn có thể gọi các hàm sử dụng các kiểu dữ liệu SQL từ dòng lệnh SQL.

Giá trị mặc định ALLTHID là DEFINER, cung cấp những gì được gọi là các quyền định nghĩa. Các quyền định nghĩa nghĩa là bất kỳ người nào có các đặc quyền thực thi chương trình lưu trữ có thể chạy nó với các đặc quyền giống như tài khoản người dùng đã định nghĩa nó. Lựa chọn CURRENT_USER cho những người có các đặc quyền thực thi gọi chương trình lưu trữ và chạy nó trên chỉ dữ liệu user/schema của mình. Đây được gọi là các quyền gọi ra và nó mô tả tiến trình gọi một chương trình nguồn chung trên các tài khoản và dữ liệu riêng lẻ.

Bạn nên tránh sử dụng mệnh đề DETERMINISTIC khi các hàm phụ thuộc vào các trạng thái của các biến cấp session. Các mệnh đề DETERMINISTIC thích hợp nhất cho các index dựa vào hàm và các view được cụ thể hoá.

Mệnh đề PARALLEL_ENABLE nên được bật cho các hàm mà bạn dự định gọi từ các câu lệnh SQL vốn sử dụng các tính năng truy vấn song song. Bạn nên xem kỹ mệnh đề này cho những công dụng lưu trữ dữ liệu.

Mệnh đề PIPELINED nâng cao hiệu suất khi các hàm trở về tập hợp (collection) như các table xếp lồng hoặc VARRAY. Bạn cũng sẽ nhận thấy những cải thiện hiệu suất khi trả về các cursor tham chiếu hệ thống bằng cách sử dụng mệnh đề PIPELINED.

Mệnh đề RESULT_CACHE chỉ định một hàm được lưu trữ chỉ một lần trong SGA và có sẵn qua các session. Nó mới trong Oracle 11g Database. Các hàm session chéo chỉ làm việc với các tham số hình thức chế độ IN.

Dòng mã sau đây minh hoạ một nguyên mẫu thủ tục khối được đặt tên:

```
PROCEDURE procedure_name
(parameter1 [IN][OUT] [NOCOPY] sql_datatype | plsql_datatype
, parameter2 [IN][OUT] [NOCOPY] sql_datatype | plsql_datatype
, parameter(n+1) [IN][OUT] [NOCOPY] sql_datatype | plsql_datatype )
[ AUTHID (DEFINER | CURRENT_USER)] IS
declaration_statements
BEGIN
    execution_statements
[EXCEPTION]
```

```
exception_handling_statements
```

```
END;
```

```
/
```

Các kiểu biến

PL/SQL hỗ trợ hai dữ liệu kiểu biến chính: biến vô hướng và biến tổng hợp. Các biến vô hướng (scalar variable) chứa chỉ một thứ như một ký tự, ngày tháng, hoặc số. Không có nhiều sự khác biệt trong các từ nhưng sách này sử dụng các biến tổng hợp (composite variables) để mô tả các mảng (array), cấu trúc (structure), và đối tượng (object). Các biến tổng hợp là các biến được tạo từ những đối tượng nguyên thủy hoặc các kiểu cơ sở trong một ngôn ngữ lập trình. Các biến tổng hợp trong Oracle là các record (cấu trúc), mảng, cursor tham chiếu, và loại đối tượng.

PL/SQL sử dụng tất cả kiểu dữ liệu Oracle SQL. PL/SQL cũng giới thiệu một kiểu dữ liệu Boolean và một số kiểu con được dẫn xuất từ các kiểu dữ liệu SQL. Các kiểu con thừa kế hành vi của một kiểu dữ liệu nhưng cũng thường có các hành vi ràng buộc. Một kiểu con không ràng buộc không thay đổi hành vi của một kiểu cơ sở. Các kiểu không ràng buộc còn được gọi là các bí danh (alias). Bạn cũng có thể gọi bất kỳ kiểu dữ liệu cơ sở là một kiểu bố (supertype) bởi vì nó là một mô hình cho các kiểu con (subtype). Các kiểu con không ràng buộc trao đổi lẫn nhau với các kiểu cơ sở của chúng trong khi chỉ các giá trị đủ điều kiện có thể được gán vào các kiểu con ràng buộc từ các kiểu cơ sở. Bạn có thể mở rộng những kiểu này bằng cách xây dựng các kiểu con riêng của bạn như được thảo luận và được trình bày trong một số phần sau của chương này.

Như các ngôn ngữ lập trình khác, PL/SQL cho bạn vừa định nghĩa các kiểu vừa khai báo các biến. Bạn tạo nhãn cho một kiểu dữ liệu và chỉ định cách quản lý kiểu dữ liệu trong bộ nhớ khi bạn định nghĩa một kiểu. Bạn định nghĩa một biến bằng cách vừa khai báo biến vừa gán cho nó một giá trị. Một tên biến vừa khai báo biến vừa gán cho nó một giá trị. Một tên biến được ánh xạ vào một kiểu dữ liệu đã biết và sau đó được thêm vào namespace của chương trình dưới dạng một định danh khi bạn khai báo một biến. Trong một số ngôn ngữ lập trình không có giá trị nào được gán vào một biến được khai báo. PL/SQL tự động gán cho hầu hết các biến được khai báo một giá trị rỗng. Điều này có nghĩa rằng các biến thường được định nghĩa trong ngôn ngữ.

Bạn khai báo các biến bằng cách gán cho chúng một kiểu hoặc neo (anchor) kiểu của chúng vào một cột catalog cơ sở dữ liệu. Các nguyên mẫu cho cả hai phần khai báo là

```
variable_name variable_type; -- An explicit datatype.
```

```
variable_name column_nameTYPE; -- An anchored datatype.
```

Neo một biến sử dụng TYPE nghĩa là chương trình tự động điều chỉnh khi kiểu dữ liệu cột thay đổi. Điều này đúng khi chỉ kích cỡ thay đổi nhưng không nhất thiết đúng khi kiểu cơ sở thay đổi. Khi sử dụng một số logic, phép gán, và phép so sánh có thể thất bại khi kiểu cơ sở bắt đầu dưới dạng một chuỗi nhưng được biến đổi thành một ngày tháng bởi vì những chuyển đổi ngầm định có thể không đáp ứng tất cả điều kiện logic.

•••••
Thủ thuật

Việc thay đổi kiểu dữ liệu cột không đưa ra một lỗi nhưng vô hiệu hoá bất kỳ thủ tục lưu trữ vốn sử dụng sai kiểu biến mới.

Các chuyển đổi ngầm định được quyết định bởi bộ máy PL/SQL. Không giống như một số ngôn ngữ lập trình, PL/SQL cho phép các chuyển đổi ngầm định vốn dẫn đến việc mất đi tính chính xác (hoặc các chi tiết). Nếu bạn gán một biến BINARY_FLOAT vào một BINARY_INTEGER, bất kỳ chữ số nằm bên phải hàng chữ số thập phân được loại bỏ một cách ngầm định. Các chuyển đổi tương minh đòi hỏi bạn chuyển đổi dữ liệu như gọi hàm cài sẵn TO_CHAR () để hiển thị tem thời gian của một biến DATE. Một danh sách các chuyển đổi ngầm định được minh hoạ trong biểu đồ dưới đây.

TO \ FROM	BINARY_DOUBLE	BINARY_FLOAT	BINARY_INTEGER	BLOB	CHAR	CLOB	DATE	LONG	NCHAR	NCLOB	NUMBER	NVARCHAR2	PLS_INTEGER	RAW	UROWID	VARCHAR2
BINARY_DOUBLE	X	X	X		X			X	X		X	X	X			X
BINARY_FLOAT	X	X	X		X				X		X	X	X			X
BINARY_INTEGER	X	X	X		X						X	X	X			X
BLOB				X											X	
CHAR	X	X	X		X	X	X	X	X		X	X	X	X	X	X
CLOB					X	X			X			X				X
DATE					X		X	X				X				X
LONG					X		X	X				X				X
NCHAR	X	X	X		X	X	X	X	X		X	X	X	X	X	X
NCLOB					X	X	X	X	X			X				X
NUMBER	X	X	X		X			X	X		X	X	X			X
NVARCHAR2	X	X	X		X	X		X	X		X					X
PLS_INTEGER	X	X	X		X			X	X	X	X	X	X			X
RAW				X	X			X	X					X		X
UROWID					X				X	X		X				X
VARCHAR2	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X

Có một giả ngoại lệ cho quy tắc khai báo biến. Các cursor tham chiếu hệ thống được định kiểu yếu không được định nghĩa cho đến thời gian

đang chạy. Một cursor tham chiếu hệ thống được định kiểu yếu lấy một số cursor được gán và chấp nhận cấu trúc record của một hàng được gán vào cursor. Các cấu trúc record chỉ có thể được gán vào các biến tổng hợp. Bạn cũng có thể neo (anchor) một cursor tham chiếu hệ thống được định kiểu mạnh vào một table hoặc view catalog. Điều này làm việc giống nhiều như cách bạn neo các biến vào các cột. Các nguyên mẫu cho việc khai báo các biến tổng hợp là

```
composite_variable_name record_type; -- An explicit datatype.
```

```
composite_variable_name catalog_objectROWTYPE; -- An anchored datatype.
```

Bạn neo một biến tổng hợp bằng cách sử dụng thuộc tính ROWTYPE. Nó cập nhật chương trình để phản ánh bất kỳ thay đổi trong định nghĩa hàng của đối tượng catalog. Loại neo này bảo đảm rằng bạn biết kiểu dữ liệu luôn khớp với đối tượng catalog. Bạn cũng nên neo bất kỳ kiểu dữ liệu biến độc lập khác.

Các kiểu dữ liệu biến có thể được định nghĩa trong SQL hoặc PL/SQL. Bạn có thể sử dụng các kiểu dữ liệu SQL trong các câu lệnh SQL và PL/SQL. Bạn chỉ có thể sử dụng các kiểu dữ liệu PL/SQL bên trong các đơn vị chương trình PL/SQL.

Bộ đệm PL/SQL và xuất sang Console

Như được minh họa trước đó trong hình 1.1, có một bộ đệm ra giữa các bộ máy SQL*Plus và PL/SQL. Bạn có thể mở bộ đệm (buffer) trong SQL*Plus bằng cách bật biến môi trường SERVEROUTPUT như:

```
SQL> SET SERVEROUTPUT ON SIZE 1000000
```

Một khi bạn bật biến môi trường SQL*Plus này, kết quả được tạo ra bởi các thủ tục PUT (), PUT_LINE (), và NEW_LINE () của gói DEMS_OUTPUT sẽ hiển thị trong môi trường SQL*Plus. Có thể bạn nhận được nhiều kết quả hơn bạn mong đợi lần đầu tiên bạn chạy một chương trình sau khi bật biến môi trường. Điều này có thể xảy ra khi bạn chạy một chương trình trong PL/SQL vốn bật bộ đệm từ PL/SQL mà trước tiên không bật biến môi trường.

Bạn bật bộ đệm trong PL/SQL bằng cách sử dụng lệnh sau đây:

```
dbms_output.enable (1000000) ;
```

Hoạt động ghi đầu tiên sang bộ đệm khi bật biến môi trường sẽ xoá sạch tất cả nội dung sang môi trường SQL*Plus, bạn xoá nội dung trước bằng cách tắt bất kỳ bộ đệm mở bằng cách bật nó bằng cách sử dụng hai thủ sau đây theo trình tự:

```
dbms_output.disable;
```

```
dbms_output.enable (1000000) ;
```

Thủ tục DISABLE được đề nghị để bảo đảm bạn không bắt giữ một số kết quả trước không mong muốn khi chạy chương trình. Bạn xuất sang console bằng cách sử dụng thủ tục PUT () hoặc PUT_LINE (). Thủ tục PUT () xuất một chuỗi sang bộ đệm không có một ký tự xuống dòng (line return), trong khi thủ tục PUT_LINE () xuất một chuỗi và ký tự dòng mới (newline) sang bộ đệm. Bạn sử dụng thủ tục NEW_LINE () sang một hoặc nhiều lệnh gọi thủ tục PUT () để ghi một ký tự xuống dòng.

Dòng mã sau đây minh họa cách xuất thông tin từ chương trình PL/SQL sang môi trường SQL*Plus:

```
BEGIN
    dbms_output.put('Line ');
    dbms_output.put('one. ');
    dbms_output.new_line;
    dbms_output.put_line('Line two. ');
END;
/
```

Chương trình khối nặc danh này xuất

Line one.

Line two.

Đây là kỹ thuật mà bạn sẽ sử dụng để xuất sang console để gỡ rối hoặc sang file để báo cáo. Bạn cũng có thể kết hợp lệnh SPOOL SQL*Plus để tách kết quả chuẩn sang console và một file (như lệnh Unix tee). Kỹ thuật này cho bạn tạo các file text để tạo báo cáo (report).

••••• Thủ thuật

Các xác lập biến môi trường SQL*Plus bị mất khi bạn thay đổi các schema. Đừng quên xác lập lại biến SERVEROUTPUT nếu bạn thay đổi các schema bởi vì bộ đệm ra được đóng vào giây phút bạn thay đổi các schema.

Mục nhỏ đầu tiên đề cập đến các kiểu dữ liệu vô hướng, mục nhỏ thứ hai đề cập đến các đối tượng lớn, mục nhỏ thứ ba đề cập đến các kiểu dữ liệu tổng hợp và mục nhỏ thứ tư đề cập đến các kiểu tham chiếu. Các mục được tổ chức cho tham chiếu và dòng chảy. Các kiểu dữ liệu vô hướng là những đối tượng nguyên thủy của ngôn ngữ và do đó là những khối tạo cho các kiểu dữ liệu tổng hợp. Phần tiếp theo đề cập đến những khối tạo nguyên thủy này.

Các kiểu dữ liệu vô hướng

Các đối tượng nguyên thủy (primitive) được nhóm thành các phần theo thứ tự bảng chữ cái. Mỗi phần (section) mô tả kiểu dữ liệu, minh họa cách định nghĩa và/hoặc khai báo kiểu hoặc các biến của kiểu và hướng dẫn cách gán cho nó các giá trị. Hình 3.5 xác định bốn kiểu chính của các biến vô hướng và kiểu con cơ sở thực thi của chúng.

Các kiểu dữ liệu vô hướng sử dụng nguyên mẫu sau đây bên trong khối khai báo của các chương trình:

```
Variable_name datatype [NOT NULL] [:= literal_value] ;
```

Một số kiểu dữ liệu đòi hỏi bạn cung cấp một độ chính xác khi định nghĩa một biến. Độ chính xác định nghĩa kích cỡ tối đa tính bằng byte hoặc ký tự cho một kiểu dữ liệu. Bạn cũng có tỷ lệ cho các kiểu dữ liệu NUMBER. Tỷ lệ định nghĩa số hàng chữ thập phân nằm bên phải dấu thập phân. Những điều này phản ánh các quy ước được tìm thấy trong SQL cho các kiểu dữ liệu này.

Boolean

Kiểu BOOLEAN có ba giá trị có thể có: TRUE, FALSE, và NULL. Trạng thái ba giá trị của các biến Boolean làm cho chương trình có thể xử lý không chính xác một điều kiện not true hoặc not false bất cứ lúc nào biến là NULL. Chương 4 đề cập cách quản lý các câu lệnh có điều kiện để bảo vệ an toàn kết quả dự tính.

Sau đây là nguyên mẫu để khai báo một kiểu dữ liệu BOOLEAN:

```
BOOLEAN [NOT FULL]
```

Bạn định nghĩa các biến Boolean bằng cách gán null ngầm định hoặc bằng cách gán tường minh một giá trị TRUE hoặc FALSE. Cú pháp sau đây thuộc về khối khai báo:

```
var1 BOOLEAN; — Implicitly assigned a null value.
```

```
var2 BOOLEAN NOT NULL := TRUE; — Explicitly assigned a TRUE value.
```

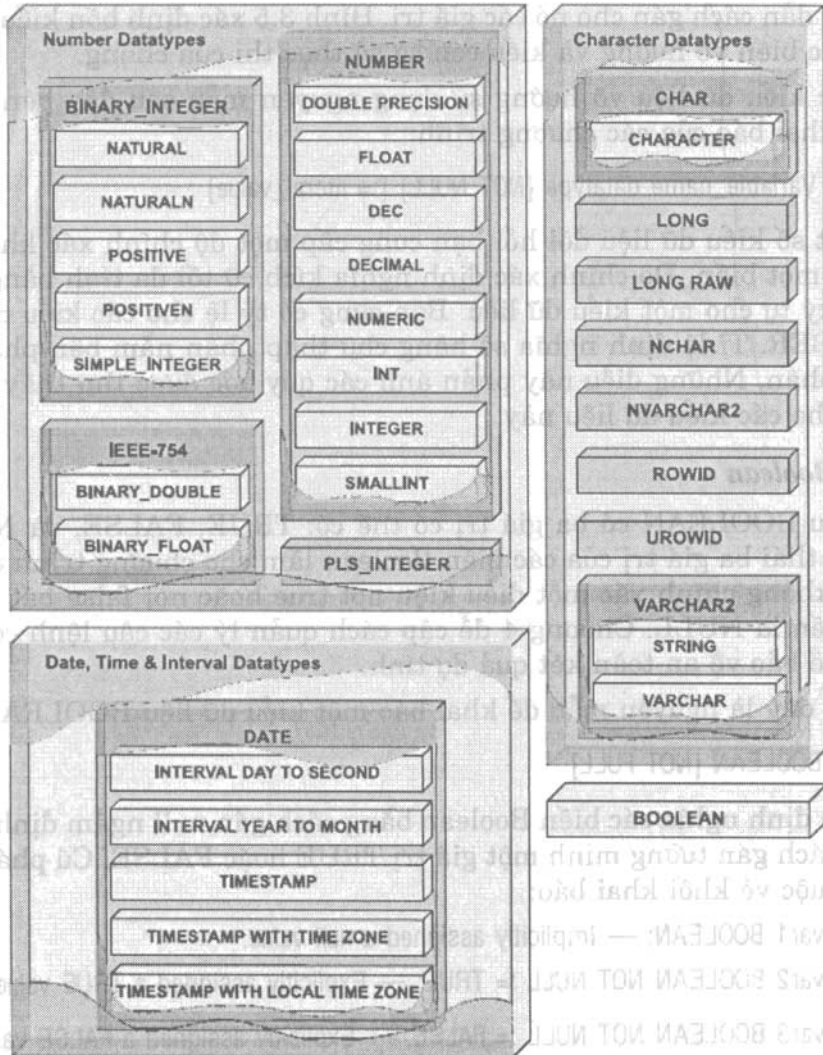
```
var3 BOOLEAN NOT NULL := FALSE; — Explicitly assigned a FALSE value.
```

Bạn luôn nên khởi tạo các biến Boolean một cách tường minh trong các đơn vị chương trình. Thói quen này tránh những hành vi bất ngờ trong những chương trình. Sử dụng mệnh đề NOT NULL trong quá trình khai báo bảo đảm các biến Boolean không bao giờ rỗng (null).

Ít cần tạo kiểu con (subtype) cho một kiểu dữ liệu BOOLEAN, nhưng bạn có thể làm điều này. Cú pháp tạo kiểu con là

```
SUBTYPE booked IS BOOLEAN;
```

Điều này tạo một kiểu con BOOKED vốn là một kiểu dữ liệu BOOLEAN không ràng buộc. Bạn có thể thấy điều này hữu dụng khi bạn cần một tên thứ hai cho một kiểu dữ liệu BOOLEAN, nhưng nói chung việc tạo kiểu con cho một Boolean thì không hữu dụng cho lắm.



Hình 3.5 Các kiểu vô hướng

Như được trình bày trong mục nhỏ trước "Các trực kiện Boolean", bạn gán cho một biến Boolean một giá trị trực kiện bên trong khối thực thi bằng cách sử dụng cú pháp sau đây:

```
var1 := TRUE;
```

SUBTYPE booked IS BOOLEAN;

Không giống như các chuỗi, các giá trị TRUE, FALSE, hoặc NULL không được tách bằng các dấu ngoặc đơn. Cả ba từ đều là các từ dành riêng PL/SQL.

Các ký tự và chuỗi

Các ký tự và chuỗi làm việc gần giống như lớp String trong ngôn ngữ lập trình Java. Các chuỗi (string) được gọi là các mảng ký tự một chiều trong các ngôn ngữ lập trình C và C++. Các kiểu dữ liệu ký tự (character) lưu trữ một chuỗi có chiều dài cố định. Bạn định kích cỡ chuỗi bằng cách cho biết số byte hoặc ký tự được phép bên trong chuỗi. Bất kỳ nỗ lực nhằm lưu trữ nhiều hơn số byte hoặc ký tự tối đa sẽ đưa ra một ngoại lệ.

Chương trình sau đây minh họa những điểm khác biệt về sự cấp phát bộ nhớ giữa các kiểu dữ liệu CHAR và VARCHAR2:

```
DECLARE
    c CHAR(32767) := ' ';
    v VARCHAR2(32767) := ' ';
BEGIN
    dbms_output.put_line('c is ['||LENGTH(c)||']');
    dbms_output.put_line('v is ['||LENGTH(v)||']');
    v := v || ' ';
    dbms_output.put_line('v is ['||LENGTH(v)||']'); END;
/
```

Chương trình định nghĩa hai biến, in chiều dài của chúng và sau đó ghép một giá trị khoảng trắng khác với VARCHAR2 để minh họa việc cấp phát bộ nhớ. Miễn là bạn đã đặt bộ đệm SQL*Plus (xác lập SERVEROUTPUT sang on), điều này sẽ xuất kết quả sau đây sang console:

```
c is [32767]
v is [1]
v is [2]
```

Kết quả cho thấy một biến CHAR xác lập kích cỡ bộ nhớ cấp phát khi được định nghĩa. Bộ nhớ cấp phát có thể vượt quá những gì được yêu cầu để quản lý giá trị trong biến. Kết quả cũng cho thấy biến VARCHAR2 cấp phát động chỉ bộ nhớ bắt buộc để chứa giá trị của nó.

Các kiểu dữ liệu CHAR và CHARACTER. Kiểu dữ liệu CHAR là một kiểu dữ liệu cơ sở cho các chuỗi có chiều dài cố định. Bạn có thể định kích cỡ một kiểu dữ liệu dài lên đến 32,767 byte, nhưng chiều dài mặc định của nó là 1 byte. Thật không may, một CHAR PL/SQL lớn hơn một tối đa 4.000 byte được phép trong một cột SQL CHAR. Bạn có thể lưu trữ

các chuỗi ký tự lớn hơn 4.000 byte trong các cột CLOB hoặc LONG. Oracle đề nghị bạn sử dụng kiểu dữ liệu CLOB bởi vì các kiểu dữ liệu LONG và LONG RAW chỉ được hỗ trợ cho các mục đích tương thích ngược.

Sau đây là nguyên mẫu để định nghĩa một kiểu dữ liệu CHAR:

```
CHAR[ (maximum_size [BYTE | CHAR] ) ] [NOT NULL]
```

Bốn cách để khai báo một biến sử dụng kiểu dữ liệu CHAR và một giá trị rỗng mặc định là

```
var1 CHAR; -- Implicitly sized at 1 byte.
```

```
var2 CHAR(1); -- Explicitly sized at 1 byte.
```

```
var3 CHAR(1 BYTE); -- Explicitly sized at 1 byte.
```

```
var4 CHAR(1 CHAR); -- Explicitly sized at 1 character.
```

Khi bạn sử dụng việc cấp phát không gian ký tự, kích cỡ tối đa thay đổi phụ thuộc vào tập hợp ký tự của cơ sở dữ liệu. Một số tập hợp ký tự sử dụng 2 hoặc 3 byte để lưu trữ các ký tự. Bạn chia 32.767 cho số byte được yêu cầu trên mỗi ký tự nghĩa là mất tối đa cho một CHAR là 16.383 cho một tập hợp ký tự 2 byte và 10.922 cho một tập hợp ký tự 3 byte.

Bạn có thể sử dụng mệnh đề NOT NULL để bảo đảm một giá trị được gán vào một biến CHAR. Thói quen chung là không giới hạn các biến CHAR mà không có lý do cơ bản bắt buộc khác.

Kiểu dữ liệu CHARACTER là một kiểu con của kiểu dữ liệu CHAR. Kiểu dữ liệu CHARACTER có cùng một dãy giá trị với kiểu cơ sở của nó. Thực ra nó là một kiểu dữ liệu bí danh và được gọi chính thức là một kiểu con không ràng buộc. Việc gán giữa các biến của các kiểu dữ liệu CHAR và CHARACTER được chuyển đổi ngầm định miễn là các biến có cùng một kích cỡ.

Kích cỡ cho các ký tự có hai yếu tố: số đơn vị được cấp phát và loại đơn vị được cấp phát. Một chuỗi ba ký tự (được dẫn xuất từ tập hợp ký tự) không thể nằm vừa trong một chuỗi ba byte và hiển nhiên hơn một chuỗi ba ký tự không thể nằm vừa trong một chuỗi hai ký tự. Bất kỳ nỗ lực nhằm thực hiện loại gán đó sẽ đưa ra một ORA-06502, nghĩa là một bộ đệm chuỗi ký tự quá nhỏ không thể chứa một giá trị.

Bạn có thể khai báo một kiểu con CHAR bằng cách sử dụng nguyên mẫu sau đây:

```
SUBTYPE subtype_name IS base_type (maximum_size [BYTE | CHAR] ) ] [NOT NULL] ;
```

Ví dụ này tạo và sử dụng một kiểu con ràng buộc CODE:

```
DECLARE
    SUBTYPE code IS CHAR(1 CHAR);
    c CHAR(1 CHAR) := 'A';
    d CODE;
BEGIN
    d := c;
END;
/
```

Các ký tự và chuỗi không thể xác định các dãy ký tự. Chúng có thể chỉ xác lập kích cỡ tối đa. Điều này khác với các hành vi định kiểu con của các số bởi vì chúng có thể giới hạn các dãy.

Việc toàn cầu hoá (globalization) đưa ra vô số vấn đề liên quan đến cách bạn sử dụng các chuỗi có chiều dài phổ biến. Bạn nên xem xét sử dụng các kiểu dữ liệu NCHAR khi quản lý nhiều tập hợp ký tự hoặc Unicode.

Các kiểu dữ liệu LONG và LONG RAW. Các kiểu dữ liệu LONG và LONG RAW chỉ được cung cấp để tương thích ngược. Bạn nên sử dụng CLOB hoặc NCLOB nơi bạn sẽ sử dụng một LONG và BLOB hoặc BFILE thay vì một LONG RAW. Kiểu dữ liệu LONG lưu trữ các luồng ký tự và LONG RAW lưu trữ các chuỗi nhị phân.

Các kiểu dữ liệu LONG và LONG RAW lưu trữ chuỗi ký tự có chiều dài phổ biến lên đến 32.760 byte trong các chương trình PL/SQL. Giới hạn này nhỏ hơn nhiều so với 2 gigabyte mà bạn có thể lưu trữ trong các cột cơ sở dữ liệu LONG hoặc LONG RAW. Kích cỡ tối đa kiểu dữ liệu LONG và LONG RAW thực sự nhỏ hơn mức tối đa cho các kiểu dữ liệu CHAR, NCHAR, VARCHAR2, và NVARCHAR2, và nó trở nên nhỏ bé trước 8 đến 128 terabyte của các kiểu dữ liệu LOB.

Sau đây là các nguyên mẫu để khai báo các kiểu dữ liệu LONG và LONG RAW:

```
LONG [NOT NULL]
```

```
LONG RAW [NOT NULL]
```

Bạn có thể sử dụng mệnh đề NOT NULL để bảo đảm một giá trị được gán vào các biến LONG và LONG RAW. Thói quen chung là không giới hạn những kiểu dữ liệu này mà không có một số lý do cơ bản bắt buộc khác.

Các kiểu dữ liệu LONG và LONG RAW có thể được khai báo với một giá trị rỗng mặc định bằng:

```
var1 LONG; -- Implicitly sized at 0 byte.
```

```
var2 LONG RAW; -- Implicitly sized at 0 byte.
```

Bạn có thể định nghĩa các biến của các kiểu này và gán các giá trị bằng cách sử dụng cú pháp sau đây:

```
var1 LONG := 'CAR' ;
```

```
var2 LONG RAW := HEXTORAW ( '43' || '41' || '52' ) ; -- CAR assigned in Hexadecimal.
```

Trong khi kiểu dữ liệu LONG dễ sử dụng, nó nhỏ so với các kiểu dữ liệu CLOB và NCLOB. Các kiểu dữ liệu CHAR hoặc VARCHAR2 cũng lưu trữ dữ liệu ký tự nhiều hơn kiểu dữ liệu LONG 7 byte.

••••• Thủ thuật

Bạn nên xem xét sử dụng các kiểu dữ liệu biến ánh xạ vào các kiểu dữ liệu cột bởi vì theo thời gian nó đơn giản hơn (rẻ hơn) cho các nhà lập trình bảo trì hỗ trợ. Bạn nên ghi chú các kiểu dữ liệu cột LONG sang các LOB.

Bạn nên chú ý hàm HEXTORAW () là bắt buộc để chuyển đổi các luồng thập lục phân thành các luồng thô (raw stream) trước khi gán sang các kiểu dữ liệu LONG RAW. Bất kỳ nỗ lực nhằm gán một luồng ký tự không được chuyển đổi đưa ra một ORA-06502 dưới dạng một lỗi chuyển đổi thập lục phân sang thô. Cũng nên chú ý luồng dữ liệu LONG RAW không được PL/SQL hiểu.

Các kiểu dữ liệu ROWID và UROWID. Kiểu dữ liệu ROWID ánh xạ sang giá cột ROWID trong bất kỳ bảng cơ sở dữ liệu Oracle. Bạn có thể chuyển đổi nó từ một ROWID thành một chuỗi 18 ký tự bằng cách sử dụng hàm ROWIDTOCHAR (), hoặc trở lại một chuỗi dữ liệu bằng cách sử dụng hàm CHARTOROWID ().

Kiểu dữ liệu ROWID là universal rowid. Nó làm việc với các định danh ROWID logic được lưu trữ bởi một bảng được tổ chức bằng index, trong khi kiểu dữ liệu ROWID thì không. Bạn nên sử dụng giá trị UROWID cho tất cả việc quản lý Oracle ROWID trong những chương trình PL/SQL, và khi bạn làm việc với các giá trị ROWID không phải Oracle.

Sau đây là những nguyên mẫu để khai báo các kiểu dữ liệu ROWID và UROWID:

```
ROWID
```

```
UROWID
```

Sự chuyển đổi ngầm định thì tốt cho các kiểu dữ liệu ROWID và UROWID. Hiếm khi cần sử dụng hàm ROWIDTOCHAR () hoặc CHARTOROWID ().

Kiểu dữ liệu VARCHAR2. Kiểu dữ liệu VARCHAR2 là một kiểu dữ liệu cơ sở cho các chuỗi có chiều dài khả biến. Bạn có thể định kích cỡ một kiểu dữ liệu VARCHAR2 dài lên đến 32.767 byte. Thật không may, một kiểu dữ liệu PL/SQL VARCHAR2 có thể lớn hơn kích cỡ tối đa 4.000 byte được lưu trữ trong một cột SQL VARCHAR2. Bạn có thể lưu trữ các kiểu dữ liệu lớn hơn 4.000 trong các cột CLOB hoặc LONG. Oracle đề nghị sử dụng kiểu dữ liệu CLOB bởi vì kiểu dữ liệu LONG chỉ được hỗ trợ cho những mục đích tương thích ngược.

Sau đây là nguyên mẫu để khai báo một kiểu VARCHAR2:

```
VARCHAR2 (maximum_size [BYTE | CHAR]) (NOT NULL)
```

Bạn có thể sử dụng mệnh đề NOT NULL để bảo đảm một giá trị được gán vào một biến VARCHAR2. Thói quen chung là không giới hạn các chuỗi chiều dài biến mà không có một số lý do cơ bản bắt buộc khác. Bạn nên xem xét lại một kiểu con vốn thi hành sự ràng buộc.

Có thể bạn nhận thấy kích cỡ vật lý bắt buộc cho các kiểu dữ liệu VARCHAR2, trong khi nó tùy chọn cho kiểu dữ liệu CHAR và các kiểu con của nó. Kích cỡ vật lý được bắt buộc bởi vì cơ sở dữ liệu cần biết bao nhiêu không gian để cấp phát cho một biến bằng cách sử dụng kiểu dữ liệu này. Khi bạn được kích cỡ một biến VARCHAR2 với 2000 byte không dây trở lên, bộ máy PL/SQL.

Chỉ cấp phát đủ không gian để quản lý các giá trị dữ liệu vật lý. Điều này thường được tô hoá thời gian chạy chương trình.

••••• Thủ thuật

Oracle 11g cấp phát 1.999 byte khi bạn khai báo một biến VARCHAR2 gồm 1.999 byte bất kể vật lý của dữ liệu. Các chuỗi chiều dài phổ biến lớn luôn nên được định nghĩa là 2000 byte trở lên.

Có ba cách để định nghĩa một biến VARCHAR2 với một biến giá trị rỗng mặc định:

```
var1 VARCHAR2(100); — Explicitly sized at 100 byte.
```

```
var2 VARCHAR2(100 BYTE); — Explicitly sized at 100 byte.
```

```
var3 VARCHAR2(100 CHAR); — Explicitly sized at 100 character.
```

Khi bạn sử dụng việc cấp phát không gian ký tự, kích cỡ tối đa thay đổi phụ thuộc vào tập hợp ký tự của cơ sở dữ liệu. Một số tập hợp sử dụng hai hoặc ba byte để lưu trữ ký tự. Bạn chia 32.767 với số byte được yêu

cầu cho mỗi ký tự nghĩa là mức tối đa cho một VARCHAR là 16,383 cho một tập hợp ký tự hai byte và 10.922 cho một tập hợp ký tự ba byte.

Các kiểu dữ liệu STRING và VARCHAR là các kiểu con của kiểu dữ liệu VARCHAR2. Chúng đều có cùng một dãy giá trị với kiểu cơ sở VARCHAR2. Thật ra, chúng là các bí danh và được gọi chính thức là các kiểu con không ràng buộc. Các phép gán giữa các biến của những kiểu con này được chuyển đổi ngầm định miễn là các biến có cùng một kích cỡ.

Kích cỡ cho các chuỗi có hai yếu tố: số đơn vị được cấp phát và loại đơn vị được cấp phát. Một chuỗi ba ký tự (được dẫn xuất từ tập hợp ký tự) không thể nằm vừa cho một chuỗi ba byte, và hiển nhiên hơn một chuỗi ba ký tự không thể nằm vừa trong một chuỗi hai ký tự. Bất kỳ nỗ lực nhằm thực hiện kiểu gán này sẽ đưa ra một lỗi ORA-06502, nghĩa là một bộ đệm chuỗi ký tự quá nhỏ không thể chứa một giá trị.

Bạn có thể khai báo một kiểu con VARCHAR2 bằng cách sử dụng nguyên mẫu sau đây:

```
SUBTYPE subtype_name IS base_type(maximum_size [BYTE | CHAR]) [NOT NULL] ;
```

Ví dụ này tạo một kiểu con ràng buộc DB_STRING:

```
DECLARE
    SUBTYPE db_string IS VARCHAR2(4000 BYTE);
    c VARCHAR2(1 CHAR) := 'A';
    d DB_STRING;
BEGIN
    d := c;
END;
/
```

Ví dụ này tạo một kiểu con vốn không thể vượt quá giới hạn vật lý cho một cột VARCHAR2. Nó làm việc nhất quán bất kể tập hợp cơ sở dữ liệu. Điều này có thể hữu dụng khi bạn muốn bảo đảm việc tuân theo cơ sở dữ liệu vật lý trong các khối mã PL/SQL.

Các chuỗi không thể xác định các dãy ký tự theo cách các kiểu con số có thể xác định các số dãy. Chúng chỉ có thể xác lập kích cỡ tối đa vốn có thể được ghi đè bằng cách khai báo kiểu con bằng một kích cỡ tối đa mới nhỏ hơn hoặc bằng 32.767 byte.

Việc toàn cầu hoá (globalization) đưa ra vô số vấn đề liên quan đến cách bạn sử dụng các chuỗi có chiều dài khả biến. Bạn nên xem xét sử dụng các kiểu dữ liệu NVARCHAR2 khi quản lý nhiều tập hợp ký tự hoặc Unicode.

Các ngày tháng, thời gian, và khoảng thời gian

Kiểu dữ liệu DATE là kiểu cơ sở cho các ngày tháng, thời gian, và khoảng thời gian. Có hai kiểu con để quản lý các khoảng thời gian intervals và ba kiểu con để quản lý các tem thời gian. Ba mục nhỏ tiếp theo đề cập đến ngày tháng, các khoảng thời gian, và tem thời gian.

Kiểu dữ liệu DATE. Kiểu dữ liệu DATE trong Oracle chứa một tem thời gian hoạt động thực sự. Dãy hợp lệ là bất kỳ ngày tháng từ January 1, 4712 BCE (Before Common Era) đến December 31, 9999 CE (Common Era). Cách phổ biến nhất để thu thập một tem thời gian là gán hàm cài sẵn SYSDATE hoặc SYSTIMESTAMP. Chúng đều trả về các ngày tháng xác định đầy đủ và chứa tất cả phần tử trường của một biến hoặc cột DATE. Index trường cho một kiểu dữ liệu DATE nằm trong bảng 3.2.

Bảng 3.2 Index trường kiểu dữ liệu DATE

Tên trường	Dãy hợp lệ	Các giá trị khoảng thời gian hợp lệ
YEAR	-4712 đến 9999 (ngoại trừ năm 0)	Bất kỳ số nguyên khác 0
MONTH	01 đến 12	0 đến 11
DAY	01 đến 31 (giới hạn bằng các quy tắc lịch)	Bất kỳ số nguyên khác 0
HOURL	00 đến 23	0 đến 23
MINUTE	00 đến 59	0 đến 59
SECOND	00 đến 59	0 đến 59.9 (trong đó các 1/10 là khoảng phân số thứ hai)
TIMEZONE_HOUR	-12 đến 14 (dãy điều chỉnh cho các thay đổi thời gian kéo dài giờ làm việc ban ngày)	Không thể áp dụng
TIMEZONE_MINUTE	00 đến 59	Không thể áp dụng
TIMEZONE_REGION	Giá trị trong V\$TIMEZONE_NAMES	Không thể áp dụng
TIMEZONE_ABBR	Giá trị trong V\$TIMEZONE_NAMES	Không thể áp dụng

Sau đây là nguyên mẫu để khai báo một kiểu dữ liệu DATE:

DATE [NOT NULL]

Bạn có thể sử dụng mệnh đề NOT NULL để bảo đảm một giá trị được gán vào một biến DATE. Có những trường hợp bạn sẽ muốn giới hạn các biến DATE. Nếu bạn không giới hạn chúng thì bạn sẽ cần bao bọc chúng

trong các hàm cài sẵn NVL () để hỗ trợ các phép so sánh logic.

Bạn có thể định nghĩa một biến DATE với một giá trị rỗng hoặc giá trị khởi tạo mặc định như sau:

```
var1 DATE; — Implicitly assigns a null value.
var2 DATE := SYSDATE; — Explicitly assigns current server timestamp.
var3 DATE := SYSDATE + 1; — Explicitly assigns tomorrow server timestamp.
var4 DATE := '29-FEB-08'; — Explicitly assigns leap year day for 2008.
```

Hàm TO_DATE () cũng có thể chuyển đổi các định dạng ngày tháng không phù hợp thành các giá trị DATE hợp lệ. Hoặc hàm CAST () cũng làm việc bằng các mặt nạ định dạng mặc định. Các mặt nạ định dạng mặc định cho các ngày tháng là DD-MON-RR hoặc DD-MON-YYYY.

Sử dụng TRUNC (date_variable) khi bạn muốn trích xuất một ngày tháng từ một tem thời gian. Điều này hữu dụng khi bạn muốn tìm tất cả giao dịch xảy ra vào một ngày cụ thể. Theo mặc định, hàm cài sẵn TRUNC () cắt tỉa thời gian, tạo một ngày có 00 giờ, 00 phút, và 00 giây. Chương trình sau đây minh họa khái niệm này

```
DECLARE
    d DATE := SYSDATE;
BEGIN
    dbms_output.put_line(TO_CHAR(TRUNC(d), 'DD-MON-YY HH24:MI:SS'));
END;
/
```

Việc chạy script này sẽ tạo ra kết quả sau đây:

```
31-JUL-07 00:00:00
```

Hàm cài sẵn EXTRACT () cũng cho bạn thu thập tháng, năm, hoặc ngày dạng số từ một giá trị DATE.

Bạn có thể khai báo một kiểu con DATE bằng cách sử dụng nguyên mẫu sau đây:

```
SUBTYPE subtype_name IS base_type [NOT NULL] ;
```

Bạn nên chú ý rằng khi sử dụng các kiểu con ký tự, bạn không thể xác lập một dãy ngày tháng. Tạo một kiểu con DATE vốn đòi hỏi một giá trị thì có thể. Sử dụng DATEN cho một DATE bắt buộc rỗng tuân theo quy ước được sử dụng bởi các kiểu con NATURALN và POSITVEN.

Các kiểu con Interval. có hai kiểu con DATE cho phép quản lý các khoảng thời gian: INTERVAL TO SECOND và INTERVAL YEAR TO MONTH. Các nguyên mẫu của chúng là

```
INTERVAL DAY [ (leading_precision) ] TO SECOND [ (fractional_second_precision) ]
```

```
INTERVAL YEAR [ (precision) ] TO MONTH
```

Giá trị mặc định cho độ chính xác đứng trước của ngày là 2, và độ chính xác thứ hai phân số của giây là 6. Giá trị mặc định cho độ chính xác của năm là 2.

Bạn có thể định nghĩa một biến INTERVAL DATE TO SECOND với một giá trị rỗng hoặc giá trị khởi tạo mặc định, như được trình bày:

```
var1 INTERVAL DAY TO SECOND; -- Implicitly accept default precisions.
```

```
var2 INTERVAL DAY(3) TO SECOND; -- Explicitly set day precision.
```

```
var3 INTERVAL DAY(3) TO SECOND(9); -- Explicitly set day and second precision.
```

Bạn gán một giá trị biến bằng cách sử dụng nguyên mẫu sau đây cho một kiểu dữ liệu INTERVAL DAY TO SECOND, trong đó D, tượng trưng cho day (ngày) và HH:MI:SS tượng trưng cho giờ (hour), phút (minute), và giây (second):

```
variable_name := 'D HH:MI:SS' ;
```

Một phép gán thực sự vào cùng một kiểu trông như sau

```
var1 := '5 08:21:20'; -- Implicit conversion from the string.
```

Bạn có thể khai báo một biến INTERVAL YEAR TO MONTH với một giá trị rỗng hoặc giá trị khởi tạo mặc định như được trình bày:

```
var1 INTERVAL YEAR TO MONTH; -- Implicitly accept default precisions.
```

```
var2 INTERVAL YEAR (3) TO MONTH; -- Explicitly set year precision.
```

Có bốn phương thức gán. Chương trình sau đây minh họa một phép gán vào var2:

```
DECLARE
```

```
var2 INTERVAL YEAR(3) TO MONTH;
```

```
BEGIN
```

```
-- Shorthand for a 101 year and 3 month interval.
```

```
var2 := '101-3';
```

```
var2 := INTERVAL '101-3' YEAR TO MONTH;
```

```
var2 := INTERVAL '101' YEAR;
```

```

var2 := INTERVAL '3' MONTH;
END;
/

```

Điều này lần lượt xuất các giá trị sau đây:

```

+101-03
+101-03
+101-00
+000-03

```

Các phép toán số học giữa kiểu dữ liệu DATE và các kiểu con interval tuân theo các quy tắc trong bảng 3.3. Phép toán đặc trưng là một phép tính khoảng thời gian như lấy một tem thời gian khác trừ cho một tem thời gian để có được số ngày giữa các ngày tháng.

Các khoảng thời gian giúp đơn giản hoá các phép so sánh nâng cao nhưng cần nỗ lực thêm để nắm vững.

Bảng 3.3 Tem thời gian và số học khoảng thời gian

Loại toán hạng 1	Toán tử	Loại toán hạng 2	Loại kết quả
Timestamp	+	Interval	Timestamp
Timestamp	-	Interval	Timestamp
Interval	+	Timestamp	Timestamp
Timestamp	-	Interval	Interval
Interval	+	Interval	Interval
Interval	-	Interval	Interval
Interval	*	Numeric	Interval
Numeric	*	Interval	Interval
Interval	/	Numeric	Interval

Các kiểu con TIMESTAMP. Kiểu con TIMESTAMP mở rộng kiểu cơ sở DATE bằng cách cung cấp một thời gian chính xác hơn. Bạn sẽ nhận được các kết quả tương tự nếu biến TIMESTAMP được tập hợp lại bằng cách gọi SYSDATE cài sẵn. SYSTIMESTAMP cung cấp một thời gian chính xác hơn phụ thuộc vào nền.

Sau đây là nguyên mẫu để khai báo một kiểu dữ liệu TIMESTAMP:

```
TIMESTAMP [ (precision) ] [NOT NULL]
```

Bạn có thể xác lập mệnh đề NOT NULL để bảo đảm một giá trị được gán vào một biến TIMESTAMP. Có nhiều trường hợp nơi bạn sẽ muốn giới hạn các biến TIMESTAMP. Nếu bạn không giới hạn chúng thì bạn sẽ cần bao bọc chúng trong các hàm cài sẵn NVL () để hỗ trợ các phép so sánh logic.

Bạn có thể định nghĩa một biến **TIMESTAMP** với một giá trị rỗng hay giá trị khởi tạo mặc định như được trình bày:

- var1 **TIMESTAMP**; — Implicitly assigns a null value.
- var2 **TIMESTAMP := SYSTIMESTAMP**; — Explicitly assigns a value.
- var3 **TIMESTAMP(3)**; — Explicitly sets precision for null value.
- var4 **TIMESTAMP(3) := SYSTIMESTAMP**; — Explicitly sets precision and value.

Chương trình sau đây minh họa sự khác biệt giữa các kiểu dữ liệu **DATE** và **TIMESTAMP**:

```

DECLARE
    d DATE := SYSTIMESTAMP;
    t TIMESTAMP(3) := SYSTIMESTAMP;
BEGIN
    dbms_output.put_line('DATE ['|| d || ']');
    dbms_output.put_line('TO_CHAR ['|| TO_CHAR(d,'DD-MON-YY
    HH24:MI:SS') || ']');
    dbms_output.put_line('TIMESTAMP ['|| t || ']');
END;
/
    
```

Khối nặc danh quay trở lại

```

DATE [31-JUL-07]
TO_CHAR [31-JUL-07 21:27:36]
TIMESTAMP [31-JUL-07 09.27.36.004 PM]
    
```

Hai kiểu dữ liệu **timestamp** khác minh họa những hành vi tương tự. Các nguyên mẫu của chúng là:

```

TIMESTAMP [ (precision) ] WITH TIME ZONE
TIMESTAMP [ (precision) ] WITH LOCAL TIME ZONE
    
```

Bạn có thể khai báo một biến **TIMESTAMP WITH TIME ZONE** với một giá trị rỗng hay giá trị khởi tạo mặc định, như được trình bày:

```

var1 TIMESTAMP WITH LOCAL TIME ZONE;
var2 TIMESTAMP WITH LOCAL TIME ZONE := SYSTIMESTAMP;
var3 TIMESTAMP(3) WITH LOCAL TIME ZONE;
var4 TIMESTAMP(3) WITH LOCAL TIME ZONE := SYSTIMESTAMP;
    
```

Sự khác biệt giữa các tem thời gian (**timestamp**) này là các tem thời gian có các múi giờ (**time zone**) thêm múi giờ vào tem thời gian. **Time zone qualifier** trả về thời gian chuẩn và một bộ chỉ báo cho thấy múi giờ

có sử dụng thời gian kéo dài giờ làm việc ban ngày (daylight saving) hay không. Time zone qualifier cục bộ trả về hiệu giữa thời gian cục bộ và Greenwich Mean Time (GMT).

Các ký tự và chuỗi Unicode

Các ký tự và chuỗi Unicode hiện hữu để hỗ trợ việc toàn cầu hoá (globalization). Sự toàn cầu hoá được thực hiện bằng cách sử dụng kiểu mã hoá ký tự vốn hỗ trợ nhiều tập hợp ký tự. Kiểu mã hoá AL16UTF16 hoặc UTF8 được cung cấp bởi cơ sở dữ liệu Oracle. Kiểu mã hoá AL16UTF16 lưu trữ tất cả ký tự trong hai byte vật lý trong khi kiểu mã hoá UTF8 lưu trữ tất cả ký tự trong ba byte vật lý.

Kiểu dữ liệu NCHAR là kiểu dữ liệu Unicode tương đương dữ liệu CHAR, và kiểu dữ liệu NVARCHAR2 là kiểu dữ liệu Unicode tương đương với kiểu dữ liệu VARCHAR2. Bạn nên sử dụng những kiểu dữ liệu này khi xây dựng các ứng dụng mà sẽ hỗ trợ nhiều tập hợp ký tự trong cùng một cơ sở dữ liệu.

Kiểu dữ liệu NCHAR. Kiểu dữ liệu NCHAR là một kiểu dữ liệu cơ sở cho các chuỗi Unicode khả biến. Kiểu dữ liệu NCHAR chia sẻ chiều dài tối đa 32,767 byte cho tất cả kiểu dữ liệu và chuỗi khác. Bạn có thể lưu trữ chiều dài lưu trữ 16.383 (32.767 chia cho 2) ký tự sử dụng kiểu mã hoá AL16UTF16 hoặc 10.922 (32.767 chia cho 3) ký tự sử dụng kiểu mã hoá UTF8.

Như kiểu dữ liệu CHAR, kiểu dữ liệu NCHAR cũng là một kiểu dữ liệu chuỗi có chiều dài cố định. Một chuỗi các chiều dài cố định xác lập kích cỡ vật lý của biến trong bộ nhớ bất kể kích cỡ thực của giá trị bên trong biến.

Kiểu dữ liệu PL/SQL NCHAR có thể lớn hơn mức tối đa 4000 byte được lưu trữ trong một cột SQL NCHAR. Bạn nên lưu trữ các chuỗi ký tự Unicode lớn hơn 4000 byte trong các cột NCLOB.

Sau đây là nguyên mẫu để khai báo một kiểu dữ liệu NCHAR:

```
NCHAR( (maximum_size) ) [NOT NULL]
```

Có thể bạn đã thấy kích cỡ vật lý cho kiểu dữ liệu NCHAR khác với kích cỡ vật lý của các kiểu dữ liệu CHAR và VARCHAR2. Không có tùy chọn để xác định các byte hoặc ký tự khi khai báo việc cấp phát không gian cho các biến NCHAR. Không gian Unicode luôn được cấp phát trong các ký tự bằng một trục kiện số.

Bạn có thể sử dụng mệnh đề NOT NULL để bảo đảm giá trị được gán vào một biến NCHAR. Thói quen chung là không có lý do cơ bản bắt buộc khác.

Chỉ có một cách để định nghĩa một biến NCHAR với một giá trị rỗng mặc định:

```
var1 NCHAR; -- Implicitly sized at 1 character.
```

```
var1 NCHAR (100) ; -- Explicitly sized at 100 character.
```

Bạn có thể khai báo một kiểu con NCHAR bằng cách sử dụng nguyên mẫu sau đây:

```
SUBTYPE subtype_name IS base_type(maximum_size) [NOT NULL] ;
```

Các thay đổi kích cỡ tối đa phụ thuộc vào kiểu mã hoá ký tự Unicode được đề cập. Mức tối đa là 16.383 ký tự sử dụng kiểu mã hoá AL16UTF16 hoặc 10.922 ký tự sử dụng kiểu mã hoá UTF8. Bất kỳ nỗ lực nhằm xác định một kích cỡ tối đa bằng một từ khoá BYTE đưa ra một lỗi không cho phép ngữ nghĩa byte, đây là một mã lỗi PLS-00639.

Sự toàn cầu hoá thích hợp nhất cho các kiểu dữ liệu NCHAR hoặc NVARCHAR2. Bạn nên sử dụng những kiểu này khi cơ sở dữ liệu hỗ trợ Unicode hoặc có thể hỗ trợ ở nó trong tương lai.

Kiểu dữ liệu NVARCHAR2. Kiểu dữ liệu NVARCHAR2 là một kiểu dữ liệu cơ sở cho các chuỗi Unicode khả biến. Các kiểu dữ liệu NVARCHAR2 chia sẻ chiều dài tối đa 32.767 byte cho tất cả dữ liệu ký tự và chuỗi khác. Bạn có thể lưu trữ chiều dài tối đa 16.383 (32.767 chia cho 2) ký tự sử dụng kiểu mã hoá AL16UTF16 hoặc 10.922 (32.767 chia cho 3) ký tự sử dụng kiểu mã hoá UTF8.

Như các kiểu ký tự khác, kiểu dữ liệu PL/SQL NVARCHAR2 có thể lớn hơn mức tối đa 4000 byte được lưu trữ trong cột SQL NVARCHAR2. Bạn nên lưu trữ các chuỗi ký tự Unicode lớn hơn 4000 byte trong các cột NCLOB.

Sau đây là nguyên mẫu để khai báo một kiểu dữ liệu NVARCHAR2:

```
NVARCHAR2 (maximum_size) [NOT]
```

Có thể bạn thấy rằng kích cỡ vật lý cho kiểu dữ liệu NVARCHAR2 khác với kích cỡ vật lý của kích cỡ dữ liệu CHAR và VARCHAR2. Không có tùy chọn để xác định các byte hoặc ký tự khi khai báo việc cấp phát không gian cho các biến NVARCHAR2. Không gian Unicode luôn được cấp phát cho các ký tự bằng một trực diện số.

Bạn có thể sử dụng mệnh đề NOT NULL để bảo đảm một giá trị được gán vào một biến NVARCHAR2. Thói quen chung là không giới hạn các biến chuỗi mà không có lý do cơ bản bắt buộc khác. Bạn nên xem xét tạo một kiểu con vốn thi hành sự ràng buộc.

Chỉ có một cách để định nghĩa một biến NVARCHAR2 với một giá trị rỗng mặc định:

```
var1 NVARCHAR2 (100) ; -- Explicitly sized at 100 character.
```

Bạn có thể định nghĩa một kiểu con NVARCHAR2 bằng cách sử dụng nguyên mẫu sau đây:

```
SUBTYPE subtype_name IS base_type (maximum_size) [NOT NULL] ;
```

Các số

Có bốn kiểu dữ liệu số chính. Các kiểu dữ liệu là định dạng BINARY_INTEGER, IEEE 754 (BINARY_DOUBLE và BINARY_FLOAT), NUMBER, và PLS_INTEGER. Các kiểu dữ liệu BINARY_INTEGER và PLS_INTEGER thì giống nhau, cả hai đều sử dụng các thư viện toán học riêng hệ điều hành. Oracle sử dụng PLS_INTEGER để mô tả cả BINARY_INTEGER và PLS_INTEGER là có thể thay thế cho nhau và sách này cũng vậy.

Các số định dạng IEEE 754 cung cấp các số với độ chính xác đơn và độ chính xác kép để hỗ trợ việc tính toán khoa học. Kiểu dữ liệu NUMBER sử dụng một thư viện (library) tùy ý được cung cấp như là một phần của Oracle 11g Database. Nó có thể lưu trữ các số dấu cố định hoặc dấu động rất lớn.

Kiểu dữ liệu BINARY_INTEGER. Kiểu dữ liệu BINARY_INTEGER giống hệt như PLS_INTEGER và lưu trữ các số nguyên từ -2,147,483,648 đến 2,147,483,647 dưới dạng 32 bit hoặc 4 byte. Như kiểu PLS_INTEGER, nó tính toán hiệu quả hơn trong dãy số của nó và chiếm ít không gian hơn nhiều so với kiểu dữ liệu NUMBER trong bộ nhớ. Các phép toán sử dụng hai biến BINARY_INTEGER tạo ra một kết quả bên ngoài dãy kiểu dữ liệu sẽ đưa ra một lỗi chèn số ORA-01426. Sau đây là nguyên mẫu để khai báo một kiểu dữ liệu BINARY_INTEGER:

```
BINARY_INTEGER
```

Bạn có thể định nghĩa một biến BINARY_INTEGER với một giá trị rỗng hoặc được khởi tạo trong quá trình khai báo. Cú pháp cho cả hai như sau:

```
var1 BINARY_INTEGER;
```

```
var2 BINARY_INTEGER := 21;
```

BINARY_INTEGER sử dụng các thư viện toán học riêng, và do đó câu lệnh khai báo không cấp phát bộ nhớ để lưu trữ biến cho đến khi một giá trị được gán.

Bạn có thể định nghĩa một kiểu con BINARY_INTEGER bằng cách sử dụng nguyên mẫu sau đây:

```
SUBTYPE subtype_name IS base_type [RANGE low_number...high_number] [NOT NULL] ;
```


Có một số kiểu con định nghĩa sẵn của kiểu `BINARY_INTEGER`. Các kiểu con `NATURAL` và `POSITIVE` giới hạn việc sử dụng chúng chỉ trong các giá trị số nguyên dương. Các kiểu con `NATURALN` và `POSITIVEN` giới hạn các phép gán rỗng. Một lỗi PLS-00218 được đưa ra khi bạn cố khai báo một `NATURALN` hoặc `POSITIVEN` mà không khởi tạo giá trị. Chúng đề nghị thực thi một sự ràng buộc không rỗng trên kiểu dữ liệu.

Kiểu con mới nhất là kiểu dữ liệu `SIMPLE_INTEGER` được giới thiệu trong Oracle 11g. Nó cắt xén sự tràn (overflow) và bỏ qua việc đưa ra bất kỳ lỗi liên quan đến sự tràn. Hiệu suất của kiểu `SIMPLE_INTEGER` phụ thuộc vào giá trị của tham số cơ sở dữ liệu `plsql_code_type`. Hiệu suất cao hơn khi kiểu `plsql_code_type` được xác lập sang `NATIVE` bởi vì các phép toán số học được thực thi với các thư viện hệ điều hành và cả việc kiểm tra sự chèn số và kiểm tra giá trị rỗng được vô hiệu hoá. Hiệu suất chậm hơn khi `plsql_code_type` được xác lập sang `INTERPRETED` bởi vì nó ngăn sự quá tải và tiến hành việc kiểm tra giá trị rỗng.

Bạn cũng nên biết rằng một thao tác cast từ một `PLS_INTEGER` hoặc `BINARY_INTEGER` vào một `SIMPLE_INTEGER` không thực hiện sự chuyển đổi trừ phi giá trị rỗng. Một lỗi run-time được đưa ra khi cast một giá trị rỗng vào một biến `SIMPLE_INTEGER`.

Kiểu dữ liệu định dạng IEEE 754-. Các số độ chính xác đơn và độ chính xác của định dạng IEEE 754 được cung cấp để hỗ trợ việc tính toán khoa học. Chúng tạo ra các vấn đề tràn và vô cực truyền thống như là một phần định nghĩa và thực thi của chúng.

Cả hai môi trường SQL và PL/SQL định nghĩa các hằng `BINARY_FLOAT_NAN` và `BINARY_FLOAT_INFINITY`. Môi trường PL/SQL cũng định nghĩa bốn hằng khác. Tất cả 6 hằng được tìm thấy với những giá trị của chúng trong bảng 3.4.

Ghi chú

Tài liệu Oracle 11g Database không liệt kê những ràng buộc này trong các danh sách từ dành riêng hoặc từ khoá. Chúng thường được tìm thấy bằng cách in chúng từ một chương trình PL/SQL hoặc truy vấn bảng `V$RESERVED_WORDS`.

Sau đây là nguyên mẫu để khai báo các kiểu dữ liệu IEE-754:

```
BINARY_DOUBLE
```

```
BINARY_FLOAT
```

Bạn có thể định nghĩa các biến của những kiểu này với những giá trị rỗng hoặc khởi tạo chúng trong quá trình khai báo. Cú pháp cho cả hai như sau:

```

var1 BINARY_DOUBLE;
var2 BINARY_DOUBLE := 21D;
var3 BINARY_FLOAT;
var4 BINARY_FLOAT := 21f;

```

Bảng 3.4 Các hằng IEEE-754

Tên hằng	Môi trường	Giá trị
BINARY_FLOAT_NAN	SQL & PL/SQL	Nó chứa Nan, nhưng các phép toán so sánh xem nó là một chuỗi không nhạy kiểu chữ. NaN trong ký hiệu khoa học là Not a Number.
BINARY_FLOAT_INFINITY	SQL & PL/SQL	Nó chứa Inf, nhưng các phép toán so sánh xem nó là một chuỗi không nhạy kiểu chữ.
BINARY_FLOAT_MIN_NORMAL	PL/SQL	Nó chứa 1.17549435E-038.
BINARY_FLOAT_MAX_NORMAL	PL/SQL	Nó chứa 3.40282347e+038.
BINARY_FLOAT_MIN_SUBNORMAL	PL/SQL	Nó chứa 1.40129846E-045.
BINARY_FLOAT_MAX_SUBNORMAL	PL/SQL	Nó chứa 1.17549421E-038.

Bạn phải luôn sử dụng một d cho các trục kiện số được gán vào BINARY_DOUBLE và một f cho các trục kiện số được gán vào một BINARY_FLOAT. Oracle 11g Database quá tải các thường trình con nào tận dụng tốc độ xử lý của các kiểu dữ liệu IEEE-754 này.

Bạn cũng có thể định nghĩa một kiểu con BINARY_DOUBLE hoặc BINARY_FLOAT bằng cách sử dụng nguyên mẫu sau đây:

```
SUBTYPE subtype_name IS base_type [NOT NULL] ;
```

Bạn nên chú ý rằng không giống như các kiểu dữ liệu số khác, những kiểu dữ liệu số này không thể bị ràng buộc bằng dãy. Ràng buộc duy nhất mà bạn có thể áp đặt là các kiểu con không cho phép gán giá trị rỗng.

Kiểu dữ liệu NUMBER. Dữ liệu NUMBER sử dụng một thư viện tùy ý được cung cấp như là một phần của Oracle 11g Database. Nó có thể lưu trữ các số trong dãy 1.0E-130 (1 nhân với 10 nâng lên thành lũy thừa âm 130) đến 1.0E126 (1 nhân với 10 nâng lên lũy thừa 126). Oracle đề nghị sử dụng kiểu dữ liệu NUMBER chỉ khi việc sử dụng hoặc kết quả tính toán bị thất bại trong dãy giá trị có thể có. Kiểu dữ liệu NUMBER không

đưa ra một lỗi NaN (not a number) hoặc vô cực (infinity) khi một giá trị trực kiện hoặc giá trị tính toán nằm bên ngoài dãy kiểu dữ liệu. Những ngoại lệ này có những kết quả sau đây:

- Một giá trị trực kiện bên dưới giá trị dãy tối thiểu lưu trữ một zero trong một biến NUMBER.
- Một giá trị trực kiện ở trên giá trị dãy tối đa đưa ra một lỗi biên dịch.
- Một kết quả tính toán trên giá trị dãy tối đa đưa ra một lỗi biên dịch.

Kiểu dữ liệu NUMBER hỗ trợ các số dấu cố định và dấu động. Các số dấu cố định được định nghĩa bằng cách xác định số chữ số (được gọi là độ chính xác) và số chữ số nằm bên phải dấu thập phân được gọi là thang. Dấu thập phân không được lưu trữ vật lý trong biến bởi vì nó được tính bằng mối quan hệ giữa độ chính xác và thang.

Sau đây là nguyên mẫu để khai báo một kiểu dữ liệu NUMBER dấu cố định:

```
NUMBER [ (precision, [scale] ) ] [NOT NULL]
```

Cả độ chính xác và thang là những giá trị tùy ý khi bạn khai báo một biến NUMBER. Kích cỡ mặc định kiểu dữ liệu NUMBER, số chữ số, hoặc độ chính xác là 18. Bạn có thể khai báo một biến NUMBER với chỉ độ chính xác nhưng bạn phải xác định độ chính xác để định nghĩa thang.

Bạn có thể khai báo các biến NUMBER dấu cố định với các giá trị rỗng hoặc định nghĩa chúng trong quá trình khai báo. Cú pháp cho các phần khai báo kiểu dữ liệu NUMBER với các giá trị rỗng là

```
var1 NUMBER;           -- A null number with 38 digits.
var2 NUMBER(15);       -- A null number with 15 digits.
var3 NUMBER(15,2);     -- A null number with 15 digits and 2 decimals.
```

Cú pháp cho các phần khai báo kiểu dữ liệu NUMBER với những giá trị khởi tạo là

```
var1 NUMBER := 15;     -- A number with 38 digits.
var2 NUMBER(15) := 15; -- A number with 15 digits.
var3 NUMBER(15,2) := 15.22; -- A number with 15 digits and 2 decimals.
```

Bạn cũng có thể khai báo các số dấu cố định bằng cách sử dụng các kiểu con DEC, DECIMAL, và NUMERIC. Hoặc, bạn có thể khai báo các số nguyên bằng cách sử dụng các kiểu con INTEGER, INT, và SMALLINT. Chúng đều có cùng một độ chính xác tối đa là 38.

Sau đây là các nguyên mẫu để khai báo một kiểu dữ liệu NUMBER dấu động được gọi là các kiểu con DOUBLE PRECISION hoặc FLOAT:

```
DOUBLE PRECISION [ (precision) ]
FLOAT [ (precision) ]
```

Định nghĩa độ chính xác của các biến DOUBLE PRECISION hoặc FLOAT thì tùy chọn. Bạn gặp rủi ro mất đi độ chính xác tự nhiên của một số dấu động khi bạn ràng buộc độ chính xác. Cả hai biến này có một kích cỡ mặc định, số chữ số, hoặc độ chính xác 126. Bạn có thể định nghĩa độ chính xác của một biến FLOAT, nhưng không phải thang. Bất kỳ nỗ lực nhằm định nghĩa thang của một trong hai kiểu con này sẽ đưa ra một lỗi PLS-00510 bởi vì chúng không thể có một số chữ số cố định nằm bên phải dấu thập phân.

Cú pháp cho các phần khai báo DOUBLE PRECISION hoặc FLOAT với các giá trị rỗng là

```
var1 DOUBLE PRECISION;    -- A null number with 126 digits.
var2 FLOAT;               -- A null number with 15 digits.
var3 DOUBLE PRECISION;    -- A null number with 126 digits.
var4 FLOAT(15);           -- A null number with 15 digits.
```

Cú pháp cho các phần khai báo DOUBLE PRECISION hoặc FLOAT với các giá trị khởi tạo là

```
var1 DOUBLE PRECISION := 15;    -- A number with 126 digits.
var2 FLOAT := 15;               -- A number with 126 digits.
var3 DOUBLE PRECISION(15) := 15; -- A number with 15 digits.
var4 FLOAT(15) := 15;           -- A number with 15 digits.
```

Bạn cũng có kiểu con REAL của NUMBER lưu trữ các số dấu động nhưng chỉ sử dụng một độ chính xác 63 chữ số. Kiểu con REAL cung cấp độ chính xác 18 chữ số nằm bên phải dấu thập phân.

Kiểu dữ liệu PLS_INTEGER. Các kiểu dữ liệu PLS_INTEGER và BINARY_INTEGER giống hệt và sử dụng số học dành riêng cho hệ điều hành cho các phép tính. Chúng có thể lưu trữ các số nguyên từ 2,147,483,648 đến 2,147,483,647 dưới dạng 32 bit hoặc 4 byte. PLS_INTEGER chiếm ít không gian hơn một kiểu dữ liệu NUMBER nhiều để lưu trữ trong bộ nhớ. Nó cũng tính hiệu quả hơn miễn là các số và kết quả của phép toán nằm trong dãy số của nó. Bất kỳ phép toán tạo ra một kết quả bên ngoài dãy sẽ đưa ra một lỗi chèn số ORA-01426. Lỗi được đưa ra ngay cả khi bạn gán kết quả của phép toán vào một kiểu dữ liệu NUMBER. Sau đây là nguyên mẫu để định nghĩa một kiểu dữ liệu NVARCHAR2:

```
PLS_INTEGER
```

Bạn có thể khai báo một biến PLS_INTEGER với một giá trị rỗng hoặc được khởi tạo trong quá trình khai báo. Cú pháp cho cả hai như sau:

```
var1 PLS_INTEGER;    -- A null value requires no space.
var2 PLS_INTEGER := 11; -- An integer requires space for each character.
```

PLS_INTEGER sử dụng các thư viện toán học riêng và do đó câu lệnh khai báo không cấp phát bộ nhớ để lưu trữ biến cho đến khi một giá trị được gán. Bạn có thể test điều này bằng cách sử dụng hàm cài sẵn LENGTH ().

Hàm cài sẵn LENGTH

Hành vi này nhất quán với những gì bạn sẽ thấy khi viết các chương trình C hoặc C++. Khi một giá trị được gán, hàm cài sẵn LENGTH () trả về số ký tự chứ không phải số byte được yêu cầu để lưu trữ. Điều này có nghĩa một PLS_INTEGER với 5 hoặc 6 số dường như có một chiều dài ký tự lần lượt là 5 hoặc 6 nhưng thực ra chỉ chiếm 4 byte không gian trong cả hai trường hợp. Kết quả này dường như được liên kết với cách làm việc cơ sở dữ liệu NUMBER nơi các giá trị cột NUMBER được lưu trữ dưới dạng các mảng ký tự một chiều C. Hàm LENGTH () dường như đếm các vị trí trong tất cả kiểu dữ liệu số.

Bạn có thể khai báo một kiểu con PLS_INTEGER bằng cách sử dụng nguyên mẫu sau đây:

```
SUBTYPE subtype_name IS base_type [RANGE low_number..high_number] [NOT NULL];
```

Ghi chú

Đừng nhầm lẫn một PLS_INTEGER với một INTEGER. Cái trước sử dụng các thư viện toán học hệ điều hành trong khi cái sau là một kiểu con của kiểu cơ sở NUMBER.

Các đối tượng lớn (LOB)

Các đối tượng lớn (LOB) cung cấp bốn kiểu dữ liệu - BFILE, BLOB, CLOB, và NCLOB. BFILE là kiểu dữ liệu trỏ vào một file bên ngoài, điều này giới hạn kích cỡ tối đa của nó chỉ trong 4 gigabyte. BLOB, CLOB, và NCLOB là các kiểu được quản lý bên trong. Kích cỡ tối đa của chúng là 8 đến 128 terabyte, phụ thuộc vào giá trị tham số db_block_size.

Các cột LOB chứa một locator trỏ sang nơi dữ liệu thực sự được lưu trữ. Bạn phải truy cập một giá trị LOB trong phạm vi của một giao tác. Về cơ bản, bạn sử dụng locator làm lộ trình để đọc dữ liệu từ hoặc ghi dữ liệu sang cột LOB. Chương 8 trình bày chi tiết về cách bạn truy cập các cột LOB và các kiểu dữ liệu làm việc với LOB kể cả gói cài sẵn DBMS_LOB.

Kiểu dữ liệu BFILE

Kiểu dữ liệu BFILE là một kiểu dữ liệu chỉ đọc ngoại trừ xác lập thư mục ảo (virtual directory) và tên file cho file ngoài. Bạn sử dụng hàm BFILENAME () cài sẵn để xác lập thông tin định vị cho một cột BFILE. Trước khi bạn sử dụng hàm BFILENAME (), có một số bước thiết lập. Bạn phải tạo một thư mục vật lý trên server, lưu trữ file trong thư mục, tạo một thư mục ảo trỏ sang thư mục vật lý và cấp các quyền đọc trên thư mục đến schema vốn sở hữu table hoặc chương trình lưu trữ truy cập cột BFILE.

Bạn truy tìm descriptor (tên cột), bí danh (một thư mục ảo dẫn sang vị trí thư mục vật lý) và tên file bằng cách sử dụng thủ tục FILEGETNAME () từ gói DBMS_LOB. Tham số cơ sở dữ liệu session_max_open_files xác lập số cột BFILE mở tối đa. Chương 8 tập hợp cách những mảnh này lắp ghép lại với nhau như thế nào và cung cấp cho bạn một số đơn vị chương trình lưu trữ để đơn giản hoá tiến trình. Sau đây là nguyên mẫu để khai báo một kiểu dữ liệu BFILE

```
BFILE
```

Có một cách để định nghĩa một biến BFILE và nó luôn chứa một tham chiếu rỗng theo mặc định:

```
var1 BFILE;          -- Declare a null reference to a BFILE.
```

BFILE không thể được định nghĩa với một tham chiếu trừ phi bạn viết một wrapper cho thủ tục DBMS_LOB.FILEGETNAME () cung cấp một hàm wrapper và giải thích những giới hạn vốn đòi hỏi hàm wrapper.

Kiểu dữ liệu BLOB

Cột BLOB là một kiểu dữ liệu lớn nhị phân đọc - ghi. Các kiểu dữ liệu lớp tham gia các giao tác và có thể phục hồi. Bạn chỉ có thể đọc và ghi giữa các biến BLOB và các cột cơ sở dữ liệu trong một phạm vi giao tác. Các kiểu dữ liệu BLOB là những đối tượng và được xử lý khác với các biến vô hướng. Chúng có ba trạng thái: null, empty và populated (không rỗng). Chúng đòi hỏi khởi tạo bằng hàm empty_blob () để di chuyển từ một tham chiếu null đến một trạng thái empty, hoặc một phép gán thập lục phân trực tiếp để trở thành populated.

Các BLOB có thể lưu trữ các file nhị phân giữa 8 và 32. Thật không may bạn có thể truy cập các cột BLOB bằng cách sử dụng gói DBMS_LOB để đọc và ghi các giá trị sau khi gán một giá trị.

PL/SQL cho bạn khai báo các biến lớp cục bộ trong các khối nặc danh và khối được đặt tên. Tuy nhiên, bạn phải thiết lập một liên kết hoạt động giữa chương trình của bạn và cột BLOB lưu trữ để chèn, thêm hoặc

đọc giá trị cột. Nói chung bạn sẽ chỉ đọc hoặc lưu trữ các cụm giá trị BLOB lớn hoặc bạn có thể làm cạn kiệt các nguồn tài nguyên hệ thống.

Sau đây là nguyên mẫu để khai báo một kiểu dữ liệu BLOB:

BLOB

Chỉ có một cách để khai báo một biến BLOB với một tham chiếu rỗng mặc định:

```
var1 BLOB;          -- Declare a null reference to a BLOB.
```

Có hai cách được định nghĩa một biến BLOB trống (empty) và được tập hợp (populated):

```
var1 BLOB := empty_blob();    -- Declare an empty BLOB.
```

```
var2 BLOB := '43' || '41' || '52';    -- Declare a hexadecimal BLOB for CAR.
```

Các kiểu dữ liệu BLOB đặc biệt hữu dụng khi bạn muốn lưu trữ các file ảnh lớn, movie, hoặc các file nhị phân khác. Tiện ích của chúng phụ thuộc rất nhiều vào việc bạn viết giao diện tốt như thế nào. Chương 8 thảo luận các cách để xử lý những giao tác giữa các cột BLOB và các biến PL/SQL.

Kiểu dữ liệu CLOB

Cột CLOB là một kiểu dữ liệu lớn ký tự đọc - ghi. Các kiểu dữ liệu CLOB tham gia vào các giao tác và có thể phục hồi. Bạn chỉ có thể đọc và ghi giữa các biến CLOB và các cột cơ sở dữ liệu trong một phạm vi giao tác. Các kiểu dữ liệu CLOB là những đối tượng như BLOB và được xử lý khác với các biến vô hướng. Chúng cũng có ba trạng thái: null, empty, hoặc populated (không trống). Các CLOB đòi hỏi khởi tạo bằng hàm `empty_clob()` để di chuyển từ một tham chiếu rỗng đến một trạng thái empty, hoặc từ một phép gán ký tự trực tiếp để trở thành populated.

Các CLOB có thể lưu trữ các file ký tự giữa 8 và 32 byte. Các CLOB cũng bị cùng một giới hạn như các kiểu biến không phải Unicode. Việc cấp phát không gian tính bằng byte trong khi việc mã hoá Unicode được thực hiện trong các ký tự được định nghĩa bởi 2 hoặc 3 byte mỗi ký tự. Như với các BLOB, bạn chỉ có thể truy cập các cột CLOB bằng cách sử dụng gói `DBMS_LOB` để đọc hoặc ghi các giá trị sau khi gán một giá trị.

PL/SQL cho bạn khai báo các biến CLOB cục bộ trong các khối nặc danh và khối được đặt tên. Như với các BLOB, bạn phải thiết lập một liên kết hoạt động giữa chương trình của bạn và cột CLOB lưu trữ để chèn, thêm, hoặc đọc giá trị cột. Nói chung, bạn sẽ muốn chỉ đọc hoặc lưu trữ các cụm giá trị CLOB lớn, hoặc bạn có thể làm cạn kiệt các nguồn tài nguyên hệ thống.

Sau đây là nguyên mẫu để khai báo một kiểu dữ liệu CLOB:

```
CLOB
```

Chỉ có cách để định nghĩa một biến CLOB với một tham chiếu rỗng mặc định:

```
var1 CLOB; -- Declare a null reference to a CLOB.
```

Có hai cách để định nghĩa một biến CLOB trống và một biến CLOB được tập hợp (populated):

```
var1 CLOB := empty_clob(); -- Declare an empty CLOB.
```

```
var2 CLOB := 'CAR'; -- Declare a CLOB for CAR.
```

Kiểu dữ liệu NCLOB

Cột NCLOB là một kiểu dữ liệu lớn ký tự Unicode đọc - ghi. Các kiểu dữ liệu NCLOB tham gia vào các giao tác và có thể phục hồi. Bạn chỉ có thể đọc và ghi giữa các biến NCLOB và các cột cơ sở dữ liệu trong một phạm vi giao tác. Các kiểu dữ liệu NCLOB là những đối tượng như BLOB và CLOB và được xử lý khác với các biến vô hướng. Chúng không có ba trạng thái. Chúng có thể null, empty, hoặc populated (không phải empty). Các NCLOB đòi hỏi khởi tạo bằng cùng một hàm empty_clob () được sử dụng để khởi tạo một biến hoặc cột CLOB. Hàm empty_clob () thay đổi tham chiếu rỗng thành một trạng thái empty. Hoặc bạn có thể gán chuỗi ký tự trực tiếp để tập hợp lại biến.

Các NCLOB có thể lưu trữ các file ký tự Unicode giữa 8 và 32 byte. Các giới hạn chuỗi ký tự Unicode xác lập kích cỡ tối đa tương ứng với tập hợp ký tự cơ sở dữ liệu. Một số tập hợp ký tự sử dụng 2 hoặc 3 byte để lưu trữ các ký tự. Bạn chia kích cỡ tối đa (8 đến 32 gigabyte) cho số byte được yêu cầu trên mỗi ký tự, nghĩa là mức tối đa cho một NCLOB là 4 đến 16 gigabyte cho một tập hợp ký tự 2 byte (AL16UTF16), và 2.67 đến 8.66 gigabyte cho một tập hợp ký tự 3 byte (UTF8). Như với các BLOB và CLOB, bạn chỉ có thể truy cập các cột NCLOB bằng cách sử dụng gói DBMS_LOB để đọc và ghi các giá trị sau khi gán ban đầu một giá trị.

PL/SQL cho bạn khai báo các biến NCLOB cục bộ trong các khối nặc danh và khối được đặt tên. Như với các BLOB và CLOB, bạn phải thiết lập một liên kết hoạt động giữa chương trình của bạn và cột NCLOB lưu trữ để chèn, thêm hoặc đọc giá trị cột. Nói chung, bạn sẽ chỉ muốn đọc hoặc lưu trữ các cụm giá trị NCLOB lớn, hoặc bạn có thể làm tiêu hao các nguồn tài nguyên hệ thống.

Sau đây là nguyên mẫu để khai báo một kiểu dữ liệu NCLOB:

```
NCLOB
```


Chỉ có một cách để định nghĩa một biến NCLOB với một tham chiếu rỗng mặc định:

```
var1 NCLOB; -- Declare a null reference to a NCLOB.
```

Có hai cách để định nghĩa một biến NCLOB trống và được tập hợp lại:

```
var1 NCLOB := empty_clob();    -- Declare an empty NCLOB.
var2 NCLOB := 'CAR';          -- Declare a NCLOB for CAR.
```

Các kiểu dữ liệu tổng hợp

Có hai kiểu dữ liệu tổng hợp được khái quát hoá: các record và collection (tập hợp). Một record còn được gọi là một cấu trúc thường chứa một tập hợp các phần tử liên quan như một bảng cơ sở dữ liệu được chuẩn hoá. Các collection là các tập hợp những thứ tương tự. Những thứ có thể là các biến vô hướng, đối tượng lớn, đối tượng do người dùng định nghĩa (xem chương 8) hoặc các record.

Hai mục nhỏ tiếp theo mô tả việc khai báo các record và collection. Sách thực thi các record qua suốt sách này. Trong khi các record không phải là những đối tượng nguyên thủy, chúng là những cấu trúc cơ bản trong ngôn ngữ. Chương 7 đề cập chi tiết về các collection.

Các record

Một kiểu dữ liệu record là một cấu trúc. Một cấu trúc (structure) là một biến tổng hợp chứa một danh sách các biến mà thường có các tên và những kiểu dữ liệu khác nhau. Bạn định nghĩa một kiểu dữ liệu record một cách ngầm định hoặc tường minh. Có những giới hạn về việc sử dụng các record được tạo ngầm định. Bạn không thể sử dụng một record được tạo ngầm định trong một mảng (array). ROWTYPE cho phép bạn định nghĩa một loại record được neo một cách ngầm định. Khi bạn muốn xây dựng một record và sử dụng nó trong một record hoặc mảng khác, bạn phải xây dựng record một cách tường minh.

Sau đây là nguyên mẫu để định nghĩa tường minh một kiểu dữ liệu record:

```
TYPE type_name IS RECORD
  ( element1 sql_datatype | plsql_datatype [NOT NULL][[DEFAULT | :=] literal]
    , element2 sql_datatype | plsql_datatype [NOT NULL][[DEFAULT | :=] literal]
    , element(n+1) sql_datatype | plsql_datatype [NOT NULL][[DEFAULT | :=] literal]
```

);

sql_datatype của một phần tử trong loại record được định nghĩa tường minh có thể sử dụng một cột được neo một cách ngầm định. Bạn neo (anchor) cột sử dụng thuộc tính TYPE. plsql_datatype của một phần tử trong loại record được định nghĩa một cách tường minh có thể sử dụng một loại record được neo một cách ngầm định (sử dụng ROWTYPE). Sau khi bạn định nghĩa loại record, nó có sẵn dưới dạng một kiểu dữ liệu cục bộ. Bạn có thể khai báo một biến bằng cách sử dụng kiểu dữ liệu record trong các chương trình khối nặc danh và khối được đặt tên.

Dòng mã sau đây minh họa việc khai báo một cấu trúc được khởi tạo:

```
DECLARE
    TYPE demo_record_type IS RECORD
        ( id NUMBER DEFAULT 1
          , value VARCHAR2(10) := 'One');
    demo DEMO_RECORD_TYPE;
BEGIN
    dbms_output.put_line(['| |demo.id| |'] ['| |demo.value| |']);
END;
/
```

Khối thực thi in nội dung của record bằng cách sử dụng ký hiệu chấm. Chấm là component selector từ bảng 3.1 trước. Component selector tách các tên biến và tên phần tử. Đây còn được gọi là móc nối thuộc tính (attribute chaining) trong một số tài liệu Oracle.

Cũng có thể xếp lồng các record. Bạn truy cập tên của các record được xếp lồng bằng cách sử dụng một component selector, hoặc dấu chấm khác như được trình bày:

```
DECLARE
    TYPE full_name IS RECORD
        ( first VARCHAR2(10 CHAR) := 'John'
          , last VARCHAR2(10 CHAR) := 'Taylor');
    TYPE demo_record_type IS RECORD
        ( id NUMBER DEFAULT 1
          , contact FULL_NAME);
    demo DEMO_RECORD_TYPE;
BEGIN
    dbms_output.put_line(['| |demo.id| |']);
    dbms_output.put_line(['| |demo.contact.first| |'] ['| |demo.contact.last| |']
```

END;

/

Các record cực kỳ hữu dụng khi làm việc với các cursor và collection. Chương 4 đề cập đến các cursor bởi vì bạn cần hiểu các cấu trúc điều khiển lập lại để làm việc với chúng. Phần tiếp theo "các collection" hướng dẫn cách xây dựng các tập hợp record.

Các record có sẵn độc quyền bên trong phạm vi thực thi PL/SQL. Bạn có thể định nghĩa một hàm lưu trữ để trả về một loại record nhưng điều đó giới hạn cách sử dụng một hàm. SQL chỉ có thể truy cập các hàm lưu trữ khi chúng trở về các kiểu dữ liệu SQL. Lựa chọn cho việc trả về một record là một loại đối tượng SQL. Chương 4 đề cập đến các loại đối tượng nhưng bạn nên chú ý rằng chúng không có sẵn trong Oracle 11g Express Edition. Bạn cũng có thể trả về một loại record bằng cách sử dụng một hàm pipelined vốn chuyển đổi nó thành một bản gộp hàng đơn. Chương 6 trình bày cách thực thi các hàm pipelined.

Collections

Các collection (tập hợp) là các mảng (array) và danh sách (list). Các mảng khác với các danh sách trong đó chúng sử dụng một index được đánh số theo trình tự trong khi các danh sách sử dụng một index số không theo trình tự hoặc index chuỗi duy nhất. Các mảng là những danh sách được tập hợp dày đặc bởi vì chúng có các index được đánh số theo trình tự. Trong khi các danh sách có thể có các index số được tập hợp dày đặc. Chúng cũng có thể được tập hợp thưa. Được tập hợp thưa nghĩa là có các khoảng hở trong một dãy hoặc không theo trình tự.

Oracle hỗ trợ ba loại tập hợp. Hai loại tập hợp là các kiểu dữ liệu SQL và PL/SQL phụ thuộc vào cách bạn định nghĩa như thế nào: VARRAY và bảng xếp lồng (nested table). Loại tập hợp thứ ba là một kiểu dữ liệu chỉ PL/SQL được gọi là một mảng kết hợp (associative array). Mảng kết hợp còn được gọi là một table PL/SQL hoặc một table index-by. Các mục nhỏ mô tả cách khai báo các tập hợp của các loại VARRAY, bảng xếp lồng, và mảng kết hợp. Chương 7 đề cập chi tiết đến các tập hợp.

Kiểu dữ liệu VARRAY. Kiểu dữ liệu VARRAY giống mảng truyền thống nhất. Các phần tử có cùng một kiểu và sử dụng một index số trình tự. Điều này có nghĩa index của các biến VARRAY được tập hợp dày đặc. Bạn chọn sử dụng một VARRAY khi biết số phần tử sẽ nằm trong tập hợp trước khi khai báo biến. Như các mảng trong những ngôn ngữ lập trình khác, VARRAY không thể tăng kích cỡ sau khi đã được khai báo. Bạn nên sử dụng một bảng xếp lồng hoặc mảng kết hợp khi bạn không chắc chắn bạn có biết trước các giá trị số tối đa hay không.

Có hai nguyên mẫu cho một VARRAY bởi vì bạn có thể định nghĩa nó trong SQL hoặc PL/SQL. Bạn cũng nên chú ý rằng kiểu dữ liệu được

định nghĩa, không được khai báo. Sự khác biệt này xảy ra bởi vì VARRAY là một loại đối tượng. Các đối tượng đòi hỏi việc xây dựng tường minh. Chương 14 về các đoạn đối tượng giải thích thêm về cách bạn xây dựng các đối tượng.

Một VARRAY có ba trạng thái: *defined*, *initialized*, hoặc *allocated*. Bạn định nghĩa một VARRAY bằng cách gán cho nó một tên và một kiểu. Bạn khởi tạo một VARRAY bằng cách gọi một constructor (phương thức tạo) theo cùng một tên như định nghĩa trong kiểu dữ liệu VARRAY. Bạn cấp phát (*allocate*) không gian một cách ngầm định hoặc bằng cách gọi phương thức *EXTEND ()* trong API tập hợp được tìm thấy trong chương 7.

Sau đây là nguyên mẫu SQL để định nghĩa một VARRAY gồm các biến vô hướng:

```
CREATE OR REPLACE TYPE varray_name AS VARRAY (maximum_size)
OF sql_datatype [NOT NULL] ;
```

Nguyên mẫu sau đây định nghĩa một VARRAY của bất kỳ kiểu dữ liệu trong một khối PL/SQL:

```
TYPE varray_name IS VARRAY (maximum_size) OF [sql_datatype |
plsqli_datatype] [NOT NULL] ;
```

Cả hai định nghĩa kiểu xác định một kích cỡ cố định. Kích cỡ tối đa giới hạn số phần tử mà bạn có thể lưu trữ trong một VARRAY. Bạn cũng có thể định nghĩa các biến VARRAY sử dụng các đối tượng SQL cho người dùng định nghĩa hoặc các loại record PL/SQL. Chương 14 hướng dẫn cách xây dựng các biến VARRAY với các loại đối tượng do người dùng định nghĩa. Chương 7 hướng dẫn cách tận dụng các loại record PL/SQL trong các biến VARRAY.

Có một nguyên mẫu để khai báo một tập hợp VARRAY, và có hai nguyên mẫu để định nghĩa một tập hợp VARRAY. Bạn có thể định nghĩa một tập hợp VARRAY dưới dạng một tập hợp được khởi tạo mà không cấp phát bất kỳ bộ nhớ hoặc dưới dạng một tập hợp được khởi tạo với bộ nhớ được cấp phát. Bạn cấp phát một bộ nhớ bằng cách sử dụng một phương thức đặt dữ liệu vào tập hợp. Dòng sau đây là các nguyên mẫu được đề cập trước mà bạn sử dụng trong một khối PL/SQL:

```
var1 varray_name;
var2 varray_name := varray_name();
var3 varray_name := varray_name(value1, value2, .. ., value9, value10);
```

Gọi tên biến của kiểu VARRAY không có bất kỳ đối số tạo một biến trống nhưng được khởi tạo. Một lệnh gọi tương tự của tên biến với các đối số tạo và cấp phát các giá trị cho VARRAY. Trừ phi bạn đã xác định

ràng buộc not-null, bạn có thể cấp phát không gian bằng cách gán các giá trị rỗng (null).

Mã sau đây minh họa việc khai báo một VARRAY của biến vô hướng:

```
DECLARE
    TYPE number_varray IS VARRAY(10) OF NUMBER;
    list NUMBER_VARRAY := number_varray(1,2,3,4,5,6,7,8,NULL,NULL);
BEGIN
    FOR i IN 1..list.LIMIT LOOP
        dbms_output.put('[' || list(i) || ' ]');
    END LOOP;
    dbms_output.new_line;
END;
/
```

Chương trình in kết quả sau đây sang console:

```
[1] [2] [3] [4] [5] [6] [7] [8] [ ] [ ]
```

Nó khởi tạo 8 phần tử đầu tiên với các giá trị hai phần tử cuối cùng với các giá trị rỗng. Phần khai báo các phép không gian cho 10 phần tử bằng cách xác lập tất cả phần tử sang value, vốn có thể chứa một giá trị rỗng. Phương thức LIMIT () trả về kích cỡ tối đa; nó là một phần của Oracle Collection API và chỉ áp dụng vào các biến VARRAY. Bạn không thể sử dụng phương thức DELETE để loại bỏ một phần tử sau khi nó được định nghĩa. Các index VARRAY luôn được tập hợp dày đặc.

Kiểu dữ liệu Nested Table. Kiểu dữ liệu nested table giống như một danh sách được tạo index bằng số hoặc lớp Java. Như trong VARRAY, các phần tử cùng một loại và sử dụng một index số trình tự. Điều này có nghĩa index của các biến nested table được tập hợp dày đặc. Bạn nên sử dụng một nested table khi bạn không biết số phần tử vốn sẽ nằm trong tập hợp trước khi khai báo biến. Như các danh sách trong những ngôn ngữ lập trình khác, nested table có thể phát triển kích cỡ một khi đã được khai báo.

Có hai nguyên mẫu cho một nested table bởi vì bạn có thể định nghĩa nó trong SQL hoặc PL/SQL. Bạn cũng nên chú ý rằng kiểu dữ liệu được định nghĩa, không thể được khai báo. Sự khác biệt này xảy ra bởi vì nested table là một loại đối tượng. Các đối tượng đòi hỏi một sự xây dựng tường minh. Chương 14 về các loại đối tượng giải thích thêm xây dựng đối tượng.

Một nested table có ba trạng thái: defined, initialized, hoặc allocated. Bạn định nghĩa một nested table bằng cách gán cho nó một tên và một

kiểu. Bạn khởi tạo một nested table bằng cách gọi một phương thức tạo (constructor), vốn luôn có cùng một tên như kiểu dữ liệu nested table được định nghĩa. Bạn cấp phát không gian bằng cách tường minh hoặc bằng cách gọi phương thức EXTEND trong collection API, được mô tả trong chương 7.

Sau đây là nguyên mẫu SQL để định nghĩa một nested table gồm các biến vô hướng:

```
CREATE OR REPLACE TYPE table_name AS TABLE
OF sql_datatype [NOT NULL] ;
```

Nguyên mẫu sau đây định nghĩa một nested table của bất kỳ kiểu dữ liệu được định nghĩa trong một khối PL/SQL:

```
TYPE table_name IS TABLE OF [sql_datatype | plsql_datatype]
[NOT NULL] ;
```

Cả hai định nghĩa kiểu không xác lập một kích cỡ tối đa bởi vì không có giới hạn về bao nhiêu phần tử bạn có thể lưu trữ trong một nested table. Bạn cũng có thể định nghĩa các biến nested table sử dụng các đối tượng SQL do người dùng định nghĩa hoặc các loại record PL/SQL.

Ghi chú

DBA được xác lập giới hạn, và khi bạn vi phạm việc cấp phát không gian PGA, bạn sẽ đưa ra một ngoại lệ.

Có một nguyên mẫu để khai báo một tập hợp nested table, và có hai nguyên mẫu để định nghĩa một tập hợp nested table. Bạn có thể định nghĩa một tập hợp nested table dưới dạng một tập hợp được khởi tạo mà không cấp phát bất kỳ bộ nhớ. Hoặc dưới dạng một tập hợp được khởi tạo với bộ nhớ được cấp phát. Bạn cấp phát bộ nhớ bằng cách sử dụng một phương thức tạo vốn đặt dữ liệu vào tập hợp.

Sau đây là các nguyên mẫu đã đề cập mà bạn sẽ sử dụng trong một khối PL/SQL:

```
var1 varray_name;
var2 varray_name := varray_name();
var3 varray_name := varray_name(value1, value2, .. ., value9, value10);
```

Gọi tên biến của loại nested table không có bất kỳ đối số nào sẽ tạo ra một biến trống nhưng được khởi tạo. Một lệnh gọi tương tự của tên biến với các đối số tạo và cấp phát các giá trị cho nested table. Trừ phi bạn đã xác định ràng buộc not-null, bạn cũng có thể cấp phát không gian bằng cách gán các giá trị rỗng.

Mã sau đây minh hoạ cách bạn khai báo nested table của một biến vô hướng:

```

DECLARE
    TYPE number_table IS TABLE OF NUMBER;
    list NUMBER_TABLE := number_table(1,2,3,4,5,6,7,8);
BEGIN
    list.DELETE(2);
    FOR i IN 1..list.COUNT LOOP
        IF list.EXISTS(i) THEN
            dbms_output.put('[' || list(i) || '] ');
        END IF;
    END LOOP;
    dbms_output.new_line;
END;
/

```

Chương trình in kết quả sau đây sang console:

```
[1] [3] [4] [5] [6] [7] [8]
```

Nó khởi tạo 8 phần tử đầu tiên với các giá trị và hai phần tử cuối cùng với các giá trị rỗng (null). Phần khai báo cấp phát không gian cho 8 phần tử. Sau đó, phương thức DELETE xoá phần tử thứ hai ra khỏi danh sách nhưng nó không loại bỏ không gian được cấp phát. Việc xoá làm cho index trở nên không theo trình tự. Bạn có thể chèn lại một giá trị mới nhưng chỉ khi bạn tái sử dụng giá trị index bị xoá. Hành vi này khác với một VARRAY, nơi bạn không thể loại bỏ một phần tử một khi nó được cấp phát trong bộ nhớ.

Phương thức COUNT trả về số phần tử không gian được cấp phát trong bất kỳ loại tập hợp. Trong trường hợp này, nó vẫn toả về 7, trong chỉ 6 phần tử có các giá trị. Khối if tránh tham chiếu phần tử bị xoá bởi vì index 2 không còn hiện hữu nữa. Chương 7 trình bày một giải pháp khác để định hướng các index thưa.

Kiểu dữ liệu Associative Array. Kiểu dữ liệu associative array giống danh sách liên kết C/C++ truyền thống nhiều nhất. Bạn có thể index một mảng kết hợp (associative array) với các số hoặc các chuỗi duy nhất. Nếu bạn chọn các số, chúng không cần phải thay trình tự. Điều này có nghĩa index của một mảng được tập hợp thưa. Như một nested table, một mảng kết hợp lý tưởng khi bạn không biết số phần tử sẽ nằm trong tập hợp trước khi khai báo nó. Như các danh sách trong những ngôn ngữ lập trình khác, nested table có thể phát triển kích cỡ

sau khi đã được khai báo. Bộ nhớ cũng được cấp phát một cách ngầm định trong quá trình gán vào một mảng kết hợp.

Có một nguyên mẫu cho một mảng bởi vì bạn chỉ có thể khai báo nó trong PL/SQL. Một mảng kết hợp được khai báo như các biến vô hướng bởi vì nó không phải là một loại đối tượng nghĩa là nó không đòi hỏi việc xây dựng. Không giống như các biến vô hướng, bạn không thể định nghĩa một mảng kết hợp bởi vì mỗi lần chỉ được gán một phần tử.

Nguyên mẫu sau đây định nghĩa một mảng kết hợp của bất kỳ loại dữ liệu và sử dụng cùng một index số:

```
TYPE table_name IS TABLE OF [sql_datatype | plsql_datatype]
INDEX BY PLS_INTEGER [NOT NULL] ;
```

Định nghĩa kiểu rất tương tự như một định nghĩa nested table. Nó chỉ có một điểm khác biệt chính: nó xác định cách index được lưu giữa như thế nào. Một nguyên mẫu thay thế cho các mảng kết hợp sử dụng một chuỗi có chiều dài phổ biến dưới dạng một index:

```
TYPE table_name IS TABLE OF [sql_datatype | plsql_datatype]
INDEX BY VARCHAR2 (10) [NOT NULL] ;
```

Cả hai loại định nghĩa kiểu không xác lập một kích cỡ tối đa bởi vì không có giới hạn về bao nhiêu phần tử mà bạn có thể lưu trữ trong một mảng kết hợp. Giới hạn thực sự chỉ phối việc cấp phát không gian PGA. Bạn cũng có thể định nghĩa các biến mảng kết hợp vốn sử dụng các đối tượng SQL do người dùng định nghĩa hoặc các loại record PL/SQL. Chương 14 hướng dẫn bạn cách xây dựng các biến mảng kết hợp với các loại đối tượng do người dùng định nghĩa. Chương 7 hướng dẫn bạn cách tận dụng các loại record PL/SQL trong các biến mảng kết hợp.

Sau đây là nguyên mẫu mà bạn sử dụng trong một khối PL/SQL:

```
var1 assoc_array name;
```

Mã sau đây minh họa việc khai báo một mảng kết hợp của một biến vô hướng:

```
DECLARE
    TYPE number_table IS TABLE OF NUMBER
    INDEX BY PLS_INTEGER;
    list NUMBER_TABLE;
BEGIN
    FOR i IN 1..6 LOOP
        list(i) := i; -- Explicit assignment required for associative arrays.
```



```

END LOOP;
list.DELETE(2);
FOR i IN 1..list.COUNT LOOP
    IF list.EXISTS THEN
        dbms_output.put('|' || list(i) || '|');
    END IF;
END LOOP;
dbms_output.new_line;
END;
/

```

Chương trình in kết quả sau đây sang console:

```
[1] [3] [4] [5] [6]
```

Nó khởi tạo 6 phần tử với các giá trị bên trong khối thực thi. Các phần tử được tập hợp bằng phép gán trực tiếp vào các phần tử được tạo index của mảng kết hợp. Sau đó phương thức DELETE loại bỏ phần tử thứ hai ra khỏi danh sách. Không giống trong VARRAY và nested table, việc xoá một phần tử ra khỏi một mảng kết hợp cũng sẽ loại bỏ không gian được cấp phát. Việc xoá làm cho index trở nên không theo trình tự. Bạn có thể chèn lại một giá trị mới với giá trị index bị xoá hoặc một giá trị index mới không được sử dụng. Điều này phản ánh hành vi nested table.

Phương thức COUNT trả về số phần tử không gian được cấp phát trong bất kỳ loại tập hợp. Khối if tránh tham chiếu phần tử bị xoá bởi vì index 2 không còn hiện hữu nữa. Chương 7 trình bày một giải pháp khác cho việc định hướng các index được tập hợp thưa.

Các cursor tham chiếu hệ thống

Các cursor tham chiếu hệ thống là các pointer dẫn sang các tập hợp kết quả trong các vùng query work. Một vùng query work là một vùng bộ nhớ (được gọi là một vùng ngữ cảnh) trong Oracle 11g Database Process Global Area (PGA). Vùng query work (làm việc truy vấn) chứa thông tin về query. Bạn sẽ tìm thấy các hàng được trả về bởi một query, số hàng được xử lý bởi query, và một pointer dẫn sang query được phân tích cú pháp trong vùng làm việc query. Vùng làm việc query thường trú trong Oracle Shared Pool.

Bạn sử dụng các cursor tham chiếu khi bạn truy vấn dữ liệu trong một chương trình và xử lý nó trong một chương trình khác đặc biệt khi hai chương trình nằm trong các ngôn ngữ lập trình khác nhau. Bạn có tùy chọn thực thi một cursor tham chiếu bằng hai cách: một là được định

kiểu mạnh và một là được định kiểu yếu. Các cursor tham chiếu là một kiểu dữ liệu chỉ PL/SQL. Bạn có thể định nghĩa chúng trong các khối nặc danh hoặc được đặt tên. Chúng hữu dụng nhất khi bạn định nghĩa chúng trong các tham số package bởi vì các tiến trình có thể chia sẻ chúng.

Có một nguyên mẫu nhưng việc bạn chọn thực thi cursor như thế nào quyết định nó được định kiểu mạnh hay yếu. Nguyên mẫu là

```
TYPE reference_cursor_name IS REF CURSOR
```

```
[RETURN catalog_object_nameROWTYPE] ;
```

Bạn tạo một cursor tham chiếu được định kiểu yếu bằng cách định nghĩa nó mà không có một kiểu trả về. Một cursor tham chiếu được định kiểu mạnh có một kiểu trả về được định nghĩa. Theo quy tắc chung bạn nên sử dụng các cursor tham chiếu được định kiểu mạnh khi bạn cần neo (anchor) một cursor tham chiếu sang một đối tượng catalog. Các cursor tham chiếu được định kiểu yếu lý tưởng khi query trả về một thứ gì đó ngoại trừ một đối tượng catalog. Một cursor tham chiếu được định kiểu yếu chung chung đã được định nghĩa là SYS_REFCURSOR, và nó có sẵn bất cứ nơi nào trong môi trường lập trình PL/SQL.

Sức mạnh của một cursor tham chiếu trở nên quan trọng hơn khi bạn sử dụng chúng bên trong các đơn vị chương trình lưu trữ. Bạn cũng sử dụng các cursor tham chiếu trong các chương trình khối nặc danh và gán chúng vào một biến môi trường tham chiếu SQL*Plus.

Bạn định nghĩa một biến môi trường tham chiếu SQL*Plus bằng cách định nghĩa một biến và nhấn ENTER. Các câu lệnh SQL*Plus không đòi hỏi một dấu chấm phẩy hoặc dấu gạch chéo tiến (/) để chạy. Mã sau đây tạo một cursor tham chiếu SQL*Plus được định kiểu yếu:

```
SQL> VARIABLE refcur REFCURSOR
```

Chương trình sau đây định nghĩa và khai báo một cursor tham chiếu trước khi mở nó một cách tường minh và gán các giá trị của nó vào một biến cấp session bên ngoài:

```
DECLARE
TYPE weakly_typed IS REF CURSOR;
quick WEAKLY_TYPED;
BEGIN
  OPEN quick FOR
    SELECT  item_title
           , COUNT(*)
    FROM    item
```

```
HAVING (COUNT(*) > 2)
GROUP BY item_title;
:refcur := quick;
END;
/
```

Cursor tham chiếu chung chung SYS_REFCURSOR có thể thay thế loại cursor tham chiếu định nghĩa cục bộ. Bạn có thể truy vấn biến cấp session để xem nội dung của cursor tham chiếu với dòng sau đây:

```
SELECT :refcur
FROM dual;
```

Query trả về kết quả sau đây miễn là bạn đã chạy các script hạt giống được tìm thấy trong phần giới thiệu của sách:

```
:REFCUR
-----
CURSOR STATEMENT : 1
CURSOR STATEMENT : 1
ITEM_TITLE                                               COUNT (*)
-----
Harry Potter and the Chamber of Secrets                3
Harry Potter: Goblet of Fire                            3
Die Another Day                                          3
The Lord of the Rings - Two Towers                     3
The Lord of the Rings - Fellowship of the Ring         3
Chronicles of Narnia - The Lion, the Witch and the Wardrobe 50
Harry Potter and the Goblet of Fire                    3
Pirates of the Caribbean - The Curse of the Black Pearl 3
Pirates of the Caribbean                               4
The Lord of the Rings - The Return of the King         3

10 rows selected.
```

Chương 6 trình bày cách sử dụng một cursor tham chiếu bên trong các hàm và thủ tục. Các cursor tham chiếu là những kiểu dữ liệu cực kỳ hữu dụng khi bạn muốn chuyển pointer vùng làm việc query đến một chương trình bên ngoài. Bạn có thể chuyển đến chương trình bên ngoài bằng cách sử dụng các thư viện Oracle Call Interface 8 (OCI8).

Phạm vi biến

Như được thảo luận, PL/SQL là một ngôn ngữ lập trình kết khối. Các đơn vị chương trình có thể là các khối được đặt tên hoặc không được đặt tên. Mỗi khối lập trình thiết lập phạm vi chương trình riêng của nó.

Phạm vi chương trình bao gồm một danh sách các biến (định danh) vốn có thể chứa dữ liệu. Một chương trình bao gồm các biến được định nghĩa cả trong header (chỉ áp dụng cho các đơn vị chương trình được đặt tên) và trong khối khai báo. Chúng xem xét cục bộ đối với khối lập trình.

Các khối nặc danh xếp lồng là ngoại lệ cho quy tắc phạm vi. Chúng có sự truy cập đến các định danh khối PL/SQL chứa của chúng. Điều này đúng cho dù khối chứa là nặc danh hoặc được đặt tên. Hình 3.6 minh họa sự truy cập phạm vi của các khối và của các chương trình nặc danh xếp lồng.

Bạn có thể vô ý ghi đè sự truy cập phạm vi đến các khối chứa bằng cách tái sử dụng một định danh trong một khối xếp lồng. Hành vi này được minh họa trong chương trình sau đây:

```

DECLARE
    current_block VARCHAR2(10) := 'Outer';
    outer_block VARCHAR2(10) := 'Outer';
BEGIN
    dbms_output.put_line('[current_block][1 | current_block | 1]');
    DECLARE
        current_block VARCHAR2(10) := 'Inner';
    BEGIN
        dbms_output.put_line('[current_block][1 | current_block | 1]');
        dbms_output.put_line('[outer_block][1 | outer_block | 1]');
    END;
    dbms_output.put_line('[current_block][1 | current_block | 1]');
END;
/

```

Các định danh `current_block` và `outer_block` (các biến cục bộ) được khai báo trong một chương trình khối nặc danh ngoài với giá trị `outer`. Định danh `current_block` được khai báo trong khối xếp lồng hoặc khối trong với một giá trị `inner`, trong khi định danh `outer_block` không được khai báo trong khối xếp lồng.

Chương trình kết xuất kết quả sau đây:

```

[current_block][Outer]
[current_block][Inner]
[outer_block] [Outer]
[current_block][Outer]

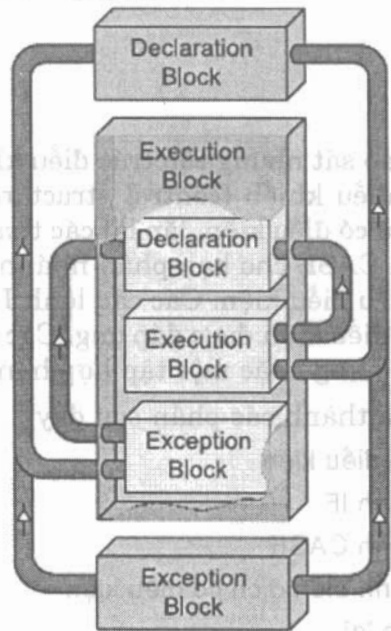
```

Khối xếp lồng ghi đè phạm vi của khối chứa bằng cách định nghĩa cùng một định danh. Khối chứa không thấy định danh `current_block` được khai báo bên trong. Đây là một sắc thái của phạm vi khá tinh vi trong PL/SQL.

Một khía cạnh khác của phạm vi là chuyển các giá trị từ một chương trình đến một khối được đặt tên. Điều này được thực hiện qua danh sách tham số chính thức vốn tạo nên chữ ký của các hàm và thủ tục. Chương 6 giải thích những khối được đặt tên này nhận các giá trị dưới dạng các tham số thực sự và các giá trị trả về như thế nào.

Tóm tắt

Chương này đã giải thích các dấu tách (delimiter); cách bạn định nghĩa, truy cập và gán các giá trị vào các biến như thế nào; các đơn vị chương trình khối nặc danh và khối được đặt tên, các kiểu biến và cách phạm vi biến làm việc trong những chương trình PL/SQL như thế nào.



Hình 3.6 Sơ đồ tham chiếu phạm vi PL/SQL

CHƯƠNG 4

CÁC CẤU TRÚC điều khiển

Chương này khảo sát những cấu trúc điều khiển trong PL/SQL. Các cấu trúc điều khiển (control structure) cho phép đưa ra những lựa chọn có điều kiện, lặp lại các thao tác và truy cập dữ liệu. Các câu lệnh IF và CASE cho bạn phân nhánh sự thực thi chương trình theo một hoặc nhiều điều kiện. Các câu lệnh Loop cho bạn lặp lại hành vi cho đến khi các điều kiện được đáp ứng. Các cursor cho bạn truy cập dữ liệu mỗi lần một hàng hoặc một tập hợp hàng.

Chương này được chia thành các phần sau đây:

- Các câu lệnh có điều kiện
 - Các câu lệnh IF
 - Các câu lệnh CASE
 - Các câu lệnh biên dịch có điều kiện
- Các câu lệnh lặp lại
 - Các câu lệnh vòng lặp Simple
 - Các câu lệnh vòng lặp While
 - Các câu lệnh vòng lặp FOR
- Các cấu trúc cursor
 - Các cursor ngầm định
 - Các cursor tường minh

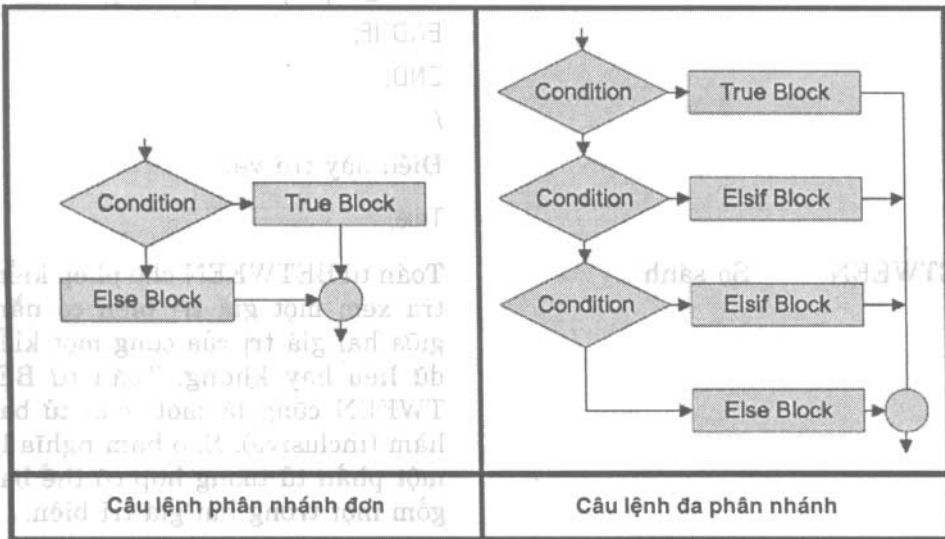
- Các câu lệnh Bulk
 - Các COLLECT BULK
 - Các câu lệnh vòng lặp FORALL

Các câu lệnh có điều kiện

Có ba loại câu lệnh có điều kiện trong các ngôn ngữ lập trình: các câu lệnh phân nhánh đơn, các câu lệnh đa phân nhánh không có fall-through, và các câu lệnh đa phân nhánh có fall-through. Fall through nghĩa là xử lý tất cả điều kiện tiếp theo sau khi tìm thấy một câu lệnh CASE tương hợp. Các câu lệnh phân nhánh đơn là những câu lệnh if-then-else. Các câu lệnh đa phân nhánh không có fall-through là những câu lệnh if-then-elsif-then-else, và với fall-through là các câu lệnh case. Hình 4.1 minh họa dòng chảy logic của hai câu lệnh có điều kiện đầu tiên. Câu lệnh thứ ba không được hiển thị bởi vì PL/SQL không hỗ trợ fall-through, và PL/SQL thực thi câu lệnh case như một câu lệnh if-then-elsif-then-else.

Ghi chú

PL/SQL sử dụng từ dành riêng ELSIF thay cho hai từ riêng biệt else if. Đây là một sự thừa kế từ các ngôn ngữ lập trình Pascal và Ada.



Hình 4.1 Các dòng chảy logic câu lệnh phân nhánh

Các hình thời trong hình 4.1 là các cây quyết định (decision tree). Các cây quyết định tượng trưng cho việc phân nhánh mã xảy ra do những phép toán so sánh. Khi một biểu thức có thể trả về một giá trị null, bạn

nên đặt biểu thức trong một NVL () cài sẵn và cung cấp một giá trị Boolean mặc định tương minh.

PL/SQL hỗ trợ các symbol từ vựng, các tập hợp symbol, và các định danh dưới dạng các toán tử so sánh hợp lệ. Bảng 4.1 trình bày một danh sách và định nghĩa về các toán tử so sánh symbol. Bảng 4.1 mở rộng danh sách toán tử so sánh bằng cách cung cấp các toán tử so sánh vốn là các định danh (identifier). Các định danh như vậy là những từ dành riêng hoặc từ khoá.

Bảng 4.1 Các toán tử so sánh

AND	So sánh	<p>Toán tử AND cho phép kết hợp hai phép so sánh thành một phép so sánh. Toán tử này làm cho câu lệnh tổ hợp chỉ true (đúng) khi cả hai câu lệnh riêng lẻ là true. Bạn cũng có thể sử dụng toán tử AND với toán tử BETWEEN để gắn kết các giá trị dãy trên và dãy dưới.</p> <pre>BEGIN IF 1 = 1 AND 2 = 2 THEN dbms_output.put_line('True.');</pre> <p>END IF;</p> <p>END;</p> <p>/</p> <p>Điều này trả về:</p> <p>True.</p>
BETWEEN	So sánh	<p>Toán tử BETWEEN cho phép kiểm tra xem một giá trị biến có nằm giữa hai giá trị của cùng một kiểu dữ liệu hay không. Toán tử BETWEEN cũng là một toán tử bao hàm (inclusive). Bao hàm nghĩa là một phần tử tương hợp có thể bao gồm một trong hai giá trị biên.</p> <pre>BEGIN IF 1 BETWEEN 1 AND 3 THEN dbms_output.put_line('In the range.');</pre> <p>END IF;</p> <p>END;</p> <p>/</p>

Điều này trả về kết quả sau đây:

In the range.

IN So sánh

Toán tử IN cho bạn kiểm tra xem một giá trị biến có nằm trong một tập hợp giá trị được tách bằng dấu phẩy hay không.

BEGIN

IF 1 IN (1,2,3) THEN

dbms_output.put_line('In the set.');

END IF;

END;

/

Điều này trả về kết quả sau đây:

In the set.

IS EMPTY So sánh

Toán tử IS EMPTY cho phép bạn kiểm tra xem một biến tập hợp VARRAY hoặc NESTED TABLE có trống hay không. Trống (empty) nghĩa là tập hợp đã được tạo không có bất kỳ phần tử mặc định. Điều này có nghĩa không gian đã không được cấp phát cho SGA cho các phần tử trong tập hợp. Khi không gian phần tử không được cấp phát, phép so sánh IS EMPTY trả về true, và nó trả về false khi tối thiểu một phần tử được cấp phát. Bạn đưa ra một ngoại lệ PLS-00306 khi tập hợp đã không được khởi tạo thông qua việc tạo tường minh. Chương 12 giải thích cách tạo các tập hợp (collection) như thế nào. Điều này chỉ có tác dụng với các tập hợp của các kiểu dữ liệu SQL vô hướng.

DECLARE

TYPE list IS TABLE OF INTEGER;

a LIST := list();

BEGIN

IF a IS EMPTY THEN

```

dbms_output.put_line("a is empty.");
END IF;
END;
/

```

Điều này trả về kết quả sau đây:

'a' is empty.

IS NULL

So sánh

Toán tử IS NULL cho phép kiểm tra xem một giá trị biến có rỗng hay không. NVL () cài sẵn có thể cho phép gán vào bất kỳ Boolean hoặc biểu thức một giá trị true hoặc false tương minh.

```

DECLARE
var BOOLEAN;
BEGIN
IF var IS NULL THEN
dbms_output.put_line('It is null.');
```

END IF;

END;

/

Điều này trả về kết quả sau đây:

It is null.

Toán tử IS A SET cho phép kiểm tra xem một biến là một biến tập hợp VARRAY hay NESTED TABLE, miễn là một instance của biến đã được tạo. Nó trả về true khi kiểu dữ liệu biến là một VARRAY hoặc NESTED TABLE và biến đã được tạo. Được tạo nghĩa là một instance của tập hợp đã được tạo có hoặc không có các thành viên. Chương 14 trình bày thêm chi tiết khái niệm về việc tạo một biến.

IS A SET

So sánh

Toán tử so sánh IS A SET trả về false khi kiểu dữ liệu biến là một VARRAY hoặc NESTED TABLE và biến không được tạo. Một mảng kết hợp (một loại tập hợp khác) không

phải là một đối tượng tập hợp và khi bạn cố kiểm tra xem nó có phải là một tập hợp hay không, bạn đưa ra một ngoại lệ FLS-00306. Tương tự các loại biến khác đưa ra cùng một ngoại lệ PLS-00306. Chương 7 giải thích cách tạo các tập hợp. Toán tử so sánh này chỉ làm việc với các tập hợp vốn sử dụng các kiểu dữ liệu SQL vô hướng. Nếu bạn quen "A" trong toán tử IS A SET, nó đưa ra một ngoại lệ PLS-00103 định danh bị biến dạng.

PLS-00103 exception.

```
DECLARE
    TYPE list IS TABLE OF INTEGER;
    a LIST := list();
BEGIN
    IF a IS A SET THEN
        dbms_output.put_line("'a' is a set.');
```

END IF;

END;

/

Điều này trả về kết quả sau đây:

'a' is a set.

Toán tử LIKE cho phép bạn kiểm tra xem một giá trị biến có phải là một phần của một giá trị khác hay không. Việc so sánh có thể được thực hiện với dấu gạch dưới từ điển SQL cho một wildcard (ký tự đại diện) ký tự đơn, hoặc wildcard nhiều ký tự. Giá trị từ vựng % bên trong một chuỗi không tương đương với việc sử dụng nó làm một chỉ báo thuộc tính trong PL/SQL.

```
BEGIN
    IF 'Str%' LIKE 'String' THEN
        dbms_output.put_line('Match');
```

END IF;

LIKE

So sánh

MEMBER OF	So sánh	<pre> END; / Điều này trả về kết quả sau đây: Match. MEMBER OF là một toán tử so sánh logic. Nó cho bạn tìm xem một phần tử có phải là một thành viên của một tập hợp hay không. Nó chỉ làm việc với các tập hợp của các kiểu dữ liệu SQL vô hướng. Nó trả về true khi phần tử hiện hữu trong tập hợp và false khi nó không hiện hữu. DECLARE TYPE list IS TABLE OF NUMBER; n VARCHAR2(10) := 'One'; a LIST := list('One', 'Two', 'Three'); BEGIN IF n MEMBER OF a THEN dbms_output.put_line('"n" is member. '); END IF; END; / </pre>
NOT	So sánh	<p>Khi phần tử toán hạng trái rỗng, toán tử trả về false. Điều này có nghĩa bạn luôn nên kiểm tra tìm một giá trị trước khi sử dụng toán tử so sánh này. Nó in kết quả sau đây:</p> <pre> 'n' is member NOT là một toán tử phủ định logic và cho phép bạn kiểm tra tìm trạng thái đối nghịch với một trạng thái Boolean của một biểu thức miễn là nó không rỗng (null). BEGIN IF NOT FALSE THEN dbms_output.put_line('True. '); </pre>

```
END IF;
```

```
END;
```

```
/
```

Khi biểu thức hoặc giá trị là null, NOT không thay đổi gì cả. Không có giá trị trái ngược với null và một sự phủ định logic của null cũng là một null. Điều này trả về kết quả sau đây bởi vì FALSE là một trực kiện Boolean và bởi vì TRUE là một thứ duy nhất không false khi bạn loại trừ các giá trị null:

```
True.
```

OR

So sánh

Toán tử OR cho bạn kết hợp hai phép so sánh thành một phép so sánh. Toán tử này làm cho câu lệnh kết hợp trở thành true khi một câu lệnh này hoặc câu lệnh kia là true. PL/SQL sử dụng việc lượng giá ngắn mạch nghĩa là nó ngưng lượng giá một phép so sánh kết hợp khi bất kỳ giá trị nào là false.

```
BEGIN
```

```
IF 1 = 1 OR 1 = 2 THEN
```

```
  dbms_output.put_line('True.');
```

```
END IF;
```

```
END;
```

```
/
```

Điều này trả về kết quả sau đây do một trong hai câu lệnh là true:

```
True.
```

SUBMULTISET So sánh

Toán tử SUBMULTISET cho phép kiểm tra xem một tập hợp VARRAY hoặc NESTED TABLE có phải là một tập hợp con của một kiểu dữ liệu đối xứng hay không. Nó trả về true nếu một số trong tất cả phần tử trong tập hợp trái được tìm thấy trong tập hợp phải. Bạn nên chú ý rằng toán tử này không kiểm tra tìm một tập hợp con thích hợp vốn

ít hơn tập hợp đầy đủ hoặc tập hợp đồng nhất thức một phần tử.

```
DECLARE
```

```
    TYPE list IS TAELE OF INTEGER;
```

```
    a LIST := list(1,2 3);
```

```
    b LIST := list(1,2,3,4);
```

```
BEGIN
```

```
    IF a SUBMULTISET b THEN
```

```
        dbms_output.put_line('Subset.');
```

```
    END IF;
```

```
END,
```

```
/
```

Điều này trả về kết quả sau đây

Valid subset.

Bạn cũng cần biết thứ tự phép toán cho các toán tử so sánh. Bảng 4.2 liệt kê thứ tự phép toán của chúng. Bạn có thể ghi đề thứ tự phép toán bằng cách đặt các biểu thức thứ cấp trong các dấu ngoặc đơn. PL/SQL so sánh bất kỳ biểu thức bên trong các dấu ngoặc đơn dưới dạng toàn bộ kết quả. PL/SQL áp dụng bất kỳ toán tử so sánh còn lại trong một biểu thức bằng thứ tự phép toán của chúng.

Các biểu thức so sánh đơn trả về một true, false, hoặc null. Cả false và null không phải true khi bạn lượng giá việc một biểu thức là true hay không. Tương tự, cả true và null không phải false khi bạn lượng giá một biểu thức dưới dạng false. Một biểu thức null không bao giờ là true hoặc false. Bảng 4.3 phản ánh các kết quả có thể có trong một bảng chân trị.

Bảng 4.2 Thứ tự của các phép toán

Thứ tự	Toán tử	Định nghĩa
1	**	Phép mũ hoá (xem bảng 3.1)
2	+, -	Đồng nhất thức và sự phủ định
3	*, /	Phép nhân và phép chia
4	+, -, !	Phép cộng, phép trừ, và phép ghép
5	=, <, >, <=, >=, <>, !=, ~=, ^=, BETWEEN, IN, IS, NULL, LIKE	Phép so sánh
6	AND	Phép hội

7	NOT	Phép phủ định logic
8	OR	Phép loại trừ

Nhiều biểu thức so sánh đòi hỏi các bảng chân trị hai phía: một bảng cho toán tử hội, AND, và một bảng khác cho toán tử bao hàm OR. Toán tử hội tạo các biểu thức nơi bạn phân giải tổ hợp của hai biểu thức trong đó cả hai là true. Toàn bộ câu lệnh không true khi một là false hoặc null. Bảng 4.4 phản ánh những kết quả có thể có của độ chính xác hội - khi các biểu thức X và Y là true, false hoặc null.

Nhiều biểu thức so sánh cũng đòi hỏi một bảng chân trị hai phía để kiểm tra toán tử bao hàm làm việc như thế nào. Bao hàm (inclusive) là nơi hai điều là true khi điều này hoặc điều kia là true, nhưng do các biểu thức null toàn bộ câu lệnh có thể là true, false, hoặc null. Bảng 4.5 phản ánh các kết quả có thể có của độ chính xác bao hàm - khi các biểu thức X hoặc Y là true, false hoặc null.

Bảng 4.3 Bảng chân trị một biến

Giá trị X	Biểu thức	Kết quả	Biểu thức phủ định	Kết quả
TRUE	X là TRUE	TRUE	X là NOT TRUE	FALSE
FALSE	X là TRUE	FALSE	X là NOT TRUE	TRUE
NULL	X là TRUE	NULL	X là NOT TRUE	TRUE

Bảng 4.4 X và Y là TRUE, FALSE hoặc NULL

X và Y	Y là TRUE	Y là FALSE	Y là NULL
X là TRUE	TRUE	FALSE	NULL
X là FALSE	FALSE	FALSE	FALSE
X là NULL	NULL	FALSE	NULL

Các bảng chân trị giúp hoạch định cách bạn sẽ phát triển logic phân nhánh trong các câu lệnh IF và CASE như thế nào. Kết quả logic tương tự này mở rộng thành ba biểu thức trở lên nhưng chúng không kết xuất trong các bảng 2 chiều.

Phần này đã cung cấp chi tiết để hỗ trợ các phần con phân nhánh. Các phần con kiểm tra các câu lệnh phân nhánh đơn và đa phân nhánh vốn sử dụng các câu lệnh IF và các câu lệnh đa phân nhánh vốn sử dụng các câu lệnh case đơn giản và tìm kiếm. Các phần con được kết nhóm bằng các câu lệnh IF và CASE.

Các câu lệnh IF

Câu lệnh IF hỗ trợ các câu lệnh phân nhánh đơn và đa phân nhánh. Các câu lệnh IF là các khối. Chúng bắt đầu với một định danh bắt đầu, hoặc từ dành riêng, và kết thúc bằng một định danh kết thúc và một dấu chấm phẩy. Tất cả câu lệnh khối đòi hỏi tối thiểu một câu lệnh tương tự như các khối nặc danh hoặc khối được đặt tên.

Các câu lệnh IF lượng giá một điều kiện. Một điều kiện có thể là bất kỳ biểu thức so sánh hoặc tập hợp biểu thức so sánh vốn lượng giá thành một true hoặc false logic. Bạn có thể so sánh hai trực kiện hoặc biến cùng một kiểu. Các biến thực sự có thể có các kiểu dữ liệu khác nhau miễn là chúng ngầm định hoặc bạn chuyển đổi một trong hai kiểu để khớp với kiểu kia một cách tường minh. Một biến Boolean có thể thay thế một phép toán so sánh. Bạn cũng có thể so sánh kết quả của hai lệnh gọi hàm như bạn thường so sánh hai biến hoặc một biến và một lệnh gọi hàm miễn là nó trả về một biến Boolean. Các toán tử so sánh hợp lệ được trình bày trong bảng 3.1 trong chương trước cũng như trong bảng 4.1 trước đó.

Bảng 4.5 X hoặc Y là TRUE, FALSE, hoặc NULL

X hoặc Y	Y là TRUE	Y là FALSE	Y là NULL
X là TRUE	TRUE	TRUE	TRUE
X là FALSE	TRUE	FALSE	NULL
X là NULL	TRUE	NULL	NULL

Các lệnh gọi hàm dưới dạng các biểu thức

Khi bạn gọi một hàm, bạn cung cấp các giá trị hoặc biến và trả về một kết quả. Nếu hàm trả về một chuỗi chiều dài khả biến, bạn có thể gọi nó là một biểu thức chuỗi bởi vì nó tạo ra một chuỗi như là một kết quả. Kết quả giống như một trực kiện chuỗi (được đề cập trong chương 3). Hoặc các định nghĩa hàm có thể trả về bất kỳ kiểu dữ liệu biến vô hướng khác và chúng trở thành các biểu thức vốn tạo ra các giá trị của các kiểu dữ liệu đó.

Ví dụ sau đây so sánh một biến và biểu thức (hoặc lệnh gọi hàm):

```

DECLARE
    one_thing VARCHAR2(5) := 'Three';
    FUNCTION ordinal (n NUMBER) RETURN VARCHAR2 IS
        TYPE ordinal_type IS TABLE OF VARCHAR2(5);
        ordinal ORDINAL_TYPE := ordinal_type('One','Two','Three','Four');
    BEGIN
        RETURN ordinal(n);
    
```



```

        END;
    BEGIN
        IF one_thing = ordinal(3) THEN
            dbms_output.put_line([' | | ordinal(3) | | ']);
        END IF;
    END;
/

```

Chương trình mẫu so sánh một giá trị biến và kết quả trả về của lệnh gọi hàm hoặc giá trị trực kiện và biểu thức. Chúng được tìm thấy lại bằng nhau. Chương trình in kết quả sau đây miễn là SQL*Plus SERVEROUTPUT được xác lập sang on:

Three

Giá trị trả về của một lệnh gọi hàm là một biểu thức hoặc một giá trị runtime vốn có thể được so sánh với nội dung của một biến, giá trị trực kiện hoặc một lệnh gọi hàm khác.

Các câu lệnh if-then-else

Câu lệnh if-then-else là một câu lệnh phân nhánh đơn. Nó lượng giá một điều kiện và sau đó chạy mã ngay tức thì sau khi điều kiện được đáp ứng. Nguyên mẫu cho một câu lệnh if-then-else là

```

IF [NOT] {comparison_expression | boolean_value} [[AND | OR]
    {comparison_expression | boolean_value}] THEN
    true_execution_block;
[ELSE
    false_execution_block;]
END IF;

```

Bạn sử dụng NOT tùy chọn (toán tử phủ định logic) được kiểm tra tìm một kết quả so sánh sai (false). Trong khi chỉ có một mệnh đề [AND | OR] trong nguyên mẫu câu lệnh IF, không có giới hạn về bao nhiêu điều kiện mà bạn lượng giá. Khối ELSE thì tùy chọn. Các câu lệnh IF không có một khối ELSE chỉ thực thi mã khi một điều kiện được đáp ứng.

Ở dạng đơn giản nhất đây là một câu lệnh if-then. Dòng mã sau đây minh họa một câu lệnh if-then so sánh hai trực kiện số:

```

BEGIN
    IF 1 = 1 THEN
        dbms_output.put_line('Condition met!');
    
```

```

END IF;
END;
/

```

Bạn nên chú ý rằng các dấu ngoặc đơn xung quanh câu lệnh so sánh không được bắt buộc. Đây là một sự tiện lợi so với một số ngôn ngữ lập trình khác nơi chúng được bắt buộc như PHP. Logic tương đương sử dụng một biến Boolean thay vì phép toán so sánh là

```

DECLARE
    equal BOOLEAN NOT NULL := TRUE;
BEGIN
    IF equal THEN
        dbms_output.put_line('Condition met!');
    END IF;
END;
/

```

Khi bạn lượng giá một biến Boolean hoặc biểu thức trả về một giá trị rỗng, câu lệnh IF trả về một giá trị false. Bạn nên dự tính trước các hành vi run-time có thể dẫn đến một giá trị rỗng và sử dụng NVL () cài sẵn ở nơi có thể để tránh các kết quả bất ngờ. Hành vi mặc định thì tốt miễn là bạn muốn chương trình xem một giá trị rỗng là false.

Miễn là bạn xác lập SERVEROUTPUT sang ON trong SQL*Plus, một trong hai khối nặc danh này phân giải phép so sánh là true và in

```
Condition met!
```

Phân nhánh ra, bạn có thể tạo một câu lệnh if-then-else như

```

BEGIN
    IF 1 = 2 THEN
        dbms_output.put_line('Condition met!');
    ELSE
        dbms_output.put_line('Condition not met!');
    END IF;
END;
/

```

Khối nặc danh phân giải phép so sánh dưới dạng false và in câu lệnh khối else:

```
Condition not met!
```

Bạn có thể hỗ trợ các biến cho các trực kiện cho những ví dụ này hoặc lệnh gọi hàm vốn trả về các kiểu dữ liệu tương hợp hoặc có thể chuyển đổi để so sánh. Một hàm đơn trả về một kiểu dữ liệu BOOLEAN cũng làm việc thay cho ví dụ Boolean.

Các câu lệnh if-then-elsif-then-else

Câu lệnh if-then-elsif-then-else là một câu lệnh đa phân nhánh. Nó lượng giá một loạt các điều kiện và sau đó nó chạy mã ngay tức thì sau điều kiện đầu tiên được đáp ứng thành công. Nó thoát khối sau khi xử lý và bỏ qua bất kỳ phép lượng giá thành công tiếp theo. Nguyên mẫu cho một câu lệnh if-then-elsif-then-else là

```
IF [NOT] {comparison_expression | boolean_value} [[AND | OR]
    {comparison_expression | boolean_value}] THEN
    true_if_execution_block;
[ELSIF [NOT] {comparison_expression | boolean_value} [[AND | OR]
    {comparison_expression | boolean_value}] THEN
    true_elsif_execution_block;
[ELSE
    all_false_execution_block;}
END IF;
```

Bạn sử dụng toán tử NOT tùy chọn để kiểm tra nhằm tìm các phép so sánh sai (false). Trong khi chỉ có một mệnh đề [AND | OR] trong các câu lệnh IF và ELSIF, không có giới hạn về bao nhiêu điều kiện mà bạn lượng giá. Khối ELSE thì tùy chọn. Một câu lệnh if-then-elsif-then-else không có một khối ELSE chỉ thực thi mã cho một điều kiện được đáp ứng.

Dòng mã sau đây minh họa một câu lệnh if-then-elsif-then-else nơi hai phép so sánh đầu tiên là true và phép so sánh thứ ba là false:

```
DECLARE
    equal BOOLEAN NOT NULL := TRUE;
BEGIN
    IF 1 = 1 THEN
        dbms_output.put_line('Condition one met!');
    ELSIF equal THEN
        dbms_output.put_line('Condition two met!');
    ELSIF 1 = 2 THEN
        dbms_output.put_line('Condition three met!');
    END IF;
```

```
END;
```

```
/
```

Khối nặc danh phân giải phép so sánh thứ nhất là true và in

```
Condition one met!
```

Như được đề cập, câu lệnh if-then-elsif-then-else thoát sau khi phép so sánh thứ nhất được tìm thấy là true. Điều kiện ELSE mặc định chỉ chạy khi không có điều kiện nào được đáp ứng.

Các câu lệnh CASE

Có hai loại câu lệnh CASE trong PL/SQL. Cả hai định nghĩa một selector. Một selector là một biến, hàm, hoặc biểu thức mà câu lệnh CASE cố gắng tương hợp trong các khối WHEN. Selector đứng ngay sau từ dành riêng CASE. Nếu bạn không cung cấp một selector, PL/SQL thêm một Boolean true dưới dạng selector. Bạn có thể sử dụng bất kỳ kiểu dữ liệu PL/SQL làm một selector ngoại trừ một BLOB, BFILE, hoặc kiểu tổ hợp. Chương 3 xác định các kiểu tổ hợp (composite) dưới dạng các record, tập hợp (collection), và các kiểu đối tượng do người dùng định nghĩa.

Nguyên mẫu câu lệnh CASE chung là

```
CASE [ TRUE | [selector_variable]]
  WHEN [criterion1 | expression1] THEN
    criterion1_statements;
  WHEN [criterion2 | expression2] THEN
    criterion2_statements;
  WHEN [criterion(n+1) | expression(n+1)] THEN
    criterion(n+1)_statements;
  ELSE
    block_statements;
END CASE;
```

Các selector câu lệnh CASE đơn giản là các biến sử dụng một hàm mà nó trả về các kiểu dữ liệu hợp lệ ngoại trừ các kiểu Boolean. Các selector câu lệnh CASE tìm kiếm là các biến hoặc hàm Boolean trả về một biến Boolean. Selector mặc định là một giá trị Boolean true. Một câu lệnh CASE tìm kiếm có thể bỏ qua selector khi tìm một biểu thức true.

Như câu lệnh IF, các câu lệnh CASE có một mệnh đề ELSE. Mệnh đề ELSE làm việc như trong câu lệnh IF với một đặc tính. Bạn không thể bỏ qua khối ELSE, nếu không bạn sẽ đưa ra một lỗi CASE_NOT_FOUND hoặc PLS-06592 khi selector không được tìm thấy. PL/SQL bao gồm

điều kiện ELSE mặc định này khi bạn không cung cấp một selector và một sự thực thi run-time không tương hợp với một khối WHEN.

Các lệnh CASE là các khối. Chúng bắt đầu với một định danh bắt đầu hoặc từ dành riêng và kết thúc bằng một định danh kết thúc và một dấu chấm phẩy. Tất cả khối câu lệnh đòi hỏi tối thiểu một câu lệnh tương tự như các khối nặc danh hoặc các khối được đặt tên. Các câu lệnh CASE đòi hỏi tối thiểu một câu lệnh trong mỗi khối WHEN và khối ELSE.

Như một câu lệnh if-then-elsif-then-else, các câu lệnh CASE lượng giá các khối WHEN bằng cách kiểm tra trình tự để tìm một phần tử tương hợp trên selector. Khối WHEN đầu tiên tương hợp với selector chạy và thoát khối CASE. Không có sẵn hành vi fall-through trong PL/SQL. Khối ELSE chỉ chạy khi khối WHEN không tương hợp với selector.

Các câu lệnh CASE đơn giản

Câu lệnh CASE đơn giản xác lập một selector vốn là bất kỳ kiểu dữ liệu PL/SQL ngoại trừ một BLOB, BFILE, hoặc kiểu tổng hợp. Nguyên mẫu cho một câu lệnh CASE đơn giản là

```

CASE selector_variable
  WHEN criterion1 THEN
    criterion1_statements;
  WHEN criterion2 THEN
    criterion2_statements;
  WHEN criterion(n+1) THEN
    criterion(n+1)_statements;
  ELSE
    block_statements;
END CASE;
```

Các câu lệnh CASE đơn giản đòi hỏi bạn cung cấp một selector. Bạn có thể thêm nhiều khối WHEN hơn những gì được trình bày, nhưng các khả năng càng nhiều thì loại giải pháp này càng kém hiệu quả. Có một giải pháp để quản lý khi bạn thường có 10 lựa chọn trở xuống. Khả năng duy trì giảm đi khi danh sách các khối WHEN tăng lên.

Ví dụ sau đây sử dụng một kiểu dữ liệu NUMBER làm selector:

```

DECLARE
  selector NUMBER := 0;
BEGIN
  CASE selector
```

```

WHEN 0 THEN
    dbms_output.put_line('Case 0!');
WHEN 1 THEN
    dbms_output.put_line('Case 1!');
ELSE
    dbms_output.put_line('No match!');
END CASE;
END;
/

```

Khối nặc danh phân giải phép so sánh đầu tiên là true bởi vì selector chứa một giá trị 0. Sau đó nó in

Case 0 !

Do đó, khối WHEN đầu tiên tương hợp với giá trị selector. Câu lệnh CASE ngưng lượng giá và trả về khối WHEN tương hợp trước khi thoát câu lệnh. Bạn có thể thay thế các kiểu dữ liệu PL/SQL khác cho giá trị selector. Các kiểu CHAR, NCHAR, và VARCHAR2 là một số lựa chọn có thể có.

Các câu lệnh CASE tìm kiếm

Selector được xác lập ngầm định cho một câu lệnh CASE tìm kiếm trừ phi bạn muốn tìm kiếm một điều kiện false. Bạn phải cung cấp một cách tương minh một false selector. Đôi khi một giá trị selector CASE tìm kiếm động (dynamic) dựa vào một số logic run-time. Khi đúng như vậy, bạn có thể thay thế một hàm trả về một biến Boolean, nghĩa là bạn xác lập động selector. Câu lệnh CASE tìm kiếm chỉ sử dụng một selector Boolean hoặc biểu thức so sánh.

Nguyên mẫu cho một câu lệnh CASE tìm kiếm là

```

CASE [(TRUE | FALSE)]
    WHEN [criterion1 | expression1] THEN
        criterion1_statements;
    WHEN [criterion1 | expression1] THEN
        criterion2_statements;
    WHEN [criterion(n+1) | expression(n+1)] THEN
        criterion(n+1)_statements;
    ELSE
        block_statements;
END CASE;

```

Như với câu lệnh CASE đơn giản, bạn có thể thêm nhiều khối WHEN hơn những gì được trình bày, nhưng càng nhiều khả năng thì loại giải pháp này càng kém hiệu quả. Câu lệnh CASE tìm kiếm sau đây kiểm tra các biểu thức so sánh:

```
BEGIN
  CASE
    WHEN 1 = 2 THEN
      dbms_output.put_line('Case [1 = 2]');
    WHEN 2 = 2 THEN
      dbms_output.put_line('Case [2 = 2]');
    ELSE
      dbms_output.put_line('No match');
    END CASE;
END;
/
```

Khối nặc danh phân giải phép so sánh đầu tiên là true bởi vì giá trị mặc định của selector là true và do đó là phép so sánh của hai sau đó nó in

```
Case [2 = 2]
```

Nếu câu lệnh CASE tìm kiếm một điều kiện false, selector tương hợp với khối WHEN đầu tiên và in 2 bằng 2. Bạn cũng có thể sử dụng một biểu thức so sánh làm selector.

Các câu lệnh biên dịch có điều kiện

Bắt đầu với Oracle 10g Release 2, bạn có thể sử dụng việc biên dịch có điều kiện (conditional compilation). Sự biên dịch có điều kiện cho phép đưa vào logic gỡ rối hoặc logic chuyên dụng mà chỉ chạy khi các biến cấp session được xác lập. Lệnh sau đây được xác lập một biến thời gian biên dịch PL/SQL DEBUG bằng 1:

```
ALTER SESSION SET PL/SQL_CCFLAGS = 'debug:1' ;
```

Lệnh này xác lập một biến thời gian biên dịch PL/SQL DEBUG bằng 1. Bạn nên chú ý rằng cờ thời gian biên dịch không nhạy kiểu chữ. Bạn cũng có thể xác lập các biến thời gian biên dịch sang true hoặc false để chúng hành động như các biến Boolean. Khi bạn muốn xác lập nhiều cờ biên dịch có điều kiện, bạn cần sử dụng cú pháp sau đây:

```
ALTER SESSION SET PL/SQL_CCFLAGS = 'name1:value1 [, name (n+1) : value (n+1) ]' ;
```

Các tham số biên dịch có điều kiện được lưu trữ dưới dạng các cặp tên và giá trị trong tham số cơ sở dữ liệu PL/SQL_CCFLAG. Chương trình sau đây sử dụng các chỉ lệnh \$IF, \$THEN, \$ELSE, \$ELSIF; \$ERROR, và \$END vốn tạo một khối mã biên dịch có điều kiện:

```
BEGIN
    $IF $$DEBUG = 1 $THEN
        dbms_output.put_line('Debug Level 1 Enabled. ');
    $END
END;
/
```

Các khối mã có điều kiện khác với các khối mã if-then-else chuẩn. Đáng chú ý nhất chỉ lệnh \$END đóng khối thay vì muốn END IF và một dấu chấm phẩy. Chỉ lệnh \$END kết thúc một câu lệnh có điều kiện. Một END IF đóng một khối mã IF. Các quy tắc cú pháp đòi hỏi việc đóng các khối kết thúc bằng một dấu chấm phẩy hoặc dấu kết thúc câu lệnh. Các dấu kết thúc câu lệnh (statement terminator) không phải là các đơn vị từ vựng điều kiện và sự xuất hiện của chúng không có một câu lệnh mã trước kích khởi sẽ gây ra lỗi thời gian biên dịch.

Ký hiệu \$\$ biểu thị một biến thời gian biên dịch có điều kiện PL/SQL. Câu lệnh ALTER SESSION cho bạn xác lập các biến thời gian biên dịch có điều kiện. Bạn xác lập chúng trong biến session PL/SQL_CCFLAGS. Một hoặc nhiều biến được xác lập trong PL/SQL_CCFLAGS. Tất cả biến là các hằng cho đến khi session kết thúc hoặc chúng được thay thế. Bạn thay thế những biến này bằng cách sử dụng câu lệnh ALTER SESSION. Tất cả biến thời gian biên dịch có điều kiện trước ngừng hiện hữu khi bạn xác lập lại biến session PL/SQL_CCFLAGS.

Các quy tắc chi phối việc biên dịch có điều kiện được xác lập bởi bộ phân tích cú pháp (parser) SQL. Bạn không thể sử dụng việc biên dịch có điều kiện trong các loại đối tượng SQL. Giới hạn này cũng áp dụng cho các nested table và VARRAY (các bảng vô hướng). Việc biên dịch các điều kiện khác nhau trong các hàm và thủ tục. Hành vi thay đổi cho dù hàm hoặc thủ tục có một danh sách tham số hình thức hay không. Bạn có thể sử dụng việc biên dịch có điều kiện sau dấu ngoặc đơn mở của một danh sách tham số hình thức như

```
CREATE OR REPLACE FUNCTION conditional_type
( magic_number $IF $$DEBUG = 1 $THEN SIMPLE_NUMBER $ELSE NUMBER
$END )
RETURN NUMBER IS
BEGIN
```



```

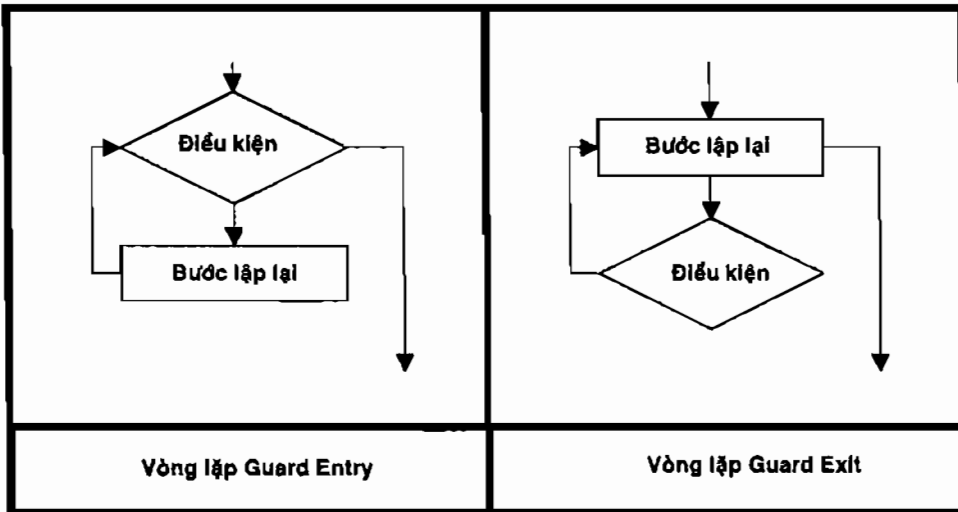
RETURN magic_number;
END;
/
    
```

Hoặc, bạn có thể sử dụng chúng sau từ khoá AS hoặc IS trong các hàm hoặc thủ tục không tham số. Chúng cũng có thể được sử dụng cả bên trong danh sách tham số hình thức và sau AS hoặc IS trong các hàm hoặc thủ tục tham số.

Việc biên dịch các điều kiện chỉ xảy ra sau từ khoá BEGIN trong các trigger và các đơn vị chương trình khối nặc danh. Chú ý rằng bạn không thể bao bọc một placeholder, hoặc biến liên kết bên trong một khối biên dịch có điều kiện.

Các câu lệnh lặp lại

Các câu lệnh lặp lại (iterative statement) là những khối cho phép lặp lại một câu lệnh hoặc tập hợp câu lệnh. Có hai loại câu lệnh lặp lại. Một câu lệnh lặp lại bảo vệ đường đi vào vòng lặp trước khi chạy các câu lệnh có thể lặp lại. Câu lệnh kia bảo vệ việc thoát ra. Một câu lệnh lặp lại chỉ bảo vệ việc thoát ra sẽ bảo đảm khối mã của nó luôn chạy một lần và được gọi là vòng lặp repeat until. Hình 4.2 minh hoạ logic thực thi cho hai loại câu lệnh lặp lại.



Hình 4.2 Các dòng chảy logic của câu lệnh lặp lại

Ngôn ngữ PL/SQL hỗ trợ các vòng lặp simple (đơn giản), FOR, FORALL, và WHILE. Nó không hỗ trợ chính thức một khối vòng lặp repeat until. Bạn có thể sử dụng câu lệnh vòng lặp đơn giản để mô phỏng hành vi của một repeat until loop. Các vòng lặp thường làm việc với các cursor. Các cursor là các batch query (các mẫu truy vấn hàng loạt) theo

từng hàng từ cơ sở dữ liệu và chúng được đề cập trong phần ngay sau các câu lệnh lặp lại.

Các câu lệnh vòng lặp đơn giản

Các vòng lặp đơn giản (simple loop) là các cấu trúc khối tương minh. Một vòng lặp đơn giản bắt đầu và kết thúc bằng từ dành riêng LOOP. Chúng đòi hỏi bạn quản lý index vòng lặp và tiêu chuẩn thoát (exit). Các vòng lặp đơn giản thường được sử dụng nơi các giải pháp dễ dàng không đòi hỏi sự phù hợp. Các giải pháp dễ dàng thường là câu lệnh vòng lặp FOR phổ biến bởi vì nó quản lý index vòng lặp và tiêu chuẩn thoát cho bạn.

Có hai nguyên mẫu cho một vòng lặp đơn giản. Chúng khác ở điểm một cái thoát ở phần trên cùng của vòng lặp và cái kia thoát ở cuối vòng lặp. Các exit (thoát) cần thiết trừ phi bạn muốn một vòng lặp vô hạn không quá thường xuyên. Các loop exit là những biểu thức so sánh. Một vòng lặp guard on entry có một câu lệnh phân nhánh trước tiên. Vòng lặp thoát khi biểu thức không còn được đáp ứng nữa. Một vòng lặp guard on exit cũng có một câu lệnh phân nhánh nhưng nó là bước cuối cùng trong khối vòng lặp. Một vòng lặp guard on exit thoát khi tiêu chuẩn exit được đáp ứng. Câu lệnh EXIT ngay tức thì dừng việc thực thi mã và thoát câu lệnh vòng lặp. Các ví dụ sau đây minh họa kỹ thuật này cho các vòng lặp guard on entry và guard on exit:

Vòng lặp Guard on Entry

```
LOOP
  [counter_management_statements;]
  IF NOT entry_condition THEN
    EXIT;
  END IF;
  repeating_statements;
END LOOP;
```

Vòng lặp Guard on Exit

```
LOOP
  [counter_management_statements;]
  repeating_statements;
  IF exit_condition THEN
    EXIT;
  END IF;
END LOOP;
```

PL/SQL đơn giản hoá việc viết một câu lệnh exit bằng cách cung cấp câu lệnh EXIT WHEN. Câu lệnh EXIT WHEN loại bỏ nhu cầu viết một câu lệnh IF xung quanh câu lệnh EXIT. Các ví dụ sau đây minh họa kỹ thuật này cho các vòng lặp guard on entry và guard on exit:

Vòng lặp Guard on Entry

```
LOOP
  [counter_management_statements;]
  EXIT WHEN NOT entry_condition;
  repeating_statements;
END LOOP;
```

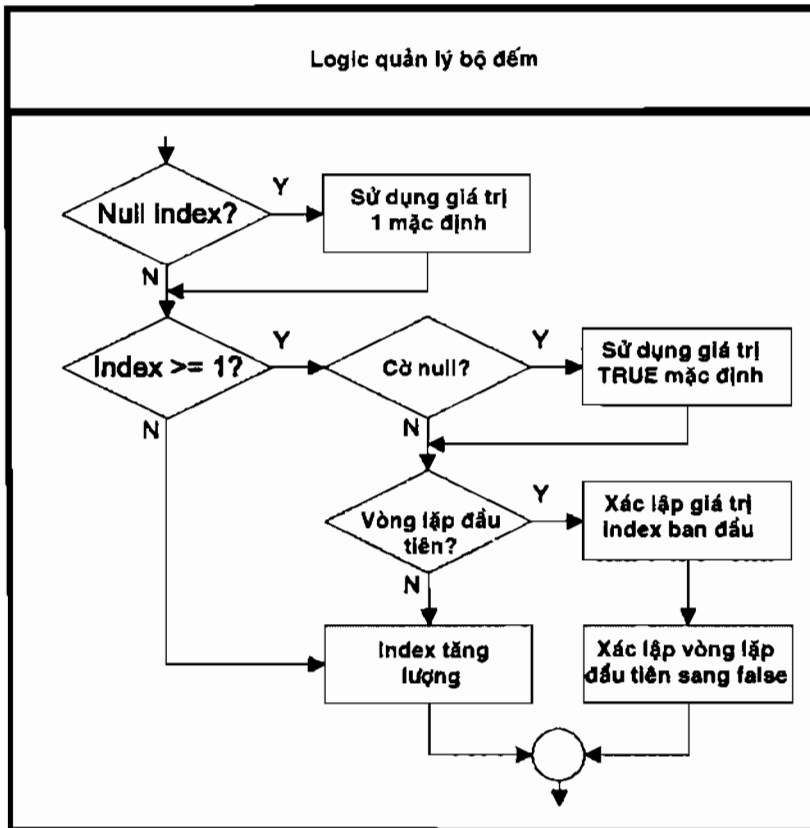
Vòng lặp Guard on Exit

```
LOOP
  [counter_management_statements;]
  repeating_statements;
  EXIT WHEN exit_condition;
END LOOP;
```

Câu lệnh CONTINUE trong Oracle 11g đòi hỏi bạn phải đặt các câu lệnh index vòng lặp ở phần trên cùng của vòng lặp. Một câu lệnh CON-

TINUE dừng thực thi trong một sự lặp lại vòng lặp và trả quyền điều khiển trở về phần trên cùng của vòng lặp. Bây giờ bạn có thể tạo một vòng lặp vô hạn giữa phần bắt đầu của vòng lặp và câu lệnh CONTINUE. Điều này có thể xảy ra bởi vì câu lệnh CONTINUE bỏ qua logic tăng hoặc giảm trừ phi nó ở phần trên cùng của vòng lặp. Điều này tương tự đối với câu lệnh CONTINUE.

Hình 4.3 minh họa một phương pháp quản lý index vòng lặp. Nó đưa ra một số giả định. Giả định thứ nhất là có thể bạn muốn cất và dán logic vào các thành phần mã khác nhau. Giả định thứ hai là có thể bạn quên khởi tạo các biến cần thiết. Tất cả những gì bạn cần làm là thực thi hai biến với các tên nhất quán để tái sử dụng phương pháp này.



Hình 4.3 Logic quản lý bộ đếm

Khối nặc danh sau đây minh họa một vòng lặp đơn giản guard on entry và thực thi logic quản lý bộ đếm từ hình 4.3:

```
DECLARE
    counter NUMBER;
    first BOOLEAN;
```

```

BEGIN
    LOOP
        -- Loop index management.
        IF NVL(counter,1) >= 1 THEN
            IF NOT NVL(first,TRUE) THEN
                counter := counter + 1;
            ELSE
                counter := 1;
                first := FALSE;
            END IF;
        END IF;
        -- Exit management.
        EXIT WHEN NOT counter < 3;
        dbms_output.put_line('Iteration [' || counter || ']');
    END LOOP;
END;
/

```

Vòng lặp đơn giản này tạo kết quả sau đây bởi vì nó bảo vệ việc đi vào sau khi chạy hai lần:

Iteration [1]

Iteration [2]

Vòng lặp đơn giản guard on exit là vòng lặp repeat until. Nó luôn chạy trước khi nó kiểm tra tiêu chuẩn. Các kết quả khác nhau giữa một vòng lặp guard on entry và vòng lặp guard on exit.

Một khối nặc danh minh họa một vòng lặp đơn giản guard on exit:

```

DECLARE
    counter NUMBER;
    first BOOLEAN;
BEGIN
    LOOP
        -- Loop index management.
        IF NVL(counter,1) >= 1 THEN
            IF NOT NVL(first,TRUE) THEN
                counter := counter + 1;
            ELSE

```

```

        counter := 1;
        first := FALSE;
    END IF;
END IF;
    dbms_output.put_line('Iteration [' || counter || ']');
-- Exit management.
EXIT WHEN NOT counter < 3;
END LOOP;
END;
/

```

Chương trình này tạo ba dòng kết quả bởi vì nó bảo vệ việc thoát ra sau ba lần thực thi

```

Iteration [1]
Iteration [2]
Iteration [3]

```

Kết quả xác nhận những gì bạn biết rằng việc kiểm tra đường vào không được tiến hành trước khi thực thi các câu lệnh lặp lại. Bạn có thể thay đổi giá trị gia tăng của index vòng lặp trong một câu lệnh vòng lặp đơn giản bằng cách thay đổi giá trị trực tiếp 1. Cả vòng lặp đơn giản và vòng lặp WHILE cũng cho bạn kiểm soát khoảng tăng lượng. Chúng cũng cho bạn giảm lượng các index vòng lặp.

Bạn không thể giảm lượng các giá trị index bằng cách sử dụng các vòng lặp FOR và FORALL. Các vòng lặp FOR và FORALL cũng không giao cho bạn nhiệm vụ quản lý index vòng lặp bởi vì index vòng lặp được thực thi một cách ngầm định và nằm bên ngoài phạm vi lặp trình có thể truy cập.

Có thể bỏ qua một giá trị index trong Oracle 11g bằng cách sử dụng câu lệnh CONTINUE mới. Câu lệnh CONTINUE báo hiệu một sự kết thúc tức thì một sự lặp lại vòng lặp và quay trở về câu lệnh đầu tiên trong vòng lặp.

Chú ý

Các câu lệnh CONTINUE và CONTINUE WHEN là những tính năng mới của Oracle 11g.

Khởi nặc danh sau đây minh họa cách thực thi một câu lệnh CONTINUE trong một vòng lặp đơn giản:

```

DECLARE
    counter NUMBER;
    first BOOLEAN;
BEGIN
    LOOP
        -- Loop index management.
        IF NVL(counter,1) >= 1 THEN
            IF NOT NVL(first,TRUE) THEN
                counter := counter + 1;
            ELSE
                counter := 1;
                first := FALSE;
            END IF;
        END IF;
        -- Exit management.
        EXIT WHEN NOT counter < 3;
        IF counter = 2 THEN
            CONTINUE;
        ELSE
            dbms_output.put_line('Index [' || counter || ']');
        END IF;
    END LOOP;
END;
/

```

Phiên bản này của chương trình chỉ in giá trị index đầu tiên trước khi chương trình thoát. Chương trình in index ban đầu 1, tăng index vòng lặp lên thành 2, bỏ qua câu lệnh in, tăng index vòng lặp lên 3, và sau đó thoát vòng lặp, không đáp ứng điều kiện guard on entry. Bạn có thể đơn giản hoá mã bằng cách thay thế tổ hợp của một khối IF và câu lệnh CONTINUE bằng câu lệnh CONTINUE WHEN. Dòng mã sau đây minh hoạ cách bạn thay thế nó như thế nào:

```

CONTINUE WHEN counter = 2;

dbms_output.put_line('Index [' || counter || '] . ');

```

Câu lệnh print trước đó nằm trong khối ELSE. CONTINUE WHEN loại bỏ nhu cầu về khối IF.

Chương trình in kết quả này sang console sau hai bước đi qua vòng lặp:

```
Iteration [1]
```

Vòng lặp đơn giản trở nên mạnh hơn khi được kết hợp với các thuộc tính cursor. Điều này được thảo luận trong phần sau "Các cấu trúc cursor" trong chương này.

Các câu lệnh vòng lặp FOR

Vòng lặp FOR là một vòng lặp ưa thích của nhiều nhà phát triển bởi vì nó mạnh và dễ sử dụng. Một vòng lặp FOR quản lý index vòng lặp và việc thoát cho bạn bởi vì nó là một phần của định nghĩa câu lệnh. Có hai loại câu lệnh vòng lặp FOR. Một là câu lệnh vòng lặp FOR dãy và một là câu lệnh vòng lặp FOR cursor.

Các câu lệnh vòng lặp FOR dãy

Một câu lệnh vòng lặp FOR dãy lý tưởng khi bạn biết điểm bắt đầu và điểm kết thúc và dãy (range) có thể được tượng trưng trong các số nguyên. Bạn cũng có thể sử dụng một câu lệnh vòng lặp FOR để định hướng nội dung của một mảng kết hợp (associative array) bằng cách truyền số phần tử trong đó. Một ví dụ về việc định hướng một mảng kết hợp sử dụng một index chuỗi được cung cấp trong chương 7.

Nguyên mẫu cho một câu lệnh vòng lặp FOR dãy là

```
FOR range_index IN range_bottom..range_top LOOP
    Repeating_statements;
END LOOP;
```

Range index có thể là bất kỳ định danh mà bạn thích hơn. Như khi viết cho các vòng lặp trong những ngôn ngữ khác, nhiều nhà phát triển sử dụng i làm tên biến. Sau đó, sử dụng j, k, l.. làm các tên biến khi xếp lồng các vòng lặp. Index dãy cho một vòng lặp FOR dãy là một PLS_INTEGER. Bạn xác lập giá trị bắt đầu khi bạn xác lập đáy của dãy (bottom of the range), và giá trị kết thúc khi bạn xác lập đỉnh của dãy (top of the range). Nó tăng lên 1 và bạn không thể thay đổi điều đó.

Chương trình khối nặc danh sau đây minh họa một câu lệnh vòng lặp FOR:

```
BEGIN
    FOR i IN 1..3 LOOP
        dbms_output.put_line('Iteration [' || i || ']');
    END LOOP;
```

```
END;
```

```
/
```

Mã này in

```
Iteration [1]
```

```
Iteration [2]
```

```
Iteration [3]
```

Giá trị biến index dãy được in trong các dấu ngoặc vuông. Bạn nên chú ý các giới hạn dãy có tính bao hàm, không có tính loại trừ. Một dãy loại trừ sẽ loại trừ 1 và 3.

Không có câu lệnh exit trong ví dụ bởi vì một câu lệnh này không được bắt buộc. Câu lệnh exit được đặt một cách ngầm định tại phần trên cùng của vòng lặp. Logic điều kiện kiểm tra xem index dãy có lớn hơn phần trên cùng của dãy hay không và nó thoát khi điều kiện đó không được đáp ứng. Điều này có nghĩa nếu bạn đảo ngược đáy và đỉnh của dãy, vòng lặp sẽ thoát trước khi xử lý bất kỳ câu lệnh bởi vì nó tìm thấy 3 không nhỏ hơn 1. Do đó, một câu lệnh vòng lặp FOR dãy là một câu lệnh vòng lặp guard on entry.

Các câu lệnh vòng lặp FOR cursor

Một câu lệnh vòng lặp FOR cursor lý tưởng khi bạn truy vấn một table hoặc view cơ sở dữ liệu. Bạn không thực sự biết nó sẽ trả về bao nhiêu hàng.

Phần này sử dụng một cursor ngầm định và phần sau "Các cấu trúc cursor" trình bày các vòng lặp FOR cursor với các cursor tường minh. Một cursor ngầm định là một câu lệnh SELECT được định nghĩa là một phần của câu lệnh vòng lặp FOR cursor. Một cursor tường minh được định nghĩa trong khối khai báo.

Nguyên mẫu cho một câu lệnh vòng lặp FOR cursor là

```
FOR cursor_index IN {cursor_name[{actual_parameters}] | (select_statement)}
LOOP repeating_statements;
END LOOP;
```

Cursor index có thể là bất kỳ định danh mà bạn thích hơn. Như khi viết cho các vòng lặp trong những ngôn ngữ khác, nhiều nhà phát triển sử dụng i làm một tên biến. Sau đó, họ sử dụng j, k, L.. làm các tên biến để xếp lồng các vòng lặp. Index cursor cho một vòng lặp FOR cursor là một pointer dẫn sang một tập hợp kết quả trong một vùng làm việc query. Một vùng làm việc query là một vùng bộ nhớ (được gọi là một vùng ngữ cảnh) trong Oracle 11g Database Process Global Area (PGA). Vùng làm việc query chứa thông tin về query. Bạn sẽ tìm thấy các hàm được trả về bởi một query, số hàng được xử lý bởi query, và một pointer

dẫn sang query được phân tích cú pháp trong vùng làm việc query. Vùng làm việc query thường trú trong Oracle Shared Pool.

Mã mẫu trình bày cách thực thi một vòng lặp FOR cursor ngầm định. Nó phụ thuộc vào việc bạn đã chạy mã seeding hay chưa và trả về tên của các bộ phim Harry Potter trong cơ sở dữ liệu mẫu của hàng cho thuê video. Sau đây là ví dụ:

```
BEGIN
    FOR i IN (SELECT          COUNT(*) AS on_hand
              ,              item_title
              ,              item_rating
              FROM            item
              WHERE item_title LIKE 'Harry Potter%'
              AND             item_rating_agency = 'MPAA'
              GROUP BY item_title
              ,            item_rating) LOOP
        dbms_output.put(' | | i.on_hand | | ');
        dbms_output.put(i.item_title | | ');
        dbms_output.put_line([' | | i.item_rating | | ']);
    END LOOP;
END;
/
```

Cursor index trở sang hàng, và component selector (dấu chấm) liên kết row pointer với tên cột hoặc bí danh được gán bởi cursor ngầm định. Điều này in kết quả sau đây từ danh sách kiểm kê:

```
Harry Potter and the Sorcerer's Stone [PG]
(3) Harry Potter and the Goblet of Fire [PG-13]
(3) Harry Potter and the Chamber of Secrets [PG]
(2) Harry Potter and the Prisoner of Azkaban [PG]
(1) Harry Potter and the Order of the Phoenix [PG-13]
```

Cũng không có câu lệnh exit trong ví dụ, bởi vì một câu lệnh như vậy không được yêu cầu. Câu lệnh exit được đặt một cách ngầm định ở phần trên cùng của vòng lặp. Điều kiện exit kiểm tra xem tất cả hàng đã được đọc hay chưa. Nó thoát khi không có thêm hàng nào để đọc.

Các cursor tường minh có một số điểm khác biệt rõ rệt và một số điểm khác biệt tinh vi. Phần sau "Các cấu trúc cursor" đề cập đến các cursor tường minh trong các câu lệnh vòng lặp FOR cursor.

Các câu lệnh vòng lặp WHILE

Các vòng lặp WHILE là các cấu trúc khối tương minh như các vòng lặp đơn giản (simple loop). Một vòng lặp WHILE bắt đầu và kết thúc bằng một từ dành riêng LOOP. Như các vòng lặp đơn giản, các vòng lặp WHILE đòi hỏi bạn quản lý cả index vòng lặp và tiêu chuẩn exit. Vòng lặp WHILE là một vòng lặp guard on entry và có thể loại trừ một index vòng lặp bởi vì điều kiện vào kiểm tra một biểu thức hoặc biến Boolean khác một cách tường minh.

Nguyên mẫu cho vòng lặp WHILE là

```

WHILE entry_condition LOOP
    [counter_management_statements;]
    repeating_statements;
END LOOP;

```

Ví dụ sau đây thực thi một vòng lặp WHILE. Vòng lặp WHILE sử dụng một giá trị index vòng lặp làm cửa của nó trên tiêu chuẩn entry:

```

DECLARE
    counter NUMBER := 1;
BEGIN
    WHILE (counter < 3) LOOP
        dbms_output.put_line('Index [' || counter || ']');
        IF counter >= 1 THEN
            counter := counter + 1;
        END IF;
    END LOOP;
END;
/

```

Nó in kết quả sau đây:

Index [1].

Index [2].

Hai điều khác nhau giữa một vòng lặp đơn giản và vòng lặp WHILE. Biến bộ đếm gate on entry phải được khởi tạo nếu không bạn không thể đi vào vòng lặp. Sự thay đổi này loại bỏ việc kiểm tra tìm một biến bộ đếm khởi tạo. Sự khác biệt thứ hai là bạn không còn kiểm tra tìm lần lặp lại đầu tiên với lần lặp lại tiếp theo qua vòng lặp. Điều này được loại bỏ bởi vì bạn không còn cần bảo vệ việc khởi tạo bộ đếm nữa.

Bạn nên chú ý rằng việc quản lý index vòng lặp là câu lệnh cuối cùng trong một vòng lặp WHILE. Nó cuối cùng bởi vì tiêu chuẩn exit là thứ đầu tiên được lượng giá ở phần trên cùng của vòng lặp. Điều này tạo ra một thách thức logic cho việc sử dụng một câu lệnh CONTINUE trong một vòng lặp WHILE bởi vì nó có thể bỏ qua logic index tăng lượng hoặc giảm lượng và tạo một vòng lặp vô hạn. Một câu lệnh GOTO và một nhãn khối thực sự là một giải pháp tốt nhất cho vấn đề này được trình bày bởi câu lệnh CONTINUE.

Đoạn mã sau đây hướng dẫn bạn cách sử dụng điều khiển có trình tự với câu lệnh GOTO và một nhãn khối:

```

DECLARE
    counter NUMBER := 1;
BEGIN
    WHILE (counter < 3) LOOP
        IF counter = 2 THEN
            GOTO loopindex;
        ELSE
            dbms_output.put_line('Index [' || counter || ']');
        END IF;
        << loopindex >>
        IF counter >= 1 THEN
            counter := counter + 1;
        END IF;
    END LOOP;
END;
/

```

Câu lệnh GOTO chuyển sự thực thi sang nhãn khối cho logic tăng lượng hoặc giảm lượng. Sau khi chạy logic index vòng lặp, quyền kiểm soát dịch chuyển sang đầu vòng lặp. Giải pháp này bảo đảm rằng mỗi sự lặp lại qua một vòng lặp WHILE tăng lượng hoặc giảm lượng index vòng lặp. Nó cũng tránh tạo một vòng lặp vô hạn ngăn mạch.

Các cấu trúc Cursor

Các cấu trúc cursor là các kết quả trả về từ những câu lệnh SQL SELECT. Trong PL/SQL, bạn có thể xử lý các câu lệnh SELECT theo từng hàng hoặc dưới dạng các câu lệnh hàng loạt. Phần này đề cập cách làm việc với các cursor xử lý câu lệnh theo từng hàng. Có hai loại cursor: ngầm định và tường minh. Bạn tạo một cursor tường minh khi bạn định

nghĩa một cursor bên trong một khối khai báo. Các câu lệnh DML bên trong bất kỳ khối thực thi hoặc khối ngoại lệ là các cursor ngầm định. Những câu lệnh này bao gồm các câu lệnh INSERT, UPDATE, và DELETE. Bạn cũng tạo các cursor ngầm định bất cứ khi nào bạn sử dụng một câu lệnh SELECT với các mệnh đề INTO hoặc BULK COLLECT INTO, hoặc bạn nhúng một câu lệnh SELECT bên trong một câu lệnh vòng lặp FOR cursor.

Sự cân bằng của phần này thảo luận các cursor ngầm định và tường minh một cách riêng biệt. Các cursor ngầm định đứng trước tiên sau đó là các cursor tường minh. Việc xử lý hàng loạt được đề cập trong phần tiếp theo.

Các cursor ngầm định

Mọi câu lệnh SQL trong một khối PL/SQL thực sự là một cursor ngầm định. Bạn có thể thấy bao nhiêu hàng được thay đổi bởi bất kỳ câu lệnh sử dụng thuộc tính ROWCOUNT sau một câu lệnh Data Manipulation Language (DML). Các câu lệnh INSERT, UPDATE và DELETE là những câu lệnh DML. Bạn cũng có thể đếm số hàng được trả về bởi một câu lệnh hoặc query SELECT.

Ví dụ sau đây minh họa một thuộc tính cursor ROWCOUNT bằng cách sử dụng một cursor ngầm định một hàng trên giả table DUAL:

```
DECLARE
    n NUMBER;
BEGIN
    SELECT 1 INTO n FROM dual;
    dbms_output.put_line('Selected [' || SQL%ROWCOUNT || ']');
END;
/
```

Từ dành riêng SQL trước thuộc tính cursor ROWCOUNT tượng trưng cho bất kỳ cursor ngầm định. Nó cũng áp dụng cho các cursor xử lý hàng loạt khi được tương hợp với một thuộc tính cursor hàng loạt. PL/SQL quản lý các cursor ngầm định và giới hạn việc truy cập đến những thuộc tính của chúng. Bảng 4.6 liệt kê các thuộc tính cursor ngầm định có sẵn.

Có ba loại cursor ngầm định. Một là cursor tập hợp hàng loạt ngầm định được đề cập trong phần sau "các câu lệnh bulk" của chương. Hai cursor ngầm định còn lại là chủ đề của phần này. Chúng là các cursor ngầm định một hàng và nhiều hàng sử dụng các câu lệnh SELECT.

Bảng 4.6 Các thuộc tính cursor ngầm định

Thuộc tính	Định nghĩa
FOUND	Thuộc tính này trả về TRUE khi một câu lệnh DML đã thay đổi một hàng, hoặc DQL đã truy cập một hàng.
ISOPEN	Thuộc tính này luôn trả về FALSE cho bất kỳ cursor ngầm định.
NOTFOUND	Thuộc tính này trả về TRUE khi một câu lệnh DML đã thay đổi một hàng, hoặc một DQL không thể truy cập một hàng khác.
ROWCOUNT	Thuộc tính này trả về một số hàng thay đổi bởi một câu lệnh DML hoặc số hàng được trả về bởi một câu lệnh SELECT INTO.

Các cursor ngầm định một hàng

Câu lệnh SELECT INTO hiện diện trong tất cả cursor ngầm định vốn truy vấn dữ liệu. Nó làm việc chỉ khi một hàng đơn được trả về bởi một câu lệnh select. Bạn có thể chọn một cột hoặc danh sách cột trong mệnh đề SELECT và gán các cột hàng vào các biến riêng lẻ hoặc chung vào một kiểu dữ liệu record.

Nguyên mẫu cho một cursor ngầm định một hàng không có các mệnh đề SQL WHERE, HAVING, GROUP BY, và ORDER BY chuẩn là

```
SELECT column1 [, column2 [, column(n+1) ]]
INTO variable1 [, variable2 [, variable(n+1) ]]
FROM table_name;
```

Cả hai chương trình mẫu sử dụng bảng ITEM được seed trong mã. Chương trình mẫu đầu tiên gán các giá trị cột và các biến vô hướng trên cơ sở một đối một:

```
DECLARE
    id          item.item_id%TYPE;
    title       item.item_title%TYPE;
    subtitle    item.item_subtitle%TYPE;
BEGIN
    SELECT item_id, item_title, item_subtitle
    INTO      id, title, subtitle
    FROM      item
    WHERE     ROWNUM < 2;
    dbms_output.put_line('Selected [ ' || title. | ' ]');
```

```
END;
```

```
/
```

Chương trình mẫu neo (anchor) tất cả biến vào bảng đích và giới hạn query chỉ trong một hàng bằng cách sử dụng giả cột Oracle SQL ROWNUM. Nó in một hàng:

```
Selected [Around the World in 80 Days]
```

Việc gán một đối một làm cho việc gõ nhập trở nên rất phiền phức sau một khoảng thời gian. Chúng cũng làm cho mã bảo trì tốn kém hơn theo thời gian. Quy ước phổ biến hơn là gán các cột dưới dạng một nhóm vào các kiểu dữ liệu record.

Ví dụ thứ hai gán các cột vào một kiểu dữ liệu record:

```
DECLARE
    TYPE item_record IS RECORD
    ( id          item.item_id%TYPE
    , title       item.item_title%TYPE
    , subtitle    item.item_subtitle%TYPE);
    dataset ITEM_RECORD;
BEGIN
    SELECT    item_id, item_title, item_subtitle
    INTO      dataset
    FROM      item
    WHERE     rownum < 2;
    dbms_output.put_line('Selected [' || dataset.title || ']');
END;
/
```

Trong khi các kiểu dữ liệu record đòi hỏi một sự xây dựng tường minh, các cột trong cấu trúc có thể được neo sang các kiểu dữ liệu cột. Cũng nên chú ý rằng component selector, hoặc dấu chấm, gắn kết biến record với tên phần tử.

Các cursor ngầm định một hàng là những cách sửa chữa nhanh tuyệt vời nhưng có một điểm yếu. Nó là một điểm yếu mà nhiều nhà phát triển cố gắng triển khai bằng cách sử dụng nó để đưa ra các ngoại lệ khi các cursor trả về nhiều hàng. Họ làm điều này bởi vì các cursor ngầm định một hàng đưa ra một lỗi "exact fetch returned too many rows" (ORA-01422) khi trả về nhiều hàng. Có sẵn các giải pháp tốt hơn để phát hiện các lỗi trước khi truy tìm dữ liệu. Bạn nên tự khám phá những lựa chọn khác khi triển khai mã và nơi có thể xử lý các lỗi một cách cụ thể. Các cursor tường minh thường là những giải pháp tốt hơn mỗi lần.

Các cursor ngầm định nhiều hàng

Có hai cách để tạo các cursor ngầm định nhiều hàng. Cách thứ nhất được thực hiện bằng cách viết bất kỳ câu lệnh DML trong một khối PL/SQL. Các câu lệnh DML được xem là các cursor ngầm định nhiều hàng mặc dù bạn có thể giới hạn chúng chỉ trong một hàng đơn. Cách thứ hai là viết một query nhúng trong một vòng lặp FOR cursor được định nghĩa trong một khối khai báo.

Query sau đây minh họa một cursor ngầm định được tạo bởi một câu lệnh DML:

```
BEGIN
    UPDATE system_user
    SET last_update_date = SYSDATE;
    IF SQL%FOUND THEN
        dbms_output.put_line('Updated [ | | SQL%ROWCOUNT | | ]');
    ELSE
        dbms_output.put_line('Nothing updated!');
    END IF;
END;
/
```

Như được minh họa trong bảng 4.6, thuộc tính cursor FOUND cho các cursor ngầm định chỉ trả về một giá trị true Boolean khi các hàng được cập nhật. Câu lệnh này cập nhật 5 hàng và in kết quả SQL ROWCOUNT:

```
Updated {5}
```

Bạn cũng có thể định nghĩa các cursor ngầm định nhiều hàng bên trong các câu lệnh vòng lặp FOR cursor. Đây là những câu lệnh select có những tính năng tuyệt vời: tất cả biến được cung cấp ngầm định trong phạm vi của vòng lặp FOR cursor.

Dòng mã sau đây minh họa một cursor ngầm định nhiều hàng trong một vòng lặp FOR cursor:

```
BEGIN
    FOR i IN (SELECT item_id, item_title FROM item) LOOP
        dbms_output.put_line('Item #[ | | i.item_id | | ] [ | | i.item_title | | ]');
    END LOOP;
END;
/
```

Cursor ngầm định này có sẵn trong phạm vi của index vòng lặp FOR cursor. Cursor index cho vòng lặp FOR cursor là một pointer dẫn sang một tập hợp kết quả trong một vùng làm việc query. Vùng làm việc query là một vùng bộ nhớ (được gọi là vùng ngữ cảnh) trong Oracle 11g Database Process Global Area (PGA).

Ghi chú

Thuộc tính SQLROWCOUNT trả về một giá trị rỗng (null) cho loại cursor ngầm định này.

Các cursor tường minh

Như được thảo luận trước đó trong phần này, bạn tạo một cursor tường minh khi bạn định nghĩa nó bên trong một khối khai báo. Các cursor tường minh có thể là các câu lệnh SELECT tĩnh (static) hoặc động (dynamic). Các câu lệnh SELECT tĩnh trả về cùng một query mỗi lần với các kết quả có thể khác nhau. Các kết quả thay đổi khi dữ liệu thay đổi trong các table hoặc view. Các câu lệnh SELECT động hành động như các thường trình con được tham số hoá. Chúng chạy các query khác nhau mỗi lần phụ thuộc vào các tham số thực sự được cung cấp khi chúng được mở.

Bạn mở các cursor ngầm định tĩnh và động một cách khác nhau miễn là chúng được định nghĩa bằng các tham số hình thức. Khi chúng không có các tham số hình thức, bạn mở chúng với cùng một cú pháp. Các tham số thực sự được ánh xạ bởi việc thay thế biến cục bộ.

Các cursor tường minh đòi hỏi bạn mở, truy tìm (fetch), và đóng chúng cho dù bạn sử dụng vòng lặp đơn giản hoặc vòng lặp WHILE hoặc các câu lệnh vòng lặp FOR cursor. Bạn sử dụng câu lệnh OPEN để mở các cursor, câu lệnh FETCH để truy tìm các record từ các cursor và câu lệnh CLOSE để đóng và giải phóng các nguồn tài nguyên của các cursor. Những câu lệnh này làm việc với các cursor động và tĩnh bên trong hoặc bên ngoài một cấu trúc vòng lặp. Các câu lệnh vòng lặp FOR cursor mở, truy tìm và đóng các cursor cho bạn một cách ngầm định. Các câu lệnh OPEN, FETCH, và CLOSE là những phần tử chính trong các mục "Các cursor tường minh tĩnh" và "Các cursor tường minh động".

Nguyên mẫu cho câu lệnh OPEN là

```
OPEN cursor_name [ (parameter1 [, parameter2 [, parameter (n+1) ] ] ) ] ;
```

Có hai nguyên mẫu cho câu lệnh FETCH. Một nguyên mẫu gán các cột riêng lẻ và các biến và các mẫu kia gán một hàng vào một cấu trúc record.

Nguyên mẫu để gán các cột riêng lẻ vào các biến tương hợp là

```
FETCH cursor_name  
  
INTO variable1 [, variable2 [, variable (n+1) ] ] ;
```

Nguyên mẫu để gán các hàng vào các biến cấu trúc record là

```
FETCH cursor_name  
  
INTO record_variable;
```

Nguyên mẫu cho câu lệnh CLOSE là

```
CLOSE cursor_name;
```

Trong khi bảng 4.6 liệt kê các thuộc tính cursor ngầm định, bảng 4.7 liệt kê các thuộc tính cursor tường minh. Các thuộc tính làm việc theo cùng một cách cho dù cursor tường minh là động hay tĩnh nhưng khác với tập hợp giới hạn với các cursor ngầm định. Các thuộc tính cursor tường minh trả về các kết quả khác nhau phụ thuộc vào nơi chúng được gọi tham chiếu với các câu lệnh OPEN, FETCH, và CLOSE.

Thuộc tính cursor FOUND báo hiệu rằng các hàng có sẵn để truy tìm từ cursor và thuộc tính NOTFOUND báo hiệu rằng tất cả hàng đã được truy tìm từ cursor. Thuộc tính ISOPEN cho biết cursor đã mở và là một điều nào đó mà bạn nên xem xét chạy trước khi cố mở một cursor. Như các cursor ngầm định, thuộc tính ROWCOUNT cho biết bao nhiêu hàng mà bạn đã truy tìm vào bất cứ thời điểm nào. Chỉ thuộc tính cursor ISOPEN làm việc bất cứ lúc nào mà không gặp lỗi gì cả. Ba thuộc tính kia đưa ra các lỗi khi cursor không mở. Bảng 4.7 thu thập những hành vi thay đổi này.

Các ví dụ sử dụng các câu lệnh vòng lặp đơn giản, nhưng bạn cũng có thể sử dụng các cursor tường minh trong các câu lệnh vòng lặp WHILE hoặc được xếp lồng bên trong các vòng lặp FOR dãy và cursor. Các cursor tĩnh và động được đề cập trong các mục khác nhau để tổ chức các ví dụ và làm nổi bật các điểm khác biệt.

Các cursor tường minh tĩnh

Một cursor tường minh tĩnh là một câu lệnh SQL SELECT vốn không thay đổi hành vi của nó. Một cursor tường minh có bốn thành phần: bạn định nghĩa, mở, truy tìm từ, và đóng một cursor. Chương trình mẫu định nghĩa một cursor làm một câu lệnh SELECT truy vấn bảng ITEM. Bảng và dữ liệu được seed được phân phối trong mã có thể download.

Bảng 4.7 Các thuộc tính cursor tường minh

Câu lệnh	Trạng thái	%FOUND	%NOTFOUND	%ISOPEN	%ROWCOUNT
OPEN	Before	Exception	Exception	FALSE	Exception
	After	NULL	NULL	TRUE	0
1 st FETCH	Before	NULL	NULL	TRUE	0
	After	TRUE	FALSE	TRUE	1
Next FETCH	Before	TRUE	FALSE	TRUE	1
	After	TRUE	FALSE	TRUE	n + 1
Last FETCH	Before	TRUE	FALSE	TRUE	n + 1
	After	FALSE	TRUE	TRUE	n + 1
CLOSE	Before	FALSE	TRUE	TRUE	n + 1
	After	Exception	Exception	FALSE	Exception

Chương trình sau đây định nghĩa, mở, truy tìm từ, và đóng một cursor tĩnh vào một loạt các biến vô hướng:

```

DECLARE
    id item.item_id%TYPE;
    title VARCHAR2(60);
    CURSOR c IS
        SELECT    item_id
           ,      item_title
        FROM      item;
BEGIN
    OPEN c;
    LOOP
        FETCH c
        INTO id, title;
        EXIT WHEN c%NOTFOUND;
        dbms_output.put_line('Title [' || title || ']');
    END LOOP;
    CLOSE c;
END;
/

```

Chương trình sử dụng việc truy tìm các cột đến các biến nào khớp với kiểu dữ liệu của chúng. Một trong các biến được neo sang bảng và biến kia được định nghĩa cụ thể. Bạn nên thực sự chọn kiểu này hoặc kiểu kia

nhưng ví dụ này muốn hiển thị cả hai. Chương trình thoát khi không còn các record để truy tìm nữa và nó đã in các tiêu đề sang console.

Chương trình này có thể được viết bằng cách sử dụng một câu lệnh vòng lặp FOR cursor. Vòng lặp FOR tạo một cách ngầm định các biến mà bạn có thể truy cập qua cursor index. Điều này loại bỏ nhu cầu bạn tạo chúng như được yêu cầu bởi câu lệnh vòng lặp đơn giản hoặc vòng lặp WHILE. Câu lệnh vòng lặp FOR cursor cũng mở, truy tìm và đóng cursor một cách ngầm định như sau:

```

DECLARE
    CURSOR c IS
        SELECT    item_id AS id
                ,    item_title AS title
        FROM      item;
BEGIN
    FOR i IN c LOOP
        dbms_output.put_line('Title [' || i.title || ']');
    END LOOP;
END;
/

```

Khi so sánh, vòng lặp FOR cursor dễ sử dụng hơn nhiều với một cursor tĩnh so với một câu lệnh vòng lặp đơn giản hoặc vòng lặp WHILE. Các biến được khai báo ngầm định bên trong vòng lặp FOR cursor không có ngữ cảnh bên ngoài câu lệnh vòng lặp FOR. Hạn chế này giới hạn cách bạn sử dụng các giá trị trả về từ một vòng lặp FOR cursor. Câu lệnh vòng lặp đơn giản hoặc vòng lặp WHILE là những giải pháp hiệu quả hơn khi bạn muốn gán những giá trị trả về và các biến vốn được trao đổi với các đơn vị chương trình khác. Chương 6 thảo luận một số ưu điểm này trong khi đề cập các hàm và thủ tục lưu trữ.

Cú pháp câu lệnh FETCH thay thế để gán một hàng dữ liệu vào một kiểu dữ liệu record được minh họa trong chương trình tiếp theo. Mọi thứ khác vẫn không đổi.

Chương trình sau đây định nghĩa, mở, truy tìm từ, và đóng một cursor tĩnh vào một cấu trúc record:

```

DECLARE
    TYPE item_record IS RECORD
    ( id    NUMBER
    , title VARCHAR2(60));
    item ITEM_RECORD;

```

```

CURSOR c IS
    SELECT    item_id
            , item_title
    FROM      item;
BEGIN
    OPEN c;
    LOOP
        FETCH c INTO item;
        EXIT WHEN c%NOTFOUND;
        dbms_output.put_line('Title [' || item.title || ']');
    END LOOP;
END;
/

```

Câu lệnh vòng lặp FOR cursor không hỗ trợ việc gán trực tiếp bất kỳ loại biến nhưng bạn có thể gán các giá trị bên trong câu lệnh vòng lặp FOR bằng cách sử dụng cursor index. Bạn có thể gán một cấu trúc record hoặc một phần tử của cấu trúc record.

Dòng mã sau đây minh họa việc gán một cấu trúc record từ cursor index:

```

DECLARE
    TYPE item_record IS RECORD
    ( id NUMBER
    , title VARCHAR2(60));
    explicit_item ITEM_RECORD;
    CURSOR c IS
        SELECT    item_id AS id
            , item_title AS title
        FROM      item;
BEGIN
    FOR i IN c LOOP
        explicit_item := i;
        dbms_output.put_line('Title [' || explicit_item.title || ']');
    END LOOP;
END;
/

```

Trong cả hai ví dụ này, có thể cursor không tìm thấy bất kỳ record nào. Khi một cursor ngấm định hoặc tường minh chạy và không có dữ liệu nào được tìm thấy, lỗi không được đưa ra. Bạn cần xác định xem có record được tìm thấy hay không. Điều này được thực hiện bằng cách sử dụng một câu lệnh IF và các thuộc tính cursor NOTFOUND và ROWCOUNT.

Chương trình sau đây in một thông báo no data found khi cursor không thể tìm thấy bất kỳ record nào bằng cách sử dụng một giá trị ITEM_ID âm mà lẽ ra không nằm trong dữ liệu:

```

DECLARE
    TYPE item_record IS RECORD
    ( id NUMBER
    , title VARCHAR2(60));
    item ITEM_RECORD;
    CURSOR c IS
        SELECT    item_id
        ,         item_title
        FROM      item
        WHERE     item_id = -1;
BEGIN
    OPEN c;
    LOOP
        FETCH c INTO item;
        IF c%NOTFOUND THEN
            IF c%ROWCOUNT = 0 THEN
                dbms_output.put_line('No Data Found');
            END IF;
            EXIT;
        ELSE
            dbms_output.put_line('Title [' || item.title || ']');
        END IF;
    END LOOP;
END;
/

```

Chương trình minh họa một sự thoát có điều kiện (conditional exit). Việc thoát chỉ có thể tiếp cận khi tất cả record được đọc. Một thông báo

đặc biệt được in chỉ khi ROWCOUNT trả về một giá trị 0. Điều này chỉ có thể xảy ra khi các hàng không được trả về bởi cursor. Bạn không thể sao chép logic này bên trong một câu lệnh vòng lặp FOR cursor.

Các cursor tường minh động

Các cursor tường minh động rất giống như các cursor tường minh tĩnh. Chúng sử dụng một câu lệnh SELECT SQL. Chỉ câu lệnh SELECT sử dụng các biến thay đổi hành vi query. Các biến thay thế những gì là các giá trị trực kiện (literal).

Các cursor tường minh động có bốn thành phần giống như các cursor tĩnh: bạn định nghĩa, mở, truy tìm từ, và đóng một cursor động. Chương trình mẫu định nghĩa một cursor dưới dạng một câu lệnh SELECT vốn truy vấn bảng ITEM cho một dãy giá trị. Cả hai biến được khai báo là biến cục bộ và được gán các giá trị trực kiện số. Tên của các biến cục bộ khác nhiều so với các tên cột hoặc các giá trị tên cột được thay thế bằng các giá trị biến.

Chương trình sau đây sử dụng hai biến cục bộ bên trong câu lệnh SELECT của cursor:

```

DECLARE
    lowend NUMBER := 1010;
    highend NUMBER := 1020;
    TYPE item_record IS RECORD
    ( id NUMBER
      , title VARCHAR2(60));
    item ITEM_RECORD;
    CURSOR c IS
    SELECT      item_id
              , item_title
    FROM        item
    WHERE       item_id BETWEEN lowend AND highend;
BEGIN
    OPEN c;
    LOOP
        FETCH c INTO item;
        EXIT WHEN c%NOTFOUND;
        dbms_output.put_line('Title [ ' || item.title || ' ]');
    END LOOP;

```

```
END;
```

```
/
```

Các giá trị cho các biến `lowend` và `highend` được thay thế khi bạn mở cursor. Điều này cũng làm việc trong các vòng lặp `FOR` và `WHILE` cursor bởi vì các biến được thay thế trong khi mở cursor. Trong khi điều này hành động hầu như là một query tĩnh bởi vì các biến là các hằng tĩnh trong phạm vi chương trình, bạn có thể thay đổi chương trình để sử dụng các tham số đầu vào như sau:

```
DECLARE
```

```
-- same as earlier example --
```

```
BEGIN
```

```
    lowend := TO_NUMBER(NVL(&1,1005));
```

```
    highend := TO_NUMBER(NVL(&2,1021));
```

```
    OPEN c;
```

```
    LOOP
```

```
        FETCH c INTO item;
```

```
        EXIT WHEN c%NOTFOUND;
```

```
        dbms_output.put_line('Title [' || item.title || ']');
```

```
    END LOOP;
```

```
END;
```

```
/
```

Bạn có thể tin cậy vào các biến cục bộ, nhưng hỗ trợ mã có thể gây rắc rối và khó hơn. Các cursor nên được định nghĩa để chấp nhận các tham số hình thức. Ví dụ tiếp theo thay thế ví dụ trước bằng cách thay đổi định nghĩa cursor và lệnh gọi đến câu lệnh `OPEN` như sau

```
DECLARE
```

```
    lowend NUMBER;
```

```
    highend NUMBER;
```

```
    item_id number := 1012;
```

```
    TYPE item_record IS RECORD
```

```
    ( id          NUMBER
```

```
    , title       VARCHAR2(60));
```

```
    item ITEM_RECORD;
```

```
    CURSOR c
```

```
    ( low_id NUMBER
```

```
    , high_id NUMBER) IS
```

```

SELECT      item_id
            ,
            item_title
FROM        item
WHERE       item_id BETWEEN low_id AND high_id;

BEGIN
    lowend := TO_NUMBER(NVL(&1,1005));
    highend := TO_NUMBER(NVL(&2,1021));
    OPEN c (lowend,highend);
    LOOP
        FETCH c INTO item;
        EXIT WHEN c%NOTFOUND;
        dbms_output.put_line('Title [' || item.title || ']');
    END LOOP;
END;
/

```

Các biến dãy trong câu lệnh **SELECT** không còn là các tên biến cục bộ nữa. Chúng là các biến cục bộ đối với cursor được định nghĩa bởi phân số hình thức trong định nghĩa cursor. Bạn nên chú ý rằng những biến này không có kích cỡ vật lý bởi vì điều đó được dẫn xuất vào thời gian chạy.

Khi bạn chạy chương trình, các giá trị đầu vào **&1** và **&2** được gán lần lượt vào các biến cục bộ **lowend** và **highend**. Các biến cục bộ trở thành các tham số thật sự được chuyển để mở cursor. Sau đó các tham số thật sự được gán vào các biến có phạm vi cursor **low_id** và **high_id**.

Lôgic này làm việc khi bạn thay thế một câu lệnh vòng lặp **FOR cursor**. Cấu trúc vòng lặp sau đây tương đương như cấu trúc vòng lặp trong câu lệnh vòng lặp đơn giản:

```

FOR i IN c (lowend,highend) LOOP
    item := i;
    dbms_output.put_line('Title [' || item.title || ']');
END LOOP;

```

Các câu lệnh Bulk

Các câu lệnh Bulk cho bạn chèn, cập nhật, hoặc xoá các tập hợp dữ liệu lớn ra khỏi các table hoặc view. Để sử dụng các câu lệnh **BULK COLLECT** với các câu lệnh **SELECT** và câu lệnh **FORALL** để chèn, cập nhật, hoặc xoá các tập hợp dữ liệu lớn.

Bảng 4.8 liệt kê các mô tả về hai thuộc tính bulk cursor. Mục "Câu lệnh INSERT" bên dưới phần "Các câu lệnh FORALL" minh họa cách sử dụng thuộc tính %BULK_ROWCOUNT. Chương 5 đề cập cách sử dụng thuộc tính %BULK_EXCEPTIONS.

Bảng 4.8 Các thuộc tính Bulk Cursor

Các thuộc tính Bulk	Định nghĩa
%BULK_EXCEPTIONS (i)	Thuộc tính %BULK_EXCEPTION (index) cho bạn thấy việc một hàng đã gặp phải một lỗi cho một câu lệnh bulk INSERT, UPDATE, hoặc DELETE hay không. Bạn truy cập các số liệu thống kê này bằng cách đặt chúng trong các câu lệnh vòng lặp FOR đây.
%BULK_ROWCOUNT (i)	Thuộc tính %BULK_ROWCOUNT (index) cho bạn thấy một phần tử có được thay đổi bởi một câu lệnh bulk INSERT, UPDATE hoặc DELETE hay không. Bạn truy cập những số liệu thống kê này bằng cách đặt chúng trong các câu lệnh vòng lặp FOR đây.

Phần này giải thích cách sử dụng các câu lệnh BULK COLLECT INTO và FORALL. Mục "Câu lệnh BULK COLLECT INTO" thảo luận những công dụng và những điểm khác biệt giữa các tập hợp vô hướng song song và tập hợp record. Mục "Các câu lệnh FORALL" có các phần độc lập về cách bạn có thể sử dụng các câu lệnh bulk INSERT, UPDATE và DELETE.

Các câu lệnh BULK COLLECT INTO

Câu lệnh BULK COLLECT INTO cho bạn chọn một cột dữ liệu và truyền nó vào các kiểu dữ liệu tập hợp Oracle. Bạn có thể sử dụng một bulk collect bên trong một câu lệnh SQL hoặc dưới dạng một phần của một câu lệnh FETCH. Tập hợp bulk câu lệnh SQL sử dụng một cursor ngầm định trong khi câu lệnh FETCH làm việc với một cursor tường minh. Bạn không thể giới hạn số hàng được trả về khi thực hiện bulk collection trong một cursor ngầm định. Câu lệnh FETCH cho bạn thêm câu lệnh LIMIT để thiết lập số hàng tối đa được đọc mỗi lần từ cursor. Bạn có thể sử dụng bất kỳ kiểu dữ liệu PL/SQL chuẩn hoặc do người dùng định nghĩa làm đích của một câu lệnh cursor ngầm định.

Sau đây là một nguyên mẫu cơ bản của câu lệnh bulk collection ngầm định:

```
SELECT column1 [, column2 [, column(n+1)]]
COLLECT BULK INTO collection1 [, collection2 [, collection(n+1)]]
```

```
FROM table_name
[WHERE where_clause_statements];
```

Các bulk collection được thực thi như là một phần của câu lệnh FETCH sử dụng một cursor tường minh. Chúng có nguyên mẫu sau đây:

```
FETCH cursor_name [(parameter1 [, parameter2 [, parameter(n+1)]])]
BULK COLLECT INTO collection1 [, collection2 [, collection(n+1)]]
[LIMIT rows_to_return];
```

Số cột được trả về bởi cursor tường minh xác định số đích tập hợp vô hướng hoặc cấu trúc của một đích tập hợp record. Câu lệnh SELECT định nghĩa số và loại cột được trả về bởi một cursor.

Bạn có thể sử dụng các câu lệnh BULK COLLECT INTO để chèn một loạt các đích hoặc một đích đơn. Một loạt các đích là một tập hợp biến tập hợp được tách biệt bằng các dấu phẩy. Các tập hợp được phân cách bằng dấu phẩy đích được gọi là các tập hợp song song bởi vì bạn thường quản lý chúng song song. Mục đích đơn là một tập hợp của cấu trúc record. Bạn không thể chèn một số cột vào một tập hợp của cấu trúc record và những cột khác và các tập hợp vô hướng trong cùng một lệnh gọi câu lệnh. Bất kỳ nỗ lực này sẽ đưa ra một lỗi PLS-00494 vốn không cho phép ép buộc vào nhiều đích record.

Câu lệnh BULK COLLECT INTO nhanh hơn nhiều so với một cursor chuẩn bởi vì nó có một chu kỳ phân tích cú pháp (parse), thực thi (execute), và truy tìm (fetch). Các cursor câu lệnh INTO ngầm định hoặc các cursor tường minh thông thường có nhiều thao tác phân tích cú pháp, thực thi và truy tìm hơn. Các thao tác Bulk mở rộng cấp độ tốt hơn khi số hàng tăng nhưng các thao tác rất lớn đòi hỏi các cấu hình cơ sở dữ liệu để hỗ trợ chúng.

Các mục "Các đích tập hợp song song" và "Các đích tập hợp record" minh họa các bulk collections sử dụng các cursor ngầm định. Các cursor tường minh được trình bày trong mục nhỏ cuối cùng cùng với câu lệnh LIMIT. Câu lệnh LIMIT cho phép ràng buộc kích cỡ của các lựa chọn lớn nhưng bạn chỉ có thể sử dụng nó với các cursor tường minh. Mục nhỏ cuối cùng về các đích tập hợp giới hạn trình bày cách bạn có thể làm việc với các ràng buộc vận hành cơ sở dữ liệu như PGA.

Các đích tập hợp song song

Các tập hợp vô hướng (Scalar collections) là các kiểu dữ liệu tập hợp SQL được hỗ trợ duy nhất. Khi bạn muốn chia sẻ dữ liệu với các chương trình bên ngoài hoặc các ứng dụng Web, bạn nên biến các lựa chọn lớn thành một loạt các tập hợp song song. Bạn có thể trao đổi các kiểu dữ liệu này với các chương trình bên ngoài và ứng dụng Web bằng cách sử dụng Oracle Call Interface (OCI).

Chương trình mẫu sử dụng một cursor câu lệnh BULK COLLECT INTO ngầm định và nó thực thi một lựa chọn lớn thành một tập hợp gồm các tập hợp vô hướng song song.

```

DECLARE
    TYPE title_collection IS TABLE OF VARCHAR2(60);
    TYPE subtitle_collection IS TABLE OF VARCHAR2(60);
    title TITLE_COLLECTION;
    subtitle SUBTITLE_COLLECTION;
BEGIN
    SELECT    item_title
    ,         item_subtitle
    BULK COLLECT INTO title, subtitle
    FROM      item;
    -- Print one element of one of the parallel collections.
    FOR i IN 1..title.COUNT LOOP
        dbms_output.put_line('Title [' || title(i) || ']');
    END LOOP;
END;
/

```

Chương trình minh họa cách bạn chuyển một tập hợp giá trị vào các tập hợp vô hướng. Sau khi bạn thay đổi cấu trúc hàng tự nhiên thành một tập hợp song song gồm các giá trị cột, điều này bảo đảm các tập hợp riêng biệt vẫn được đồng bộ. Tạo và duy trì các tập hợp được đồng bộ thì khó và mất thời gian. Bạn chỉ nên chọn hướng này khi bạn có nhu cầu di chuyển xung quanh bằng cách sử dụng cơ sở dữ liệu SQL.

Lý do chính để chọn các tập hợp song song là di chuyển dữ liệu từ PL/SQL sang các ngôn ngữ lập trình bên ngoài hoặc các ứng dụng web. Bạn nên chuyển đổi lại các tập hợp song song thành các tập hợp đa chiều sau khi chuyển dữ liệu. Các tập hợp đa chiều (Multiple dimension collections) thường là tập hợp các kiểu record.

Các đích tập hợp record

Những giới hạn hiện tại về việc xây dựng các tập hợp SQL giới hạn các tập hợp record chỉ trong các cấu trúc PL/SQL. Điều này có nghĩa bạn chỉ có thể sử dụng chúng bên trong các chương trình chạy độc quyền trong môi trường PL/SQL.

Bạn có thể sử dụng một tập hợp các loại record PL/SQL làm kiểu dữ liệu cho một tham số trong hàm hoặc thủ tục được lưu trữ khi bạn định nghĩa kiểu dữ liệu trong một form thông số gói PL/SQL. Giới hạn này có

nghĩa là bạn không thể trao kiểu dữ liệu với những chương trình bên ngoài. Bạn sử dụng các tập hợp song song khi bạn muốn trao đổi chúng với các chương trình bên ngoài và ứng dụng Web.

Chương trình mẫu sử dụng một cursor câu lệnh BULK COLLECT INTO ngầm định và nó thực thi một bulk selection thành một tập hợp của một cấu trúc record cục bộ:

```

DECLARE
    TYPE title_record IS RECORD
        ( title VARCHAR2(60)
          , subtitle VARCHAR2(60));
    TYPE collection IS TABLE OF TITLE_RECORD;
    full_title COLLECTION;
BEGIN
    SELECT item_title
        ,    item_subtitle
    BULK COLLECT INTO full_title
    FROM    item;
    -- Print one element of a structure.
    FOR i IN 1..full_title.COUNT LOOP
        dbms_output.put_line('Title [' || full_title(i).title || ']');
    END LOOP;
END;
/

```

Bạn sẽ tìm thấy các cơ cấu để tạo và truy cập các kiểu dữ liệu được định nghĩa trong các thông số gói trong chương 9. Trong tương lai Oracle có thể cho bạn tạo các tập hợp loại record SQL nhưng hiện tại bạn giới hạn chỉ trong các loại đối tượng vô hướng và các loại đối tượng do người dùng định nghĩa. Sự truy cập đến những cấu trúc này có thể đầu tiên đến từ Oracle Call Interface (OCI) vì nó đã bắt đầu cho bạn truy cập các mảng kết hợp PL/SQL và các cursor tham chiếu trong Oracle 10g Database, Release 2.

Các đích tập hợp được ràng buộc bởi LIMIT

Các lệnh LIMIT cho bạn thiết lập số hàng tối đa được trả về bởi một bulk collection. Nó ràng buộc bulk collection. Bạn chỉ có thể ràng buộc số hàng được trả về bởi các cursor tường minh trong một câu lệnh FETCH.

Khuyết điểm của phương pháp này là bị trói chặt của cách làm việc của các ứng dụng tương tác. Các ứng dụng tương tác thường đòi hỏi sự nỗ lực hết mình, không phải chỉ một số record. Các chương trình xử lý theo lô vốn quản lý các bước xử lý giao dịch tốt là ứng cử tốt nhất cho việc tận dụng phương pháp này.

Các cursor tường minh có thể trả về một đến nhiều cột dữ liệu từ câu lệnh SELECT bên trong. Bạn chọn đặt các giá trị cột đó bên trong một loạt các đích tập hợp vô hướng hoặc một đích tập hợp record. Hai mục nhỏ tiếp theo minh họa cả hai kỹ thuật.

Các đích tập hợp song song. Các đích tập hợp song song là các biến tập hợp vô hướng. Các tập hợp song song có thể khác nhau về kiểu dữ liệu, nhưng mỗi tập hợp có cùng một số hàng và giá trị index trong tập hợp. Bạn cần các biến tập hợp vô hướng cho mỗi cột được trả về bởi cursor tường minh.

Chương trình sau đây minh họa cách quản lý một bulk collection mỗi lần 10 hàng:

```

DECLARE
    -- Define scalar datatypes.
    TYPE title_collection IS TABLE OF VARCHAR2(60);
    TYPE subtitle_collection IS TABLE OF VARCHAR2(60);
    -- Define local variables.
    title      TITLE_COLLECTION;
    subtitle   SUBTITLE_COLLECTION;
    -- Define a static cursor.
    CURSOR c IS
        SELECT      item_title
        ,           item_subtitle
        FROM        item;
BEGIN
    OPEN c;
    LOOP
        FETCH c BULK COLLECT INTO title, subtitle LIMIT 10;
        EXIT WHEN title.COUNT = 0;
        FOR i IN 1..title.COUNT LOOP
            dbms_output.put_line('Title [' || title(i) || ']');
        END LOOP;
    END LOOP;
END LOOP;

```

```
END;
```

```
/
```

Tất cả các lần lặp lại qua vòng lặp truy tìm tất cả hàng có sẵn hoặc lên đến 10 hàng dữ liệu từ cursor mở. Điều này có nghĩa logic xử lý, quản lý tất cả hàng được trả về trước khi truy tìm tập hợp hàng tiếp theo. Điều kiện exit sử dụng phương pháp Oracle Collection API COUNT () để quyết định khi nào các hàng đã không được truy tìm bởi cursor. Đây là logic tương đương với câu lệnh sau đây cho một cursor bình thường:

```
EXIT WHEN c%NOTFOUND;
```

Trong khi 10 là một số nhỏ, ý kiến là giới hạn bộ nhớ tiêu thụ và giảm thiểu số lần phân tích cú pháp, thực thi và truy tìm. Giải pháp này hỗ trợ những trao đổi với các chương trình bên ngoài và ứng dụng web vốn bị giới hạn bởi thư viện OC18.

Các đích tập hợp record. Các đích tập hợp record là các biến tập hợp record. Một tập hợp record đích, ánh xạ chính xác sang cấu trúc trả về của câu lệnh SELECT của cursor. Bạn chỉ có thể sử dụng các tập hợp record bên trong môi trường PL/SQL.

Chương trình sau minh họa cách quản lý một bulk collection mỗi lần 10 hàng:

```
DECLARE
    TYPE title_record IS RECORD
    ( title VARCHAR2(60)
      , subtitle VARCHAR2(60));
    TYPE collection IS TABLE OF TITLE_RECORD;
    full_title COLLECTION;
    CURSOR c IS
    SELECT   item_title
      ,      item_subtitle
    FROM     item;
BEGIN
    OPEN c;
    LOOP
        FETCH c BULK COLLECT INTO full_title LIMIT 10;
        EXIT WHEN full_title.COUNT = 0;
        FOR i IN 1..full_title.COUNT LOOP
            dbms_output.put_line('Title [' || full_title(i).title || ']');
        END LOOP;
    END LOOP;
```

```

END LOOP;
END;
/

```

Logic này là một ảnh gương cho chương trình mẫu minh họa các tập hợp song song ngoại trừ nó trả về một hàng dữ liệu vào một tập hợp record. Tất cả lần lặp lại qua vòng lặp truy tìm tất cả hàng có sẵn hoặc lên đến 10 hàng dữ liệu từ cursor mở. Điều này có thể nghĩa lô gic xử lý, quản lý tất cả hàng trả về trước khi truy tìm tập hợp hàng kế tiếp. Điều kiện exit sử dụng phương thức Oracle Collection API COUNT () để quyết định khi nào các hàng không được truy tìm bởi cursor. Đây là logic tương đương câu lệnh sau đây cho một cursor bình thường:

```
EXIT WHEN c%NOTFOUND;
```

Các câu lệnh FORALL

Vòng lặp FORALL được thiết kế để làm việc với các tập hợp Oracle. Nó cho bạn chèn, cập nhật hoặc xoá dữ liệu lớn. Phần này tập trung vào cách bạn sử dụng câu lệnh FORALL và chuyển tiếp các tập hợp tham chiếu mà chương 7 đề cập chuyên sâu.

Các ví dụ dựa vào các ví dụ bulk collection. Chúng cũng phụ thuộc vào bảng đích ITEM_TEMP. Bạn nên tạo bảng này bằng cách sử dụng cú pháp sau đây:

```

CREATE TABLE ITEM_TEMP
( item_id NUMBER
, item_title VARCHAR2(62)
, item_subtitle VARCHAR2(60));

```

Các mục dưới đây được sắp xếp theo thứ tự để hỗ trợ mã mẫu. Bạn chèn, cập nhật và xoá dữ liệu bằng cách sử dụng các câu lệnh FORALL. Sau đó, bạn có thể loại bỏ bảng ITEM_TEMP.

Câu lệnh INSERT

Các Bulk insert đòi hỏi bạn sử dụng các tập hợp vô hướng bên trong mệnh đề VALUES. Bạn đưa ra một lỗi ORA-00947 not enough values (không đủ giá trị) khi bạn cố chèn một tập hợp record.

Mã mẫu sau đây sử dụng các tập hợp vô hướng để thực hiện một hoạt động bulk insert :

```

DECLARE
TYPE id_collection IS TABLE OF NUMBER;
TYPE title_collection IS TABLE OF VARCHAR2(60);
TYPE subtitle_collection IS TABLE OF VARCHAR2(60);

```

```

id      ID_COLLECTION;
title   TITLE_COLLECTION;
subtitle SUBTITLE_COLLECTION;
CURSOR c IS
    SELECT item_id
       , item_title
       , item_subtitle
    FROM item;
BEGIN
    OPEN c;
    LOOP
        FETCH c BULK COLLECT INTO id, title, subtitle LIMIT 10;
        EXIT WHEN title.COUNT = 0;
        FORALL i IN id.FIRST..id.LAST
            INSERT INTO item_temp VALUES (id(i),title(i),subtitle(i));
    END LOOP;
    FOR i IN id.FIRST..id.LAST LOOP
        dbms_output.put('Inserted |' || id(i) || '|');
        dbms_output.put_line(['| | SQL%BULK_ROWCOUNT(i) || ']);
    END LOOP;
END;
/

```

Câu lệnh FORALL đọc index của mảng vô hướng đầu tiên, nhưng có thể dễ dàng đọc các index khác. Chúng nên hoàn toàn giống nhau. Các Nested table sử dụng một index số nguyên dựa vào 1. Trong ví dụ này, dãy thấp sẽ luôn là 1 và giá trị dãy cao luôn là 10. Bạn đặt các biến tập hợp vô hướng ở đúng vị trí và tạo index cho chúng bằng một giá trị subscript (chỉ số dưới). Về cơ bản, điều này giống như thao tác insert từ một câu lệnh SELECT khác.

Vòng lặp FOR dãy thứ hai bắt giữ giá trị ITEM_ID và việc nó đã được chèn vào bảng hay không. Nó in 1 khi thành công và 0 khi không thành công. Điều này minh họa cách bạn có thể sử dụng thuộc tính %BULK_ROWCOUNT.

Lợi ích hiệu suất thật sự có được bằng cách đặt câu lệnh COMMIT sau khi kết thúc vòng lặp. Nếu không, bạn commit cho mỗi lô insert. Có những tình huống khi kích cỡ của dữ liệu được chèn làm cho việc đặt câu lệnh COMMIT dưới dạng câu lệnh cuối cùng sẽ có lợi hơn. Bạn nên kiểm

tra các hệ số kích cỡ và thảo luận chúng với DBA.

Câu lệnh UPDATE

Các Bulk updates đòi hỏi bạn sử dụng các tập hợp vô hướng bên trong mệnh đề SET và bất kỳ mệnh đề WHERE. Như câu lệnh INSERT, bạn đưa ra một lỗi ORA-00947 not enough values khi cố chèn một tập hợp record.

Mã mẫu sau đây sử dụng các tập hợp vô hướng để thực thi một hoạt động bulk update:

```

DECLARE
    TYPE id_collection IS TABLE OF NUMBER;
    TYPE title_collection IS TABLE OF VARCHAR2(60);
    id      ID_COLLECTION;
    title   TITLE_COLLECTION;
    CURSOR c IS
        SELECT      item_id
        ,           item_title
        FROM        item;
BEGIN
    OPEN c;
    LOOP
        FETCH c BULK COLLECT INTO id, title LIMIT 10;
        EXIT WHEN title.COUNT = 0;
        FORALL i IN id.FIRST..id.LAST
            UPDATE      item_temp
            SET         title = title(i) || ':'
            WHERE id = id(i);
    END LOOP;
END;
/

```

Câu lệnh FORALL đọc index của mảng vô hướng đầu tiên nhưng có thể dễ dàng đọc index tập hợp khác. Tất cả giá trị index nên giống y nhau. Ví dụ này dãy thấp luôn là 1 và giá trị dãy cao luôn là 10 bởi vì các index nested table là các số dựa vào 1. Bạn đặt các biến tập hợp vô hướng trong mệnh đề SET hoặc mệnh đề WHERE và tạo index chúng bằng một giá trị subscript (chỉ số dưới). Về cơ bản, điều này giống như một câu lệnh update tương quan.

Như với câu lệnh INSERT, bạn nên đánh giá nơi câu lệnh COMMIT thuộc về khi cập nhật số lượng lớn các record. Sau khi vòng lặp tốt hơn khi các cụm dữ liệu nhỏ và câu lệnh vòng lặp áp chót tốt nhất khi các cụm lớn. Trong cả hai trường hợp, bạn nên kiểm tra các hệ số kích cỡ và thảo luận chúng với DBA.

Câu lệnh DELETE

Các bulk delete đòi hỏi bạn sử dụng các tập hợp vô hướng bên trong mệnh đề WHERE. Như với các câu lệnh INSERT hoặc UPDATE, bạn không thể sử dụng các tập hợp record.

Mã mẫu sau đây sử dụng các tập hợp vô hướng để thực hiện một hoạt động bulk delete:

```

DECLARE
    TYPE id_collection IS TABLE OF NUMBER;
    TYPE title_collection IS TABLE OF VARCHAR2(60);
    id      ID_COLLECTION;
    title   TITLE_COLLECTION;
    CURSOR c IS
        SELECT      item_id
                   ,      item_title
        FROM        item;
BEGIN
    OPEN c;
    LOOP
        FETCH c BULK COLLECT INTO id, title LIMIT 10;
        EXIT WHEN title.COUNT = 0;
        FORALL i IN id.FIRST..id.LAST
            DELETE
            FROM      item_temp
            WHERE     subtitle IS NULL
            AND       Id = id(i);
        END LOOP;
    END;
/

```

Câu lệnh FORALL đọc index của mảng vô hướng đầu tiên, nhưng có thể dễ dàng đọc index tập hợp khác. Tất cả giá trị index khác nên giống y nhau. Trong ví dụ này, dãy thấp luôn là 1 và giá trị dãy cao luôn là 10 bởi vì các index nested table là các số dựa vào 1. Bạn đặt các biến tập

hợp vô hướng trong mệnh đề WHERE và tạo index cho chúng bằng một giá trị chỉ số dưới (subscript). Về cơ bản, điều này giống như một câu lệnh update tương quan.

Như với các câu lệnh INSERT và UPDATE, bạn nên đánh giá nơi câu lệnh COMMIT thuộc về khi xoá số lượng lớn các record. Sau khi vòng lặp tốt hơn khi các cụm dữ liệu nhỏ và câu lệnh vòng lặp áp chót tốt nhất khi các cụm lớn. Trong cả hai trường hợp, bạn nên kiểm tra các hệ số kích cỡ và thảo luận chúng với DBA.

Tóm tắt

Chương này đã trình bày những cấu trúc điều khiển trong PL/SQL, cách sử dụng hiệu quả các câu lệnh có điều kiện và các câu lệnh lặp lại, và cách xây dựng và quản lý các cursor trong những chương trình PL/SQL.

CHƯƠNG 5

QUẢN LÝ LỖI

Chương này đề cập đến việc quản lý lỗi PL/SQL. PL/SQL xử lý các lỗi trong khối ngoại lệ. Bạn sẽ tìm thấy hai loại lỗi trong PL/SQL: các lỗi biên dịch (compilation error) và lỗi thời gian chạy (run-time error). Bạn khám phá các lỗi biên dịch khi bạn chạy một chương trình khối nặc danh hoặc cố xây dựng một đơn vị chương trình lưu trữ - một hàm, thủ tục, hoặc loại đối tượng do người dùng định nghĩa. Các lỗi thời gian chạy phức tạp hơn bởi vì chúng có hai tình huống. Các lỗi run-time đưa ra các lỗi để quản lý trong khối thực thi hoặc khối ngoại lệ, nhưng bạn không thể đón bắt các lỗi run time được đưa ra trong khối khai báo.

Bạn sẽ học về cả hai loại lỗi và cách quản lý chúng trong chương này. Các mục trong chương này bao gồm

- Các loại ngoại lệ và phạm vi
 - Các lỗi biên dịch
 - Các lỗi thời gian chạy
- Các hàm cài sẵn quản lý ngoại lệ
- Các ngoại lệ do người dùng định nghĩa
 - Khai báo các ngoại lệ do người dùng định nghĩa
 - Các ngoại lệ động do người dùng định nghĩa
- Các hàm truy vết ngăn xếp ngoại lệ
 - Quản lý ngăn xếp ngoại lệ
 - Định dạng ngăn xếp ngoại lệ
- Quản lý ngoại lệ trigger cơ sở dữ liệu

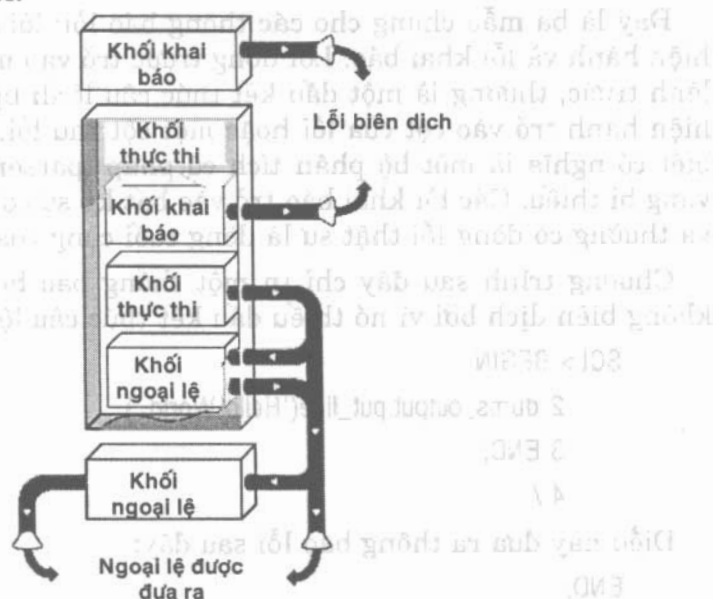
Chương này được thiết kế để đọc theo trình tự. Nếu bạn muốn tham khảo nhanh một điều gì đó, bạn nên xem lướt qua chương trước tiên trước khi nhắm vào một phần cụ thể.

Các loại ngoại lệ và phạm vi

Bạn có hai loại lỗi: các lỗi biên dịch và lỗi thời gian chạy. Các lỗi biên dịch xảy ra khi bạn phạm một lỗi gõ nhập chương trình hoặc định nghĩa chương trình. Các lỗi gõ nhập bao gồm quên một từ dành riêng, từ khoá hoặc dấu chấm phẩy. Các lỗi từ vựng này được đón bắt khi file text thuần túy được phân tích cú pháp trong quá trình biên dịch. Phân tích cú pháp (parsing) là tiến trình đọc một file text để bảo đảm rằng nó đáp ứng các qui tắc sử dụng từ vựng của một / lập trình. Các lỗi thời gian chạy xảy ra khi dữ liệu thật sự không đáp ứng được các qui tắc được định nghĩa bởi đơn vị chương trình.

Chương 3 giải thích phạm vi biến và trình bày cách nó chuyển từ khối tận cùng bên ngoài sang khối tận cùng bên trong. Trong khi phạm vi biến bắt đầu tại bên ngoài và thu hẹp khi chúng ta xếp lồng các đơn vị chương trình. Việc xử lý ngoại lệ diễn ra theo hướng ngược lại. Các ngoại lệ trong khối tận cùng được xử lý cục bộ hoặc được ném sang khối chứa theo trình tự cho đến khi chúng tiến đến session khởi đầu. Hình 5.1 minh hoạ tiến trình quản lý ngoại lệ này.

Các lỗi biên dịch thường được nhận biết nhanh chóng bởi vì chúng thất bại trong giai đoạn phân tích cú pháp. Một số lỗi trì hoãn không được xử lý cho đến khi bạn chạy các chương trình với những giá trị dữ liệu vốn kích khởi lỗi.



Hình 5.1 Phạm vi ngoại lệ và lộ trình

Bạn đã tạo các lỗi biên dịch trì hoãn khi các giá trị dữ liệu thật sự không phù hợp trong quá trình gán bởi vì chúng quá lớn hoặc có kiểu dữ liệu sai. Tất cả lỗi biên dịch được ném trở lại session và không thể được xử lý bởi phương thức xử lý ngoại lệ cục bộ, nhưng bạn có thể đón bắt chúng trong một khối wrapper (chứa bên ngoài).

Các lỗi thực thi runtime luôn có thể được đón bắt và được xử lý bởi khối ngoại lệ cục bộ hoặc bên ngoài. Các lỗi runtime trong các khối ngoại lệ chỉ có thể được đón bắt bởi một phương thức xử lý ngoại lệ khối ngoài. Bạn cũng có thể chọn không đón bắt các lỗi và ném chúng trở lại session SQL*Plus khởi động.

Hai mục tiếp theo đề cập đến các lỗi biên dịch và lỗi runtime.

Các lỗi biên dịch

Các lỗi biên dịch thường là các lỗi gõ nhập. Việc phân tích cú pháp file text PL/SQL thành một tập hợp chỉ lệnh được thông dịch được gọi là p-code, tìm các lỗi từ vựng. Các lỗi từ vựng xảy ra khi bạn dùng sai một dấu tách (delimiter), định danh (identifier), trực kiện (literal) hoặc chú giải (comment). Bạn có thể dùng sai các đơn vị từ vựng bằng việc:

- Quên một dấu chấm phẩy (dấu kết thúc câu lệnh)
- Sử dụng chỉ một dấu tách khi mà bạn nên sử dụng hai dấu tách hoặc không đặt một trực kiện chuỗi trong các dấu ngoặc đơn.
- Viết sai chính tả một định danh (các từ dành riêng và từ khoá)
- Chú giải một giá trị từ vựng được yêu cầu bởi các qui tắc phân tích cú pháp.

Đây là ba mẫu chung cho các thông báo lỗi: lỗi dòng trước, lỗi dòng hiện hành và lỗi khai báo. Lỗi dòng trước trở vào một lỗi trên dòng câu lệnh trước, thường là một dấu kết thúc câu lệnh bị thiếu. Các lỗi dòng hiện hành trở vào cột của lỗi hoặc một cột sau lỗi. Nói chung, sự khác biệt có nghĩa là một bộ phân tích cú pháp (parser) tìm một đơn vị từ vựng bị thiếu. Các lỗi khai báo trở vào bất kỳ sự cố trong khối khai báo và thường có dòng lỗi thật sự là dòng cuối cùng của thông báo lỗi.

Chương trình sau đây chỉ in một thông báo hello world nhưng nó không biên dịch bởi vì nó thiếu dấu kết thúc câu lệnh trên dòng 2:

```
SQL> BEGIN
      2 dbms_output.put_line('Hello World.')
      3 END;
      4 /
```

Điều này đưa ra thông báo lỗi sau đây:

```
END;
```

*

ERROR at line 3:

ORA-06550: line 3, column 1:

PLS-00103: Encountered the symbol "END" when expecting one of the following:

:= . (% ;

The symbol ";" was substituted for "END" to continue.

Thông báo lỗi này có thể trông có vẻ không hiểu được, nhưng thật ra nó hoàn toàn cung cấp nhiều thông tin khi bạn biết cách đọc nó. Dòng đầu tiên của thông báo lỗi cung cấp dòng mà lỗi đã xảy ra hoặc dòng sau lỗi. Dòng thứ hai đặt một dấu sao ngay bên dưới vị trí lỗi hoặc trên cột đầu tiên của dòng. Thông báo lỗi PLS-00103 được đưa ra bởi ví dụ cho thấy thiếu một đơn vị từ vựng ngay trước từ dành riêng END. Điều này cũng có nghĩa lỗi đã xảy ra trên một dòng câu lệnh trước dòng thông báo lỗi xuất hiện. Thông báo lỗi cũng cung cấp 5 giá trị từ vựng có thể có cho một ký hiệu bị thiếu. Parser đề nghị sử dụng một dấu chấm phẩy. Trong trường hợp này, dấu chấm phẩy hoặc dấu kết thúc câu lệnh là đơn vị từ vựng thiếu. Dấu chấm phẩy nên kết thúc câu lệnh trên dòng 2.

Ví dụ tiếp theo trình bày một lỗi biên dịch xảy ra trên cùng một dòng:

```
SQL> DECLARE
  2 a NUMBER := 0;
  3 b NUMBER;
  4 c NUMBER;
  5 BEGIN
  6 c := a b;
  7 dbms_output.put_line('[' || c || ']');
  8 END;
  9 /
```

Thông báo lỗi được hiển thị là

```
c := a b;
      *
```

ERROR at line 6:

ORA-06550: line 6, column 11:

PLS-00103: Encountered the symbol "B" when expecting one of the following:

. (* @ % & = - + ; < / > at in is mod remainder not rem

<an exponent (**)> <> or != or ~= >= <= <> and or like LIKE2_

LIKE4_ LIKEC_ between || multiset member SUBMULTISET_

The symbol "." was substituted for "B" to continue.

Thông báo lỗi PLS-00103 nói rằng thiếu một đơn vị từ vưng ngày trước biến b. Dấu sao trong dòng thứ hai bên dưới biến b cho bạn biết rằng lỗi xảy ra ngày trước biến. Bạn có thể sửa chữa chương trình này bằng cách đọc bất kỳ toán tử số học ở giữa các biến a và b.

Một biến thể trên thông báo lỗi trước đặt dấu sao ngay bên dưới nơi lỗi xảy ra trong một dòng câu lệnh. Chương trình sau đây đưa ra loại thông báo lỗi này:

```
SQL> DECLARE
      2 a NUMBER;
      3 BEGIN
      4 a = 1;
      5 END;
      6 /
```

Thông báo lỗi được hiển thị là

```
a = 1;
*
ERROR at line 4:
ORA-06550: line 4, column 5:
PLS-00103: Encountered the symbol "=" when expecting one of the following:
:= . ( @ % ;
The symbol ":= was inserted before "=" to continue.
```

Loại thông báo lỗi này trở sang việc sử dụng một toán tử so sánh nơi một toán tử gán xuất hiện. Đây là loại thông báo lỗi dễ đọc và dễ hiểu nhất.

• **Bạn nhận được một thông báo lỗi ít rõ ràng hơn khi bạn kích khởi một lỗi trong khối khai báo.** Ví dụ sau đây cố gán một chuỗi hai ký tự vào biến một ký tự trong khối khai báo:

```
SQL> DECLARE
      2 a CHAR := 'AB';
      3 BEGIN
      4 dbms_output.put_line([' | | a | | ']);
      5 END;
      6 /
```

Chương trình đưa ra thông báo lỗi sau đây vốn cung cấp rất ít thông tin nếu bạn cố áp dụng các qui tắc được thảo luận trước đó:

```
DECLARE
```

```
*
```


ERROR at line 1:

ORA-06502: PL/SQL: numeric or value error: character string buffer too small

ORA-06512: at line 2

Lỗi trở vào dòng 1, không giống như các lỗi trước, lỗi này không trở vào một sự cố trước câu lệnh DECLARE. Nó cho biết lỗi xảy ra trong khối khai báo và câu lệnh cuối cùng trong thông báo lỗi trở vào số dòng riêng biệt. Dòng cuối cùng thật ra là lỗi đầu tiên được viết vào vết ngăn xếp ngoại lệ (exception stack trace). Lỗi ORA-06512 trên dòng cuối cùng của thông báo lỗi trở vào dòng 2 trong chương trình. Lỗi xảy ra khi chương trình cố gán một trực diện chuỗi 'AB' vào một biến có kích cỡ ký tự đơn.

Phần này đã hướng dẫn bạn cách đọc và hiểu các lỗi biên dịch. Bây giờ bạn biết rằng có ba loại thông báo biên dịch chung. Một loại trở vào cột đầu tiên của một dòng câu lệnh khi lỗi xảy ra trên dòng câu lệnh trước. Một lỗi khác trở vào một cột nơi lỗi xảy ra hoặc một cột mà qua đó lỗi xảy ra trên cùng một dòng. Một lỗi thứ ba trở vào một lỗi khối khai báo và cung cấp số dòng cho lỗi thật sự ở cuối thông báo. Bạn có thể tăng tốc độ chẩn đoán các lỗi nếu áp dụng đúng một trong ba nguyên tắc này.

Các lỗi run-time

Các lỗi run-time có thể xảy ra trong các khối PL/SQL khai báo, thực thi và ngoại lệ. Lỗi dễ đón bắt và xử lý nhất là các lỗi đưa ra về một khối thực thi bởi vì chúng được đón bắt trước tiên bởi bất kỳ khối ngoại lệ cục bộ và tiếp theo bởi bất kỳ khối chứa. Một khác chỉ một khối ngoại ngoại có thể đón bắt các lỗi được đưa ra từ một khối ngoại lệ. Điều này có nghĩa rằng bạn có thể chỉ xử lý các lỗi khối ngoại lệ khi chúng xuất phát từ một chương trình PL/SQL được xếp lồng. Điều đó để lại các lỗi được đưa ra từ khối khai báo. Các lỗi khai báo không thể được đón bắt và được xử lý bởi một khối ngoại lệ cục bộ.

Các khối ngoại lệ chứa các khối WHEN. Các khối WHEN đón bắt các lỗi riêng biệt hay các lỗi khái quát. Nguyên mẫu cho khối WHEN là

```
WHEN {predefined_exception | user_defined_exception | OTHERS} THEN
    exception_handling_statement;
[RETURN | EXIT ];
```

Khối WHEN có thể lấy một tên ngoại lệ do Oracle ấn định sẵn, như các tên được liệt kê sau đó trong bảng 5.2. Đây là các lỗi riêng biệt. Các lỗi ấn định sẵn là các số lỗi đã biết vốn ánh xạ sang các tên. Chúng được định nghĩa trong gói SYS.STANDARD. Bạn có thể định nghĩa các ngoại lệ riêng của bạn bằng cách gán kiểu dữ liệu EXCEPTION. Các lỗi do người dùng định nghĩa cũng là các lỗi riêng biệt. Phần sau "Các lỗi do người dùng ấn định" đề cập các tiến trình định nghĩa những lỗi này. Bạn

sử dụng từ dành riêng OTHERS khi bạn muốn một mệnh đề WHEN để đón bắt bất kỳ ngoại lệ.

Bạn cũng có hai hàm cài sẵn: SQLCODE và SOLERRM. Bảng 5.1 đề cập đến hai hàm này bởi vì chúng được sử dụng trong các chương trình mẫu tiếp theo. Phụ lục J cũng đề cập đến hàm này sâu hơn.

Các mục nhỏ dưới đây đề cập đến các lỗi khối thực thi và ngoại lệ trước, sau đó đề cập đến các lỗi khối khai báo. Chúng được sắp xếp theo thứ tự bằng cách đó bởi vì bạn cần thấy các cơ cấu cơ bản làm việc như thế nào trước khi thấy chúng thất bại ra sao.

Các lỗi khối thực thi và ngoại lệ

Các lỗi được đưa ra trong khối thực thi được ném vào khối ngoại lệ cục bộ nơi chúng được đón bắt và được quản lý. Phương thức xử lý ngoại lệ (Exception handler) là một tên gọi khác cho khối ngoại lệ trong PL/SQL. Khi khối ngoại lệ cục bộ không đón bắt ngoại lệ và khối đã được gọi bởi một chương trình PL/SQL khác, ngoại lệ báo hiệu cho chương trình gọi. Chương trình gọi quản lý ngoại lệ được đưa ra miễn là nó đón bắt và quản lý loại ngoại lệ đó. Tiến trình này tiếp tục cho đến khi một khối ngoại lệ đón bắt và quản lý lỗi được đưa ra hoặc một ngoại lệ không được xử lý được đưa trở về môi trường SQL*Plus.

Dòng mã sau đây minh họa việc xử lý một lỗi gán được đưa ra bằng việc cố đặt một chuỗi hai ký tự vào biến một ký tự:

```

DECLARE
    a VARCHAR2(1);
    b VARCHAR2(2) := 'AB';
BEGIN
    a := b;
EXCEPTION
    WHEN value_error THEN
        dbms_output.put_line('You can't put [ | | b | | ] in a one character
string.');
```

```
END;
```

```
/
```

Chạy chương trình này, bạn tạo thông báo kết quả sau đây khi bạn đã bật SERVEROUTPUT trong session:

```
You can't put [AB] in a one character string.
```

Bảng 5.1 Các hàm cài sẵn quản lý ngoại lệ Oracle

Hàm	Các lỗi do Oracle ấn định sẵn	Các lỗi do người dùng ấn định
SQLCODE	Trả về một số âm vốn ánh xạ sang các ngoại lệ do Oracle ấn định sẵn nhưng cho một trường hợp đặc biệt: ngoại lệ NO_DATA_FOUND trả về dương 100.	Nếu EXCEPTION_INIT PRAGMA không được định nghĩa. Nếu một EXCEPTION_INIT PRAGMA được định nghĩa, nó trả về một số hợp lệ trong dãy âm -20001 đến -20999.
SQLERRM	Nó được quá tải (một khái niệm được đề cập trong chương 9) và thực thi khi có đủ điều kiện: trả về mã lỗi và thông báo lỗi được định nghĩa và thông báo cho một ngoại lệ được đưa ra nếu số không được chuyển đến nó. Trả về tham số số thật sự dưới dạng một số nguyên âm và một thông báo ngoại lệ không phải của Oracle nếu một số dương được chuyển đến nó hoặc một số âm vốn không phải là một ngoại lệ Oracle được ấn định sẵn.	Trả về tham số số thật sự dưới dạng một số nguyên âm và thông báo do Oracle định nghĩa nếu một số âm cho một ngoại lệ do Oracle định nghĩa sẵn được chuyển Trả về 1 và một thông báo "User-Defined Exception" nếu được kích khởi bởi lệnh RAISE. Trả về một số nguyên hợp lệ trong dãy âm 20001 đến âm 20999 và một thông báo text được xác lập bởi hàm RAISE_APPLICATION_INFO.

Ví dụ trước minh họa một lỗi cục bộ được đón bắt và được quản lý bởi một khối ngoại lệ cục bộ như thế nào. Khối ngoại lệ chỉ quản lý ngoại lệ đó; bất kỳ ngoại lệ khác được bỏ qua và được ném vào session SQL*Plus.

Dòng mã sau đây đưa ra một lỗi NO_DATA_FOUND bên trong khối trong vốn không được đón bắt cho đến khi ngoại lệ khối ngoài được xử lý:

```

DECLARE
    a NUMBER;
BEGIN
    DECLARE
        b VARCHAR2(2);
    BEGIN

```

```

SELECT    1
INTO      b
FROM      dual
WHERE     1 = 2;
a := b;
EXCEPTION
  WHEN value_error THEN
    dbms_output.put_line('You can''t put [' || b || '] in a one character
string.');
```

```

END;
EXCEPTION
  WHEN others THEN
    dbms_output.put_line('Caught in outer block [' || SQLERRM || '].');
```

```

END;
/
```

Lỗi được đưa ra là một ngoại lệ `NO_DATA_FOUND`. Khối trong chỉ kiểm tra tìm một ngoại lệ `VALUE_ERROR` bởi vì nó là một khối catch riêng biệt. Sau đó, chương trình ném lỗi vào khối chứa nơi ngoại lệ `OTHERS` đón bắt nó. Thông báo thật sự in thông báo `SQLERRM`:

```
Caught in outer block [ORA-01403: no data found].
```

Bạn có thể đưa ra bằng tay một ngoại lệ do người dùng được định nghĩa mà không gặp phải một ngoại lệ nào. Kỹ thuật này cho bạn thấy điều gì xảy ra khi một lỗi được đưa ra bên trong một khối ngoại lệ:

```

DECLARE
  a NUMBER;
  e EXCEPTION;
BEGIN
  DECLARE
    b VARCHAR2(2) := 'AB';
  BEGIN
    RAISE e;
  EXCEPTION
    WHEN others THEN
      a := b;
      dbms_output.put_line('Does not reach this line.');
```

```

END;
```

```

EXCEPTION
    WHEN others THEN
        dbms_output.put_line('Caught in outer block [' || SQLCODE || ']');
END;
/

```

Câu lệnh RAISE chuyển quyền điều khiển đến phương thức xử lý ngoại lệ cục bộ. Bên trong khối WHEN nó cố gán một trục kiện nhiều ký tự vào chuỗi một ký tự, điều này đưa ra một ngoại lệ VALUE_ERROR. Một lỗi được đưa ra bên trong một khối ngoại lệ được chuyển đến khối gọi hoặc môi trường SQL*Plus. Chương trình tạo kết quả SQLCODE từ phương thức xử lý ngoại lệ khối ngoài:

```
Caught in outer block [6502]
```

Phần này đã minh họa những điểm cơ bản về việc quản lý ngoại lệ thời gian chạy. Bạn nên chú ý khi đưa ra một lỗi trong khối thực thi, nó được xử lý cục bộ ở nơi có thể. Khi khối ngoại lệ cục bộ không quản lý lỗi, nó được gọi đến một khối ngoài hoặc môi trường SQL*Plus.

Các lỗi khối khai báo

Khối khai báo dễ bị các lỗi run-time. Những lỗi này xảy ra khi bạn thực hiện các phép gán động trong khối khai báo. Là một thói quen viết mã tốt, bạn chỉ nên thực hiện các phép gán trục kiện bên trong phần khai báo. Bạn nên thực hiện các phép gán động bên trong khối thực thi bởi vì các lỗi khối ngoại lệ được đón bắt và được quản lý bởi khối ngoại lệ cục bộ. Các lỗi gán Runtime trong khối khai báo không được đón bắt bởi khối ngoại lệ cục bộ.

Chương trình khối nặc danh sau đây sử dụng một phép gán động qua một biến thay thế:

```

DECLARE
    a varchar2(1) := '&1';
BEGIN
    dbms_output.put_line('Substituted variable value [' || a || ']');
EXCEPTION
    WHEN others THEN
        dbms_output.put_line('Local exception caught.');
```

```

END;
/

```

Việc thay thế một chuỗi hai ký tự đưa ra ngoại lệ sau đây:

```
DECLARE
```

*

ERROR at line 1:

ORA-06502: PL/SQL: numeric or value error: character string buffer too small

ORA-06512: at line 2

Thông báo lỗi này biểu thị rằng một lỗi định kích cỡ động (dynamic sizing error) bỏ qua một khối ngoại lệ cục bộ. Bạn có thể đón bắt lỗi khi đặt phần khai báo dưới dạng một khối bên trong trong một khối PL/SQL khác. Điều này dịch chuyển việc gán động từ một khối ngoài sang một khối trong.

Chương trình sau đây minh họa việc đặt lỗi khai báo trong một khối PL/SQL khác:

```
BEGIN
  DECLARE
    a varchar2(1) := '&1';
  BEGIN
    dbms_output.put_line('Substituted variable value [' || a || ']');
  EXCEPTION
  WHEN others THEN
    dbms_output.put_line('Local exception caught.');
```

```
END;
EXCEPTION
WHEN others THEN
  dbms_output.put_line('Outer exception caught.');
```

```
END;
/
```

Khi bạn gán một chuỗi hai ký tự, lỗi được đón bắt bởi khối ngoại lệ ngoài như được minh họa:

```
Outer exception caught
```

Hành vi này hiện hữu trong các đơn vị chương trình lưu trữ như các hàm và thủ tục. Trong khi các thủ tục đòi hỏi việc bao bọc các lệnh gọi của chúng, các hàm thì không. Nếu bạn gọi trực tiếp một hàm từ SQL, nó có thể đưa ra một ngoại lệ không được xử lý.

Ghi chú

Bạn có thể gọi các hàm lưu trữ từ SQL khi chúng trả về một kiểu dữ liệu SQL riêng.

Hàm sau đây sao chép vấn đề gán động trong một đơn vị lập trình lưu trữ:

```
CREATE OR REPLACE FUNCTION runtime_error
(variable_in VARCHAR2) RETURN VARCHAR2 IS
  a VARCHAR2(1) := variable_in;
BEGIN
  RETURN NULL;
EXCEPTION
  WHEN others THEN
    dbms_output.put_line('Function error.');
```

END;

/

Bạn có thể gọi hàm này trong SQL bằng cách sử dụng một câu lệnh truy vấn từ DUAL giả table:

```
SELECT runtime_error ('AB') FROM dual;
```

Nó tạo ngoại lệ không được xử lý sau đây:

```
SELECT runtime_error ('AB') FROM dual;
```

*

ERROR at line 1:

ORA-06502: PL/SQL: numeric or value error: character string buffer too small

ORA-06512: at "PLSQL.RUNTIME_ERROR", line 3

Phần này đã cho thấy bạn đang thực hiện các phép gán động trong các khối thực thi bởi vì PL/SQL không đón bắt các lỗi gán động trong các phương thức xử lý ngoại lệ cục bộ. Bạn cũng đã thấy rằng bạn có thể bao bọc các phép gán động bên trong một khối ngoại để đón bắt các lỗi.

Các hàm cài sẵn quản lý ngoại lệ

Oracle cung cấp một loạt các ngoại lệ định nghĩa sẵn trong gói STANDARD. Đây là những công cụ hữu dụng trong việc gỡ rối các chương trình Oracle PL/SQL. Hầu hết các lỗi đưa ra một số âm dưới dạng số lỗi của chúng. Một ORA-01001 ánh xạ sang ngoại lệ định nghĩa sẵn INVALID_CURSOR. Bạn tìm thấy các mã lỗi bằng cách sử dụng hàm cài sẵn SQLCODE. Các ngoại lệ định nghĩa sẵn được ghi chú trong bảng 5.2.

Bảng 5.2 Các ngoại lệ định nghĩa sẵn trong gói Standard.

Ngoại lệ	Lỗi	Khi nào được đưa ra
ACCESS_INTO_NULL	ORA-06530	Bạn gặp phải lỗi này khi cố truy cập một đối tượng không được khởi tạo.
CASE_NOT_FOUND	ORA-06592	Bạn gặp phải lỗi này khi bạn đã định nghĩa một câu lệnh CASE mà không có một mệnh đề ELSE và các câu lệnh CASE không đáp ứng điều kiện run-time.
COLLECTION_IS_NULL	ORA-06531	Bạn gặp phải lỗi này khi cố truy cập một NESTED TABLE hoặc VARRAY không được khởi tạo.
CURSOR_ALREADY_OPEN	ORA-06511	Bạn gặp phải lỗi này khi cố mở một cursor đã mở.
DUP_VAL_ON_INDEX	ORA-00001	Bạn gặp phải lỗi này khi cố chèn một giá trị bản sao sang cột của một bảng khi có một index duy nhất trên đó.
INVALID_CURSOR	ORA-01001	Bạn gặp phải lỗi này khi cố thực hiện một thao tác không được cho phép trên một cursor như đóng một cursor được đóng.
INVALID_NUMBER	ORA-01722	Bạn gặp phải lỗi này khi cố gán một thứ gì đó không phải một số vào một số hoặc khi mệnh đề LIMIT của một thao tác bulk fetch trả về một số không dương.
LOGIN_DENIED	ORA-01017	Bạn gặp phải lỗi này khi cố đăng nhập với một chương trình với username hoặc password không hợp lệ.
NO_DATA_FOUND	ORA-01403	Bạn gặp phải lỗi này khi cố sử dụng cấu trúc SELECT-INTO và câu lệnh trả về một giá trị rỗng, khi bạn cố truy cập một phần tử bị xoá trong một nested table hoặc khi bạn cố truy cập một phần tử không được khởi tạo trong một bảng index-by, (được gọi là một mảng kết hợp kể từ Oracle 10g).

NO_DATA_NEEDED	ORA-06548	Bạn đưa ra lỗi này khi một đối tượng gọi (caller) đến một hàm PIPELINED báo hiệu không cần thêm các hàng.
NOT_LOGGED_ON	ORA-01012	Bạn gặp phải lỗi này khi một chương trình thực thi một lệnh gọi cơ sở dữ liệu và không được kết nối, điều này thường xảy ra sau khi instance đã ngắt nối kết session của bạn.
PROGRAM_ERROR	ORA-06501	Bạn gặp phải lỗi này quá thường xuyên khi một lỗi xảy ra mà Oracle đã chưa chính thức bẫy. Điều này xảy ra với một số tính năng Oracle của cơ sở dữ liệu.
ROWTYPE_MISMATCH	ORA-06504	Bạn gặp phải lỗi này khi cấu trúc cursor không đồng ý với biến cursor PL/SQL hoặc một tham số cursor thật sự khác với một tham số cursor hình thức.
SELF_IS_NULL	ORA-30625	Bạn gặp phải lỗi này khi bạn cố gọi một phương thức thành viên không static là đối tượng trong đó một instance của lỗi đối tượng đã không được khởi tạo.
STORAGE_ERROR	ORA-06500	Bạn gặp phải lỗi này khi SGA đã hết bộ nhớ hoặc đã bị hỏng.
SUBSCRIPT_BEYOND_COUNT	ORA-06533	Bạn gặp phải lỗi này khi không gian được cấp phát cho NESTED TABLE hoặc VARRAY nhỏ hơn giá trị subscript được sử dụng.
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	Bạn gặp phải lỗi này khi sử dụng một giá trị index không hợp lệ để truy cập một NESTED TABLE hoặc VARRAY nghĩa là một số nguyên không dương.
SYS_INVALID_ROWID	ORA-01410	Bạn gặp phải lỗi này khi cố chuyển đổi một chuỗi thành một giá trị ROWID không hợp lệ.
TIMEOUT_ON_RESOURCE	ORA-00051	Bạn gặp phải lỗi này khi cơ sở dữ liệu không thể bảo vệ một khoá (lock) cho một nguồn tài nguyên.

TOO_MANY_ROWS	ORA-01422	Bạn gặp phải lỗi này khi sử dụng SELECT-INTO và query trả về nhiều hàng. Bạn cũng có thể nhận được lỗi này khi một subquery trả về nhiều hàng và toán tử so sánh là một toán tử đẳng thức.
USERENV_ COMMITSCN_ERROR	ORA-01725	Bạn chỉ có thể sử dụng hàm USERENV ('COMMITSCN') làm một biểu thức cấp trên cùng trong một mệnh đề VALUES của một câu lệnh INSERT hoặc làm một toán hạng phải trong mệnh đề SET của một câu lệnh UPDATE.
VALUE_ERROR	ORA-06502	Bạn gặp phải lỗi này khi bạn cố gán một biến vào một biến khác vốn quá nhỏ không thể chứa nó được.
ZERO_DIVIDE	ORA-01476	Bạn gặp phải lỗi này khi cố chia một số cho zero.

Đây là những công cụ rất tiện lợi để viết các phương thức xử lý ngoại lệ. Bạn nên sử dụng những công cụ này khi chúng đáp ứng nhu cầu của bạn. Khi chúng không đáp ứng nhu cầu của bạn, bạn nên tạo các ngoại lệ do người dùng định nghĩa.

Các ngoại lệ do người dùng định nghĩa

Các ngoại lệ do người dùng định nghĩa có thể được khai báo bằng hai cách: bạn có thể khai báo một biến EXCEPTION trong khối khai báo hoặc bạn có thể xây dựng một ngoại lệ động (dynamic exception) trong khối thực thi.

Có hai tùy chọn khi bạn khai báo một biến EXCEPTION. Phần thực thi đơn giản nhất cho bạn khai báo một biến và đưa nó ra theo tên. Phần thực thi thay thế cho bạn khai báo biến và ánh xạ nó sang một mã lỗi Oracle hợp lệ. Tùy chọn trước đòi hỏi bạn đón bắt các lỗi do người dùng định nghĩa bằng cách sử dụng ngoại lệ catch OTHERS. Tùy chọn sau cho bạn xây dựng các khối WHEN riêng biệt cho các lỗi riêng lẻ.

Bạn có thể xây dựng các ngoại lệ động bằng cách gọi hàm RAISE_APPLICATION_ERROR. Bạn có thể sử dụng một dãy giữa -20.000 và -20.999 khi bạn đưa ra các ngoại lệ động. Bạn gán các thông báo lỗi vào thời gian chạy khi sử dụng các ngoại lệ động. Chúng cũng không đòi hỏi bạn khai báo trước các biến EXCEPTION. Bạn nâng cao tính hữu dụng của các ngoại lệ động bằng cách khai báo các biến ngoại lệ. Chúng cải tiến cách bạn đón bắt các ngoại lệ.

Các mục dưới đây được chia thành phần khai báo các ngoại lệ do người dùng định nghĩa và phần đưa ra các ngoại lệ động do người dùng định nghĩa. Bạn nên đọc chúng theo trình tự bởi vì chủ đề thứ hai phụ thuộc vào việc bạn hiểu cách khai báo các biến EXCEPTION.

Khai báo các ngoại lệ do người dùng định nghĩa

Bạn khai báo một ngoại lệ như bất kỳ biến khác trong PL/SQL. Sau khi khai báo nó, bạn có thể đưa ra ngoại lệ, nhưng bạn không có cách nào để đón bắt nó trong phương thức xử lý ngoại lệ. Mục đích đằng sau ngoại lệ do người dùng định nghĩa chỉ định cách nào bạn khai báo nó.

Chương trình sau đây khai báo và đưa ra một ngoại lệ:

```
DECLARE
    e EXCEPTION;
BEGIN
    RAISE e;
    dbms_output.put_line('Can''t get here.');
```

EXCEPTION

```
    WHEN OTHERS THEN
        IF SQLCODE = 1 THEN
            dbms_output.put_line('This is a [' || SQLERRM || '].');
```

END IF;

```
END;
```

/

Chương trình này đưa ra ngoại lệ và in:

This is a [User-Defined Exception].

Theo mặc định, tất cả ngoại lệ do người dùng định nghĩa có một giá trị SQLCODE là 1. Khối IF cho bạn đón bắt các lỗi do người dùng định nghĩa một cách riêng biệt bên trong một khối WHEN chung.

Một tiến trình khai báo hai bước cho bạn khai báo một ngoại lệ và ánh xạ nó sang một số. Bước thứ nhất khai báo biến EXCEPTION. Bước thứ hai khai báo một PRAGMA. PRAGMA là một chỉ lệnh biên dịch. Bạn sử dụng PRAGMA để ra lệnh trình biên dịch (compiler) thực hiện một điều gì đó khác. PL/SQL hỗ trợ một số chỉ lệnh PRAGMA. Bạn sử dụng chỉ lệnh EXCEPTION_INIT để ánh xạ một ngoại lệ sang một mã lỗi. Tham số thứ nhất của lệnh gọi EXCEPTION_INIT là một biến EXCEPTION do người dùng định nghĩa và tham số thứ hai là một số lỗi hợp lệ.

••••• Thủ thuật

Bạn nên tránh ánh xạ do người dùng định nghĩa vào một mã lỗi vốn đã là một ngoại lệ định nghĩa sẵn như được trình bày trong bảng 5.2 trước.

Chương trình mẫu định nghĩa một biến EXCEPTION và ánh xạ ngoại lệ sang một số lỗi:

```
DECLARE
    a VARCHAR2(20);
    invalid_userenv_parameter EXCEPTION;
    PRAGMA EXCEPTION_INIT(invalid_userenv_parameter,-2003);
BEGIN
    a := SYS_CONTEXT('USERENV','PROXY_PUSHER');
EXCEPTION
    WHEN invalid_userenv_parameter THEN
        dbms_output.put_line(SQLERRM);
END;
/
```

ORA-02003 là một mã lỗi thực được gán trong việc thực thi gói STANDARD. Lựa chọn INVALID_USERENV_PARAMETER cũng phản ánh định nghĩa thực của chúng trong phần thân của gói STANDARD.

Mã in thông báo lỗi Oracle chuẩn:

ORA-02003: invalid USERENV parameter

Các ngoại lệ động do người dùng định nghĩa

Các ngoại lệ động do người dùng định nghĩa cho bạn đưa ra một ngoại lệ, gán cho nó một số và quản lý việc bạn có thêm lỗi mới vào một danh sách các lỗi (được gọi là một ngăn xếp lỗi) hay không. Sau đây là nguyên mẫu cho hàm ngoại lệ động:

```
RAISE_APPLICATION_ERROR (error_number, error_message [, keep_errors])
```

Tham số hình thức thứ nhất lấy một số lỗi trong dãy -20.000 đến -20.999. Bạn đưa ra một lỗi ORA-21000 khi bạn cung cấp bất kỳ giá trị khác. Tham số hình thức thứ hai là một thông báo lỗi. Tham số hình thức cuối cùng thì tùy chọn và có một giá trị mặc định là FALSE. Bạn ra lệnh rằng lỗi nên được thêm vào bất kỳ ngăn xếp lỗi (error stack) khi bạn cung cấp một giá trị TRUE tùy chọn.

Dòng mã sau đây minh họa việc đưa ra một ngoại lệ động mà trước đó không khai báo một biến EXCEPTION do người dùng định nghĩa:

```

BEGIN
    RAISE_APPLICATION_ERROR(-20001,'A not too original message.');
```

```

EXCEPTION
    WHEN others THEN
        dbms_output.put_line(SQLERRM);
END;
/
```

Ngoại lệ đón bắt lỗi sử dụng từ dành riêng OTHERS và in:

```
ORA-20001: A not too original message.
```

Chương trình kế tiếp kết hợp việc khai báo một biến EXCEPTION, ánh xạ một mã lỗi do người dùng định nghĩa sang một biến EXCEPTION và sau đó xác lập động thông báo. Điều này minh họa tất cả ba tham số có thể làm việc với nhau như thế nào để cung cấp cho bạn quyền điều khiển qua suốt chương trình như được minh họa:

```

DECLARE
    e EXCEPTION;
    PRAGMA EXCEPTION_INIT(e,-20001);
BEGIN
    RAISE_APPLICATION_ERROR(-20001,'A less than original message.');
```

```

EXCEPTION
    WHEN e THEN
        dbms_output.put_line(SQLERRM);
END;
/
```

Điều này in ra thông báo lỗi động từ hàm RAISE_APPLICATION_ERROR():

```
ORA-20001: A less than original message.
```

Không giống các file thông báo cho các lỗi Oracle chuẩn, thông báo này động đối với các đơn vị chương trình PL/SQL. SQLERRM cài sẵn không dò tìm thông báo, nhưng đơn giản thay thế trực tiếp chuỗi được cung cấp cho hàm RAISE_APPLICATION_ERROR.

Phần này đã minh họa cách khai báo các ngoại lệ và sử dụng chúng. Bạn đã thấy cách ánh xạ các lỗi Oracle và các định nghĩa thông báo lỗi hiện có sang các ngoại lệ do người dùng định nghĩa. Bạn cũng đã thấy cách cung cấp động các thông báo lỗi riêng của bạn.

Các hàm ngăn xếp ngoại lệ

Ngăn xếp ngoại lệ (exception stack) là việc xếp trình tự các lỗi từ sự kiện (event) kích khởi đến khối gọi của mã. PL/SQL đưa ra một ngoại lệ trong khối thực thi khi một sự cố xảy ra và chạy mã trong khối ngoại lệ cục bộ của nó. Nếu sự cố nằm trong một khối PL/SQL được xếp lồng hoặc được tham chiếu, đầu tiên nó chạy một phương thức xử lý ngoại lệ cục bộ trước khi chạy phương thức xử lý ngoại lệ của đơn vị chương trình gọi. Sau đó nó tiếp tục chạy các khối ngoại lệ có sẵn hoặc trả các lỗi trở về ngăn xếp lỗi cho đến khi nó trả quyền điều khiển trở về khối PL/SQL tận cùng bên ngoài.

Khi PL/SQL không chứa các khối ngoại lệ, các mã số dòng và mã lỗi sẽ được lan truyền. Bắt đầu trong Oracle 10g, bạn có thể sử dụng một khối ngoại lệ và khối DBMS_UTILITY để có được mã số dòng và mã lỗi.

Có hai phương pháp để quản lý các lỗi trong PL/SQL: việc lựa chọn cái nào được sử dụng phụ thuộc vào các yêu cầu điều khiển giao tác ứng dụng. Nếu bạn gặp phải một lỗi nghiêm trọng cho logic nghiệp vụ của ứng dụng, bạn cần đưa ra một ngoại lệ. Ngoại lệ nên dừng tiến trình nghiệp vụ và phục hồi giao tác trở lại một trạng thái nơi dữ liệu an toàn và nhất quán.

Khi lỗi không nghiêm trọng đối với logic nghiệp vụ ứng dụng, bạn có thể chọn ghi chép lỗi trong một bảng và cho phép giao tác hoàn tất. Phần "Quản lý ngoại lệ trigger cơ sở dữ liệu hướng dẫn cách ghi chép lại lỗi này. Tuy ví dụ minh họa cách bạn ghi chép một lỗi không nghiêm trọng, nhưng nó không đề cập đến việc định nghĩa cơ cấu phục hồi. Bạn phải phân tích những gì mà giao tác đang làm để hoạch định cách bạn có thể phục hồi thông tin.

Hai phần tiếp theo nêu bật về việc quản lý ngăn xếp lỗi trong các khối PL/SQL được đặt tên. Đầu tiên bạn sẽ học cách quản lý các ngăn xếp lỗi trong các đơn vị PL/SQL khối nặc danh và được đặt tên. Sau đó, bạn sẽ học cách sử dụng hàm `FORMAT_ERROR_BACKTRACE`.

Quản lý ngăn xếp ngoại lệ

Phần này hướng dẫn cách định dạng các ngăn xếp lỗi (error stack) mà không sử dụng các hàm gói DBMS_UTILITY - một kỹ năng cần thiết khi bạn làm việc trong Oracle 9i trở về trước. Phần này cũng tham chiếu về phía trước những khái niệm được đề cập trong các chương 6 và 7 trên các hàm và thủ tục lưu trữ cũng như các tập hợp. Bạn cũng học cách xây dựng một thủ tục quản lý sự kiện lỗi chuẩn và cách test nó bằng một tập hợp thủ tục liên quan.

Cho dù các lỗi được đưa ra từ các khối PL/SQL cục bộ hoặc được đặt tên được gọi, tiến trình quản lý ngăn xếp thì như nhau. Các lỗi được đưa

ra và được đặt trong một hàng đợi last_in, first_out (LIFO) vốn được gọi là một ngăn xếp (stack). Khi các lỗi được đưa ra được đặt trong ngăn xếp, chúng được chuyển đến các đơn vị chương trình gọi cho đến khi chúng tiến đến chương trình tận cùng bên ngoài. Chương trình tận cùng bên ngoài báo cáo ngăn xếp lỗi cho người dùng cuối. Người dùng cuối có thể là một người thật, một câu lệnh SQL hoặc một script xử lý lô bên ngoài cơ sở dữ liệu.

Script tạo một thủ tục đơn giản mà bạn có thể gọi từ một khối ngoại lệ trong từng hàm và thủ tục lưu trữ PL/SQL được đặt tên sau đó trong phần này:

-- This is found in exception1.sql on the publisher's web site.

```
CREATE OR REPLACE PROCEDURE handle_errors
( object_name IN VARCHAR2
, module_name      IN  VARCHAR2 := NULL
, table_name       IN  VARCHAR2 := NULL
, sql_error_code   IN  NUMBER := NULL
, sql_error_message IN  VARCHAR2 := NULL
, user_error_message IN  VARCHAR2 := NULL ) IS
-- Define a local exception.
    raised_error EXCEPTION;
-- Define a collection type and initialize it.
    TYPE error_stack IS TABLE OF VARCHAR2(80);
    errors ERROR_STACK := error_stack();
-- Define a local function to verify object type.
    FUNCTION object_type
    ( object_name_in IN VARCHAR2 )
RETURN VARCHAR2 IS
    return_type VARCHAR2(12) := 'Unidentified';
BEGIN
    FOR i IN ( SELECT object_type
FROM user_objects
WHERE object_name = object_name_in ) LOOP
        return_type := i.object_type;
END LOOP;
    RETURN return_type;
END object_type;
```

```

BEGIN
-- Allot space and assign a value to collection.
errors.EXTEND;
errors(errors.COUNT) := object_type(object_name) || ' [' || object_name || ']';
-- Substitute actual parameters for default values.
IF module_name IS NOT NULL THEN
    errors.EXTEND;
    errors(errors.COUNT) := 'Module Name: [' || module_name || ']';
END IF;
    IF table_name IS NOT NULL THEN
        errors.EXTEND;
        errors(errors.COUNT) := 'Table Name: [' || table_name || ']';
    END IF;
    IF sql_error_code IS NOT NULL THEN
        errors.EXTEND;
        errors(errors.COUNT) := 'SQLCODE Value: [' || sql_error_code || ']';
    END IF;
        IF sql_error_message IS NOT NULL THEN
            errors.EXTEND;
            errors(errors.COUNT) := 'SQLERRM Value: [' || sql_error_message ||
']';
        END IF;
            IF user_error_message IS NOT NULL THEN
                errors.EXTEND;
                errors(errors.COUNT) := user_error_message;
            END IF;
                errors.EXTEND;
                errors(errors.COUNT) := '-----';
                RAISE raised_error;
    EXCEPTION
        WHEN raised_error THEN
            FOR i IN 1..errors.COUNT LOOP
                dbms_output.put_line(errors(i));
            END LOOP;
            RETURN;
        END;
END;

```


/

Chữ ký thủ tục lưu trữ bao gồm các tham số hình thức tùy chọn. Điều này làm cho thủ tục `handle_errors` linh hoạt hơn. Có một hàm cục bộ đón bắt và kiểm chứng các định nghĩa nguồn đối tượng. Thủ tục test để tìm những giá trị không rỗng (not-null) trước khi xử lý các giá trị thực sự được chuyển qua các tham số hình thức. Phương thức (`EXTEND ()`) tạo không gian trước khi gán các giá trị vào các danh sách. Phương thức là một phần của Oracle 11g CollectionAPI và được đề cập trong chương 7.

Ba thủ tục sau đây được tạo theo thứ tự giảm dần do những sự phụ thuộc của chúng. Thủ tục `error_level1` gọi thủ tục `error_level2` mà sau đó gọi thủ tục `error_level3`. Bạn xây dựng những thủ tục này bằng script sau đây:

```
-- This is found in exception2.sql on the publisher's web site.
```

```
CREATE OR REPLACE PROCEDURE error_level3 IS
    one_character      VARCHAR2(1);
    two_character      VARCHAR2(2) := 'AB';
    local_object       VARCHAR2(30) := 'ERROR_LEVEL3';
    local_module       VARCHAR2(30) := 'MAIN';
    local_table        VARCHAR2(30) := NULL;
    local_user_message VARCHAR2(80) := NULL;
BEGIN
    one_character := two_character;
EXCEPTION
    WHEN others THEN
        handle_errors( object_name => local_object
                      , module_name => local_module
                      , sql_error_code => SQLCODE
                      , sql_error_message => SQLERRM );
    RAISE;
END error_level3;
/

CREATE OR REPLACE PROCEDURE error_level2 IS
    local_object       VARCHAR2(30) := 'ERROR_LEVEL2';
    local_module       VARCHAR2(30) := 'MAIN';
    local_table        VARCHAR2(30) := NULL;
    local_user_message VARCHAR2(80) := NULL;
```

```

BEGIN
    error_level3();
EXCEPTION
    WHEN others THEN
        handle_errors( object_name => local_object
                        , module_name => local_module
                        , sql_error_code => SQLCODE
                        , sql_error_message => SQLERRM );
        RAISE;
END error_level2;
/

CREATE OR REPLACE PROCEDURE error_level1 IS
    local_object      VARCHAR2(30) := 'ERROR_LEVEL1';
    local_module      VARCHAR2(30) := 'MAIN';
    local_table       VARCHAR2(30) := NULL;
    local_user_message VARCHAR2(80) := NULL;
BEGIN
    error_level2();
EXCEPTION
    WHEN others THEN
        handle_errors( object_name => local_object
                        , module_name => local_module
                        , sql_error_code => SQLCODE
                        , sql_error_message => SQLERRM );
        RAISE;
END error_level1;
/

```

Script xây dựng ba thủ tục lưu trữ. Chúng gọi nhau theo thứ tự đảo ngược cho đến khi thủ tục tận cùng bên trong đưa ra một ngoại lệ. Bạn có thể test việc lan truyền và định dạng ngăn xếp lỗi bằng cách chạy chương trình thử nghiệm sau đây:

```

BEGIN
    error_level1;
END;
/

```

Bạn sẽ nhận được ngăn xếp lỗi sau đây:

```
PROCEDURE [ERROR_LEVEL3]
Module Name: [MAIN]
SQLCODE Value: [-6502]
SQLERRM Value: [ORA-06502: PL/SQL: numeric or value error: character ...
```

```
PROCEDURE [ERROR_LEVEL2]
Module Name: [MAIN]
SQLCODE Value: [-6502]
SQLERRM Value: [ORA-06502: PL/SQL: numeric or value error: character ...
```

```
PROCEDURE [ERROR_LEVEL1]
Module Name: [MAIN]
SQLCODE Value: [-6502]
SQLERRM Value: [ORA-06502: PL/SQL: numeric or value error: character ...
```

```
begin
```

```
*
```

```
ERROR at line 1:
```

```
ORA-06502: PL/SQL: numeric or value error: character string buffer too small
```

```
ORA-06512: at "PLSQL.ERROR_LEVEL1", line 14
```

```
ORA-06512: at line 2
```

Bây giờ bạn đã xem cách định dạng ngăn xếp lỗi trong PL/SQL để minh họa một vết ngăn xếp ngoại lệ thông qua các thủ tục được đặt tên. Phương thức đòi hỏi một chút nỗ lực, nhưng rõ ràng minh họa cách bạn tìm đường dẫn truyền lan để truy vết, chẩn đoán và sửa chữa các sự cố trong mã dữ liệu hoặc mã ứng dụng.

Định dạng ngăn xếp lỗi

Phần này trình bày cách định dạng việc quản lý ngăn xếp lỗi bằng các hàm trong gói DBMS_UTILITY. Đã có một tham số hình thức `user_error_message` trong thủ tục `handle_errors` mà đã không được sử dụng. Bạn sử dụng nó để quản lý kết quả từ hàm `FORMAT_ERROR_BACKTRACE` của gói DBMS_UTILITY.

Thủ tục `handle_errors` vẫn không đổi trong phần thảo luận sau đây. Tuy nhiên, ba thủ tục minh họa việc truyền ngoại lệ đã thay đổi đôi chút như như được ghi chú tiếp theo:


```

        RAISE;
    END error_level2;
/
CREATE OR REPLACE PROCEDURE error_level1 IS
    local_object          VARCHAR2(30) := 'ERROR_LEVEL1';
    local_module          VARCHAR2(30) := 'MAIN';
    local_table           VARCHAR2(30) := NULL;
    local_user_message    VARCHAR2(200) := NULL;
BEGIN
    error_level2();
EXCEPTION
    WHEN others THEN
        handle_errors(object_name => local_object
                      ,module_name => local_module
                      ,sql_error_code => SQLCODE
                      ,sql_error_message => SQLERRM
                      ,user_error_message => DBMS_UTILITY.FORMAT_
                                             ERROR_BACKTRACE);

        RAISE;
    END error_level1;
/

```

Như ví dụ trước, script xây dựng ba thủ tục lưu trữ. Chúng gọi nhau theo trình tự đảo ngược cho đến khi thủ tục tận cùng bên trong đưa ra một ngoại lệ. Bạn có thể test việc lan truyền và định dạng một ngăn xếp lỗi bằng cách chạy chương trình thử nghiệm sau đây:

```

BEGIN
    error_level;
END;
/

```

Bạn sẽ nhận được ngăn xếp lỗi được định dạng sau đây:

```

PROCEDURE [ERROR_LEVEL3]
Module Name: [MAIN]
SQLCODE Value: [-6502]
SQLERRM Value: [ORA-06502: PL/SQL: numeric or value error: character ...
ORA-06512: at "PLSQL.ERROR_LEVEL3", line 9

```

```

PROCEDURE [ERROR_LEVEL2]
Module Name: [MAIN]
SQLCODE Value: [-6502]
SQLERRM Value: [ORA-06502: PL/SQL: numeric or value error: character ...
ORA-06512: at "PLSQL.ERROR_LEVEL3", line 17
ORA-06512: at "PLSQL.ERROR_LEVEL2", line 7

```

```

PROCEDURE [ERROR_LEVEL1]
Module Name: [MAIN]
SQLCODE Value: [-6502]
SQLERRM Value: [ORA-06502: PL/SQL: numeric or value error: character ...
ORA-06512: at "PLSQL.ERROR_LEVEL2", line 15
ORA-06512: at "PLSQL.ERROR_LEVEL1", line 7

```

```
BEGIN
```

```
*
```

```
ERROR at line 1:
```

```
ORA-06502: PL/SQL: numeric or value error: character string buffer too small
```

```
ORA-06512: at "PLSQL.ERROR_LEVEL1", line 15
```

```
ORA-06512: at line 2
```

Hàm gói `DBMS_UTILITY.FORMAT_ERROR_BACKTRACE` cung cấp cho bạn một công cụ hiệu quả hơn để truy vết, chẩn đoán và sửa chữa các sự cố. Phần gây nhầm chán duy nhất là tương hợp các số dòng của các ngoại lệ với số dòng của các chương trình lưu trữ. Bạn có thể làm điều này bằng cách tận dụng từ điển dữ liệu.

Ví dụ, nếu bạn thích tìm lỗi nguồn đã xảy ra trong dòng 12 trong thủ tục `error_level3`, query tìm dòng mã chịu trách nhiệm:

```

COL line FORMAT 999
COL text FORMAT A60
SELECT      line
           , text
FROM        user_source
WHERE       name = 'ERROR_LEVEL3'
AND         line = 9;

```

Sau đây là kết quả:

```
LINE    TEXT
```

```
9 one_character := two_character;
```

Hàm `FORMAT_ERROR_BACKTRACE` trong gói `DBMS_UTILITY` cho bạn nhận dạng nhanh vị trí của một lỗi. Bây giờ bạn biết cách quản lý các ngăn xếp lỗi có hoặc không có hàm `FORMAT_ERROR_BACKTRACE`.

Quản lý ngoại lệ Trigger cơ sở dữ liệu

Các trigger cơ sở dữ liệu là các chương trình được điều khiển bằng sự kiện (event). Nếu bạn không quen thuộc với các trigger cơ sở dữ liệu, hãy xem chương 10. Các trigger được kích hoạt khi một đơn vị chương trình giao tác gọi một đối tượng cơ sở dữ liệu như một table hoặc view. Các trigger cơ sở dữ liệu đôi khi gọi các hàm lưu trữ, thủ tục và gói (package) khác. Khi các trigger gọi các đối tượng lưu trữ khác, những đơn vị chương trình đó không thể chứa bất kỳ lệnh transaction control language (TCL), như `SAVEPOINT`, `ROLLBACK` và `COMMIT`.

Các trigger cơ sở dữ liệu giải quyết hai loại vấn đề: cách xử lý các lỗi quan trọng và các lỗi không quan trọng. Bạn đưa ra các ngoại lệ và ngưng xử lý khi gặp phải các lỗi quan trọng. Bạn đưa ra và ghi chép các ngoại lệ nhưng cho phép được xử lý tiếp tục cho các lỗi không quan trọng.

Hai mục tiếp theo đề cập cách bạn quản lý các ngoại lệ quan trọng và không quan trọng trong các trigger cơ sở dữ liệu. Các chương trình mẫu là các trigger Data Manipulation Language (DML). Chúng trình bày các khái niệm mà cũng áp dụng vào các loại trigger khác. Các ví dụ sử dụng mã và dữ liệu được tìm thấy trong phần giới thiệu. Bạn có thể download nó từ web site của nhà xuất bản.

Các trigger cơ sở dữ liệu lỗi quan trọng

Các trigger cơ sở dữ liệu dùng sự thực thi bằng cách đưa ra các lỗi quan trọng khi bạn không cho phép việc xử lý tiếp tục. Các qui tắc nghiệp vụ ấn định những gì quan trọng hoặc không quan trọng. Chúng quyết định việc một giao tác có thể gây hại cho dữ liệu hay không. Bất kỳ giao tác gây hại cho dữ liệu là một lỗi quan trọng và phải được dừng trước khi nó có thể hoàn thành.

Các ràng buộc cơ sở dữ liệu thì tuyệt vời đối với việc bảo đảm tính toàn vẹn của một mô hình dữ liệu. Mô hình dữ liệu mẫu hỗ trợ một ràng buộc khoá ngoại (foreign key) trên tất cả khoá ngoại. Điều này có nghĩa bất kỳ việc chèn (insert) vào một table vi phạm ràng buộc khoá ngoại sẽ đưa ra một ngoại lệ. Tải xử lý của các ràng buộc khoá ngoại thường làm

cho chúng thực thi quá tốn kém trong các ứng dụng thực tế. Lựa chọn cho việc thay thế các ràng buộc khoá ngoại bao gồm đưa logic vào các chương trình ứng dụng. Đôi khi bạn có thể chọn đặt logic bảo vệ này trong các trigger cơ sở dữ liệu.

Các ràng buộc cơ sở dữ liệu cũng giới hạn chỉ trong những gì chúng có thể ràng buộc. Bạn không thể sử dụng một ràng buộc cơ sở dữ liệu để bảo đảm chỉ có hai signer được uỷ quyền trên một tài khoản. Các ràng buộc khoá ngoại kiểm soát các giá trị mối quan hệ trong khi các ràng buộc check (kiểm tra) và unique (duy nhất) điều khiển các giá trị dãy. Một ràng buộc khoá ngoại bảo đảm một giá trị được tìm thấy trong một danh sách các giá trị từ một cột được xác định với một ràng buộc khoá chính (primary key). Một ràng buộc check giới hạn một giá trị chỉ trong một dãy giá trị nhưng không giới hạn sự tái xuất hiện của các giá trị lặp lại trong nhiều hàng. Một ràng buộc duy nhất bảo đảm chỉ một hàng chứa bất kỳ giá trị nào đó như một giá trị khoá ngoại riêng biệt. Do đó, các ràng buộc cơ sở dữ liệu có thể chỉ ràng buộc dữ liệu đáp ứng các điều kiện nhất định và các dãy giá trị trong các table hoặc view.

Các trigger cơ sở dữ liệu cho bạn định nghĩa các qui tắc nghiệp vụ phức tạp vốn không được hỗ trợ bởi các ràng buộc cơ sở dữ liệu. Đôi khi các qui tắc nghiệp vụ rất phức tạp. Ví dụ, không có ràng buộc cơ sở dữ liệu khi một qui tắc nghiệp vụ định nghĩa rằng chỉ có thể có hai trigger được uỷ quyền. Qui tắc nghiệp vụ này nói rằng đối với mọi hàng trong table member chỉ có hai hàng được tạo quan hệ trong table contact. Chỉ các trigger cơ sở dữ liệu có thể cho bạn kiểm toán và thi hành loại ràng buộc mối quan hệ này giữa hai table.

Các ngoại lệ được đưa ra cho các lỗi quan trọng

Bạn xây dựng một trigger DML để thi hành loại quan hệ này giữa các table member và contact. Trigger có thể sử dụng một cursor nhận dạng khi nào có nhiều hàng để kích khởi một ngoại lệ động do người dùng định nghĩa. Trigger cho bạn chèn một hoặc hai hàng trong table contact nhưng không cho phép hàng thứ ba. Script sau đây thực thi trigger cho logic này:

```
-- This is found in create_contact_t1.sql on the publisher's web site.
```

```
CREATE OR REPLACE TRIGGER contact_t1
```

```
BEFORE INSERT ON contact
```

```
FOR EACH ROW
```

```
DECLARE
```

```
    CURSOR c ( member_id_in NUMBER ) IS
```

```
        SELECT    null
```

```
        FROM      contact c
```



```

,          member m
WHERE     c.member_id = m.member_id
AND      c.member_id = member_id_in
HAVING COUNT(*) > 1;

BEGIN
  FOR i IN c (:new.member_id) LOOP
    RAISE_APPLICATION_ERROR(-20001,'Already two signers.');
```

Cursor không truy tìm giá trị từ các table nhưng truy tìm một cursor một hàng chứa một giá trị rỗng. Điều này mở câu lệnh vòng lặp FOR và đưa ra ngoại lệ động do người dùng định nghĩa khi một hoạt động chèn (insert) cố thêm một hàng thứ ba vào table contact.

Câu lệnh insert này vi phạm ràng buộc miễn là cả hai script seed đã chạy:

```

INSERT INTO contact
VALUES
( contact_s1.nextval
, 1002
,(SELECT      common_lookup_id
FROM          common_lookup
WHERE         common_lookup_table = 'CONTACT'
AND          common_lookup_column = 'CONTACT_TYPE'
AND          common_lookup_type = 'CUSTOMER')
,'Sweeney','Irving','M'
, 2, SYSDATE, 2, SYSDATE);
```

Nó đưa ra ngoại lệ sau đây từ trigger contact_t 1:

```

( contact_s1.nextval
*
```

ERROR at line 2:

ORA-20001: Already two signers.

ORA-06512: at "PLSQL.CONTACT_T1", line 11

ORA-04088: error during execution of trigger 'PLSQL.CONTACT_T1'

Ngoại lệ cung cấp thông báo ngoại lệ động (dynamic exception message). Nó cũng trả về hai thông báo ngoại lệ do hệ thống tạo ra. Các

thông báo do hệ thống tạo ra cho bạn biết những gì đã đưa ra thông báo lỗi. Phương pháp này ngay lập tức truyền đạt đến người dùng cuối rằng người dùng này đã vi phạm một qui tắc nghiệp vụ.

Khuyết điểm của loại trigger này là bạn đã không đón bắt lỗi người dùng cuối. Các doanh nghiệp thường muốn vừa ngăn các lỗi vừa thu thập các hành động nhân viên. Nhiều cửa hàng video cho các bậc phụ huynh giới hạn những gì mà con cái của họ có thể thuê chẳng hạn không cho phép thuê các bộ phim MPAA R hoặc các game được đánh giá bởi ESRB M. Đôi khi trẻ em có thể cố thuê những nội dung mà cha mẹ của chúng không cho phép.

Các ngoại lệ được đưa ra và được ghi cho các lỗi quan trọng

Các trigger vừa có thể thu thập các sự kiện vừa đưa ra các ngoại quan trọng để ngưng các hoạt động. Bạn sử dụng một PRAGMA khác (lệnh tiền biên dịch) để định nghĩa một trigger là độc lập. Chỉ lệnh AUTONOMOUS_TRANSACTION nói rằng trigger nên chạy trong một phạm vi giao tác riêng biệt. Điều này cho phép trigger commit một hành động sang cơ sở dữ liệu trong khi cũng khước từ câu lệnh DML vốn đã kích hoạt trigger.

Bạn cần một nơi để lưu trữ thông tin từ nỗ lực. Sử dụng script sau đây để tạo table nc_error cho mục đích đó:

```
-- This is found in create_nc_error.sql on the publisher's web site.
```

```
CREATE TABLE nc_error
( error_id          NUMBER CONSTRAINT pk_nce PRIMARY KEY
, module_name      VARCHAR2(30) CONSTRAINT nn_nce_1 NOT NULL
, table_name       VARCHAR2(30)
, class_name       VARCHAR2(30)
, error_code       VARCHAR2(9)
, sqlerror_message VARCHAR2(2000)
, user_error_message VARCHAR2(2000)
, last_update_date DATE          CONSTRAINT nn_nce_2 NOT NULL
, last_updated_by  NUMBER        CONSTRAINT nn_nce_3 NOT NULL
, creation_date    DATE          CONSTRAINT nn_nce_4 NOT NULL
, created_by       NUMBER        CONSTRAINT nn_nce_5 NOT NULL);
```

Sau khi tạo một nơi chứa các hoạt động được thử, bạn nên viết một thủ tục lưu trữ để xử lý câu lệnh insert. Điều này quan trọng vì hai lý do.

Thứ nhất logic dễ bao bọc hoặc bảo vệ khỏi những cặp mắt tò mò. Thứ hai logic sẽ không làm bẽ bộn trigger cơ sở dữ liệu.

Thủ tục `record_errors` sau đây ghi dữ liệu sang nơi chứa lỗi không quan trọng:

-- This is found in `create_record_errors.sql` on the publisher's web site.

```
CREATE OR REPLACE PROCEDURE record_errors
( module_name      IN      VARCHAR2
, table_name       IN      VARCHAR2 := NULL
, class_name       IN      VARCHAR2 := NULL
, sqlerror_code    IN      VARCHAR2 := NULL
, sqlerror_message IN      VARCHAR2 := NULL
, user_error_message IN    VARCHAR2 := NULL ) IS
-- Declare anchored record variable.
nc_error_record NC_ERROR%ROWTYPE;
BEGIN
-- Substitute actual parameters for default values.
IF module_name IS NOT NULL THEN
    nc_error_record.module_name := module_name;
END IF;
IF table_name IS NOT NULL THEN
    nc_error_record.table_name := module_name;
END IF;
IF sqlerror_code IS NOT NULL THEN
    nc_error_record.sqlerror_code := sqlerror_code;
END IF;
IF sqlerror_message IS NOT NULL THEN
    nc_error_record.sqlerror_message := sqlerror_message;
END IF;
IF user_error_message IS NOT NULL THEN
    nc_error_record.user_error_message := user_error_message;
END IF;
-- Insert non-critical error record.
INSERT INTO nc_error
VALUES
( nc_error_s1.nextval
```

```

, nc_error_record.module_name
, nc_error_record.table_name
, nc_error_record.class_name
, nc_error_record.sqlerror_code
, nc_error_record.sqlerror_message
, nc_error_record.user_error_message
, 2
, SYSDATE
, 2
, SYSDATE);
EXCEPTION
  WHEN others THEN
    RETURN;
END;
/

```

Chữ ký thủ tục lưu trữ bao gồm các tham số hình thức tùy chọn. Điều này làm cho thủ tục `record_errors` trở nên linh hoạt hơn. Có một hàm cục bộ thu thập và kiểm chứng các định nghĩa nguồn đối tượng.

Bạn có thể thực hiện một vài thay đổi đối với trigger `contact_t 1` và định nghĩa một trigger mới, bảo đảm việc ghi lại sự nỗ lực trong khi không cho phép hành động DML. Trigger `contact_t 2` chứa những thay đổi này và định nghĩa của nó là

```

CREATE OR REPLACE TRIGGER contact_t2
BEFORE INSERT ON contact
FOR EACH ROW
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
  CURSOR c ( member_id_in NUMBER ) IS
    SELECT    null
    FROM      contact c
    ,         member m
    WHERE c.member_id = m.member_id
    AND c.member_id = member_id_in
    HAVING COUNT(*) > 1;
BEGIN
  FOR i IN c (:new.member_id) LOOP

```

```

        record_errors( module_name => 'CREATE_CONTACT_T2'
                      , table_name => 'MEMBER'
                      , class_name => 'MEMBER_ID [ | | :new.contact_id | | ]'
                      , sqlerror_code => 'ORA-20001'
                      , user_error_message => 'Too many contacts per account. ');
    END LOOP;
    COMMIT;
    RAISE_APPLICATION_ERROR(-20001,'Already two signers. ');
END;
/

```

Chương trình thêm AUTONOMOUS_TRANSACTION PRAGMA, một lệnh gọi đến thủ tục lưu trữ record_errors và một câu lệnh COMMIT. Sau đó, nó đưa ra một thông báo ngoại lệ do người dùng định nghĩa. Commit xuất hiện sau vòng lặp, đây chỉ là một cấu trúc tiện lợi để mở và đóng một cursor cho mỗi hàng một cách ngầm định. Commit chỉ ảnh hưởng đến lệnh gọi đến thủ tục record_errors. Sau khi commit, một ngoại lệ được đưa ra dùng giao tác vốn đã kích hoạt trigger.

Câu lệnh insert quen thuộc sau đây vi phạm qui tắc nghiệp vụ được đặt ra bởi trigger:

```

INSERT INTO contact
VALUES
( contact_s1.nextval
, 1002
,(SELECT      common_lookup_id
   FROM      common_lookup
   WHERE     common_lookup_table = 'CONTACT'
   AND      common_lookup_column = 'CONTACT_TYPE'
   AND      common_lookup_type = 'CUSTOMER')
,'Sweeney','Irving','M'
, 2, SYSDATE, 2, SYSDATE)

```

Nó đưa ra ngoại lệ sau đây từ trigger contact_t 2:

```

INSERT INTO contact
*
ERROR at line 1:
ORA-20001: Already two signers.
ORA-06512: at "PLSQL.CONTACT_T2", line 19

```

ORA-04088: error during execution of trigger 'PLSQL.CONTACT_T2'

Khi bạn truy vấn table `nc_error`, bạn thấy rằng nỗ lực đã được thu thập. Định dạng và query sau đây cho bạn kiểm tra dữ liệu:

```
COL module_name FORMAT A17
COL user_error_message FORMAT A30
SELECT  error_id
        ,      module_name
        ,      user_error_message
FROM    nc_error;
```

Bạn thấy kết quả sau đây:

```
ERROR_ID  MODULE_NAME  USER_ERROR_MESSAGE
```

28 CREATE_CONTACT_T3 Too many contacts per account.

Những ví dụ này đã cho bạn thấy cách tạo các trigger để ngừng xử lý. Một trigger ngừng chèn dữ liệu và đưa ra một lỗi trong khi trigger kia làm điều đó và thu thập nỗ lực chèn dữ liệu. Bạn thực thi những trigger này khi bắt buộc không được vi phạm qui tắc nghiệp vụ.

Các trigger cơ sở dữ liệu lỗi không quan trọng

Các trigger cơ sở dữ liệu làm việc khác với các lỗi không quan trọng. Chúng đưa ra và ghi các ngoại lệ nhưng cho phép việc xử lý tiếp tục cho các lỗi không quan trọng. Điều này đòi hỏi bạn cung cấp một table cơ sở dữ liệu để ghi lại các lỗi không quan trọng.

Trong phần vừa rồi, bạn đã tạo table `nc_error`. Nếu bạn bỏ qua phần đó, bạn có thể sử dụng script `create_nc_error.sql` sau đây để tạo table này. Bảng này có thể lưu trữ những nỗ lực cho các lỗi quan trọng và không quan trọng. Định nghĩa table là

Name	Null?	Type
ERROR_ID	NOT NULL	NUMBER
MODULE_NAME	NOT NULL	VARCHAR2 (30)
TABLE_NAME		VARCHAR2 (30)
CLASS_NAME		VARCHAR2 (30)
SQLERROR_CODE		VARCHAR2 (9)
SQLERROR_MESSAGE		VARCHAR2 (2000)
USER_ERROR_MESSAGE		VARCHAR2 (2000)
LAST_UPDATED_BY	NOT NULL	NUMBER
LAST_UPDATE_DATE	NOT NULL	DATE
CREATED_BY	NOT NULL	NUMBER
CREATION_DATE	NOT NULL	DATE

Cùng một thủ tục `record_errors` đã được định nghĩa để quản lý các nỗ lực lỗi quan trọng làm việc với các trigger event quan trọng và không quan trọng. Đây không phải là biến cố. Không có câu lệnh `COMMIT` trong thủ tục `record_errors` và do đó bạn có thể gọi nó trong các trigger phạm vi giao tác độc lập hoặc phụ thuộc. Định nghĩa thủ tục là

PROCEDURE <code>record_errors</code>	Argument Name	Type	In/Out	Default?
	<code>MODULE_NAME</code>	<code>VARCHAR2</code>	<code>IN</code>	
	<code>TABLE_NAME</code>	<code>VARCHAR2</code>	<code>IN</code>	<code>DEFAULT</code>
	<code>CLASS_NAME</code>	<code>VARCHAR2</code>	<code>IN</code>	<code>DEFAULT</code>
	<code>SQLERROR_CODE</code>	<code>VARCHAR2</code>	<code>IN</code>	<code>DEFAULT</code>
	<code>SQLERROR_MESSAGE</code>	<code>VARCHAR2</code>	<code>IN</code>	<code>DEFAULT</code>
	<code>USER_ERROR_MESSAGE</code>	<code>VARCHAR2</code>	<code>IN</code>	<code>DEFAULT</code>

Chữ ký thủ tục lưu trữ bao gồm những tham số hình thức tùy chọn. Chúng làm cho thủ tục `record_errors` trở nên linh hoạt hơn. Có một hàm cục bộ thu thập và kiểm chứng các định nghĩa nguồn đối tượng.

Sau khi tạo thủ tục mới, bạn có thể chạy script tạo trigger `contact_t 3`. Thủ tục lưu trữ giúp bạn tránh khỏi những cặp mắt tò mò cách các lỗi không quan trọng được xử lý như thế nào và nó không làm xáo trộn trigger cơ sở dữ liệu.

Script `create_contact_t 3.sql` tự động xóa các trigger `contact_t 1` và / hoặc `contact_t 2` khi chúng hiện hữu. Lý do cho biện pháp phòng ngừa này là bạn không thể bảo đảm trigger nào kích khởi trước tiên khi có những trigger. Bạn muốn bảo đảm những gì bạn đang test. Trong trường hợp này, bạn test một trigger xử lý lỗi không quan trọng.

Script tạo trigger như sau:

```
-- This is found in create_contact_t3.sql on the publisher's web site.
```

```
CREATE OR REPLACE TRIGGER contact_t3
BEFORE INSERT ON contact
FOR EACH ROW
DECLARE
    CURSOR c ( member_id_in NUMBER ) IS
        SELECT null
        FROM    contact c
              , member m
        WHERE   c.member_id = m.member_id
        AND     c.member_id = member_id_in
```

```

HAVING COUNT(*) > 1;
BEGIN
  FOR i IN c (:new.member_id) LOOP
    record_errors( module_name => 'CREATE_CONTACT_T2'
                  , table_name => 'MEMBER'
                  , class_name => 'MEMBER_ID (' || :new.contact_id || ')'
                  , sqlerror_code => 'ORA-20001'
                  , user_error_message => 'Too many contacts per account.');
```

END LOOP;

```

END;
/
```

Trigger gọi thủ tục `record_errors` chèn dữ liệu vào table đích. Không có commit trong thủ tục `record_errors` bởi vì nó được thiết kế để làm việc chỉ với một trigger hoặc khối PL/SQL khác vốn quản lý phạm vi giao tác và thực thi một câu lệnh COMMIT.

Bây giờ bạn có thể tái sử dụng câu lệnh INSERT quen thuộc cho table contact:

```

INSERT INTO contact
VALUES
( contact_s1.nextval
, 1002
,(SELECT common_lookup_id
FROM      common_lookup
WHERE     common_lookup_table = 'CONTACT'
AND       common_lookup_column = 'CONTACT_TYPE'
AND       common_lookup_type = 'CUSTOMER')
, 'Sweeney', 'Irving', 'M'
, 2, SYSDATE, 2, SYSDATE);
```

Lần này trigger không đưa ra bất kỳ ngoại lệ. Nó thêm một hàng thứ ba và vi phạm qui tắc nghiệp vụ không quan trọng. Câu lệnh INSERT đã kích hoạt trigger và trigger gọi thủ tục và ghi dữ liệu lỗi. Mọi thứ đã xảy ra khi bạn commit dữ liệu sau câu lệnh INSERT. Lỗi không quan trọng được quản lý dưới dạng một giao tác phụ thuộc bên trong phạm vi giao tác của hoạt động chèn gốc vào table contact.

Tóm tắt

Chương này đã trình bày cách quản lý lỗi PL/SQL. Chương đã mô tả những điểm khác biệt giữa các lỗi biên dịch và các lỗi thời gian chạy. Bạn cũng đã học về hành vi không được xử lý của các lỗi runtime vốn xảy ra trong các khối khai báo và cách xử lý các lỗi được đưa ra trong cả khối thực thi và khối ngoại lệ.

PHẦN II

Lập trình PL/SQL

Chương 6: Các hàm và thủ tục

Chương 7: Các tập hợp

CHƯƠNG 6

CÁC HÀM VÀ THỦ TỤC

Như bạn đã thấy trong các chương trước, có hai loại thường trình con: các hàm và thủ tục. Bạn sử dụng các thường trình con này xây dựng các thư viện tầng cơ sở dữ liệu để đóng gói chức năng ứng dụng mà sau đó được đồng định vị trên tầng cơ sở dữ liệu nhằm đạt được hiệu quả.

Chương này bao gồm các mục sau đây:

- Cấu trúc hàm và thủ tục
- Phạm vi giao tác
- Các thường trình con gọi
- Các hàm
 - Các tùy chọn tạo
 - Các hàm chuyển theo giá trị
 - Các hàm chuyển theo tham chiếu
- Các thủ tục
 - Các thủ tục chuyển theo giá trị
 - Các hàm chuyển theo tham chiếu

Oracle 11g hỗ trợ các thường trình con (subroutine) được lưu trữ dưới dạng các hàm và thủ tục (procedure) trong cơ sở dữ liệu. Chúng là các khối PL/SQL được đặt tên. Bạn có thể triển khai chúng dưới dạng các thường trình con độc lập hoặc dưới dạng các thành mục (component) trong các gói (package). Các package và các loại đối tượng có thể chứa các hàm và thủ tục. Các khối nặc danh (anonymous block) cũng có thể có các hàm và thủ tục cục bộ được định nghĩa trong các khối khai báo

(declaration block) của chúng. Bạn cũng có thể xếp lồng các hàm và thủ tục bên trong các hàm và thủ tục khác.

Bạn xuất (publish) các hàm và thủ tục dưới dạng các đơn vị độc lập hoặc trong các package và các loại đối tượng. Điều này có nghĩa chúng được định nghĩa trong thông số package hoặc loại đối tượng chứ không phải mục thân của package hoặc mục thân loại đối tượng. Chúng là các thường trình con cục bộ khi bạn định nghĩa các hàm hoặc thủ tục bên trong các mục thân package hoặc mục thân loại đối tượng. Các thường trình con cục bộ không phải là các thường trình con được xuất. Tương tự, các thường trình con được định nghĩa trong khối khai báo của các chương trình khối nặc danh là các thường trình con cục bộ.

Bạn triển khai các tập hợp hàm và thủ tục liên quan trong các package và loại đối tượng. Các package và các loại đối tượng đóng vai trò như là đối với Các package cũng cho bạn quá tải (overload) các hàm và thủ tục. Chương 9 đề cập đến các package.

Các loại đối tượng do người dùng định nghĩa là những kiểu dữ liệu SQL. Bên trong các loại đối tượng, các hàm và thủ tục có thể được định nghĩa là các thường trình con cấp class (lớp) hoặc instance. Các hàm và thủ tục lớp là các thường trình con tĩnh (static subroutine), và bạn có thể truy cập chúng giống như cách bạn sử dụng các hàm và thủ tục trong các package. Các thường trình con cấp instance chỉ có thể truy cập được khi bạn tạo một instance của một loại đối tượng. Chương 14 đề cập về các loại đối tượng.

Các mục được trình bày theo trình tự để xây dựng một nền tảng về những khái niệm. Nếu bạn muốn nhảy về phía trước, việc xem qua từ đầu sẽ giúp bạn hiểu rõ các mục sau.

Cấu trúc hàm và thủ tục

Như được mô tả trong chương 3, các hàm và thủ tục là các khối PL/SQL được đặt tên. Bạn cũng có thể gọi chúng là các thường trình con (subroutine) hoặc chương trình con (subprogram). Chúng có các tiêu đề hoặc câu lệnh DECLARE. Tiêu đề (header) định nghĩa tên hàm hoặc thủ tục, một danh sách các tham số hình thức và một kiểu dữ liệu trả về cho các hàm. Các tham số hình thức định nghĩa các biến mà bạn có thể gửi đến các thường trình con khi bạn gọi chúng. Bạn có thể sử dụng các tham số hình thức và các biến cục bộ bên trong các hàm và thủ tục. Trong khi các hàm trả về một kiểu dữ liệu, các thủ tục thì không. Tối thiểu các thủ tục không liệt kê một kiểu dữ liệu trả về một cách hình thức bởi vì chúng trả về một void. Void được định nghĩa một cách tường minh trong các ngôn ngữ lập trình khác như C, C#, Java và C++. Các thủ tục có thể trả về các giá trị thông qua các biến danh sách tham số hình thức của chúng khi chúng được chuyển theo tham chiếu.

Có bốn loại thường trình con chung chung trong các ngôn ngữ lập trình. Bốn loại được định nghĩa bởi hai hành vi, cho dù chúng trả về một giá trị hình thức hay không và các danh sách tham số của chúng được chuyển theo giá trị hoặc tham chiếu:

Bạn xác lập các tham số hình thức khi bạn định nghĩa các thường trình con. Bạn gọi các thường trình con với các tham số thật sự. Các tham số hình thức định nghĩa danh sách các biến có thể có và các vị trí và kiểu dữ liệu của chúng. Các tham số hình thức không gán các giá trị ngoại trừ giá trị mặc định, điều này làm cho một tham số trở thành một tham số tùy chọn. Các tham số thật sự là những giá trị mà bạn cung cấp cho các thường trình con khi gọi chúng. Bạn có thể gọi các thường trình con mà không có một tham số thật sự khi tham số hình thức có một giá trị mặc định. Các thường trình con có thể được gọi mà không có các tham số thật sự nếu tất cả tham số hình thức của chúng được định nghĩa là tùy chọn.

Các thường trình con là các hộp đen (black box). Chúng được gọi như thế bởi vì các hộp đen che giấu những chi tiết thực thi của chúng và chỉ xuất bản những gì bạn có thể gửi vào chúng hoặc nhận từ chúng. Bảng 6.1 mô tả và minh họa những thường trình con này.

Các thường trình con là các hàm khi chúng trả về kết quả và những thủ tục khi chúng không trả về kết quả. Các hàm trả về kết quả dưới dạng các giá trị được thể hiện dưới dạng các kiểu dữ liệu SQL hoặc PL/SQL. Chương 3 mô tả các đặc điểm của các kiểu dữ liệu PL/SQL và phụ lục B thảo luận các kiểu dữ liệu SQL. Các hàm chuyển theo giá trị đôi khi được gọi là các biểu đồ bởi vì bạn gửi những giá trị vốn được trả về dưới dạng một kết quả. Khi kiểu dữ liệu trả về là một kiểu SQL, bạn có thể gọi hàm bên trong một câu lệnh SQL.

Bảng 6.1 Danh sách các loại thường trình con

Mô tả thường trình con

Các hàm chuyển theo giá trị:

Chúng nhận các bản sao của những giá trị khi chúng được gọi. Những hàm này trả về một biến xuất sau khi hoàn thành. Biến xuất có thể là một biến vô hướng hoặc biến phức hợp. Chúng cũng có thể thực hiện các hoạt động bên ngoài như các câu lệnh SQL cho cơ sở dữ liệu.

Các hàm chuyển theo tham chiếu:

Chúng nhận các tham chiếu dẫn

Hình minh họa thường trình con

sang các biến khi chúng được gọi. Các tham chiếu là những tham số thật sự cho hàm. Như những hàm khác, chúng trả về một giá trị xuất vốn có thể là một biến vô hướng hoặc biến phức hợp. Không giống như các hàm làm việc với các giá trị, loại hàm này cũng có thể thay đổi các giá trị của những tham số thật sự. Chúng trả về các tham chiếu tham số thật sự sau khi hoàn thành đến chương trình gọi. Chúng cũng có thể thực hiện những hoạt động bên ngoài, như các câu lệnh SQL cho cơ sở dữ liệu.

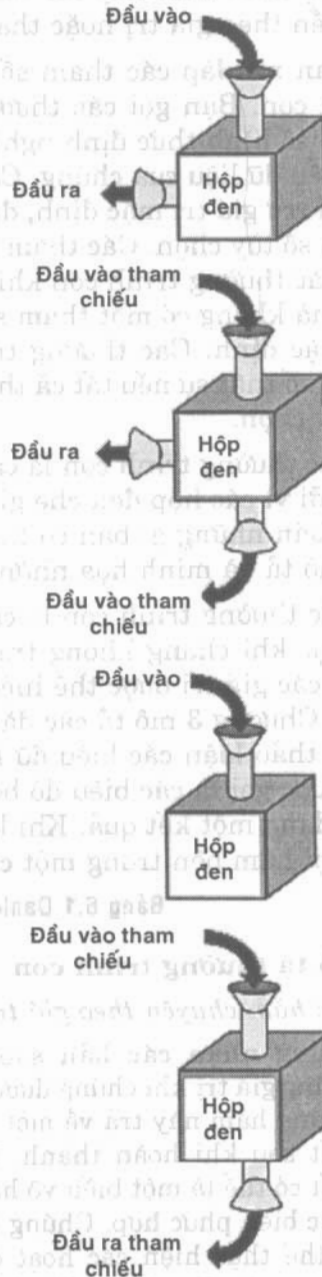
Các thủ tục chuyển theo giá trị:

Chúng nhận các bản sao của những giá trị khi chúng được gọi. Các thủ tục không trả về một biến xuất. Chúng chỉ thực hiện các hoạt động bên trong trên các biến cục bộ hoặc các hoạt động bên ngoài, như các câu lệnh SQL cho cơ sở dữ liệu.

Các thủ tục chuyển theo tham chiếu:

Chúng nhận các tham chiếu dẫn sang các biến khi chúng được gọi. Các thủ tục không trả về một biến xuất. Như các hàm chuyển theo tham chiếu, chúng có thể thay đổi giá trị của các tham số thật sự. Chúng trả các tham chiếu tham số thật sự sau khi hoàn thành trở về chương trình gọi. Chúng cũng có thể thực hiện các hoạt động bên ngoài như các câu lệnh SQL cho cơ sở dữ liệu.

Bạn có thể sử dụng các hàm làm các toán hạng phải cho các phép gán bởi vì kết quả của chúng là một giá trị của một kiểu dữ liệu được định nghĩa trong catalog cơ sở dữ liệu. Các hàm chuyển theo giá trị và chuyển theo tham chiếu hoàn thành vai trò này như nhau bên trong các khối

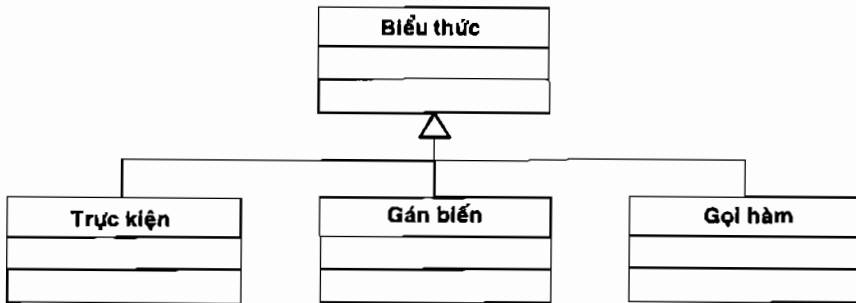


PL/SQL. Bạn có thể sử dụng các hàm chuyển theo tham chiếu trong các câu lệnh SQL chỉ khi bạn quản lý các tham số thật sự trước và sau khi gọi hàm. Bạn cũng có thể sử dụng câu lệnh CALL với mệnh đề INTO để trả về các kiểu dữ liệu SQL từ các hàm.

Hình 6.1. minh họa cách bạn có thể gán giá trị trả về từ một hàm trong một khối PL/SQL. Các câu lệnh SQL thường sử dụng các hàm chuyển theo giá trị bởi vì chúng không quản lý đầu ra tham chiếu. Hầu hết các lệnh gọi hàm SQL gởi các cột hoặc trực kiện dưới dạng các tham số thật sự và mong đợi một giá trị trả về vô hướng. Một lệnh gọi hàm SQL mô phỏng một biểu đồ SQL vốn là một query SQL trả về chỉ một cột và một hàng.

Các biểu thức PL/SQL là gì?

Các biểu đồ là những giá trị như các trực kiện ký tự, ngày tháng, số và chuỗi. Ngoài những giá trị trực kiện, các biểu thức là các phép gán biến hoặc các giá trị trả về hàm. Hình minh họa UML sau đây trình bày một số loại biểu thức.



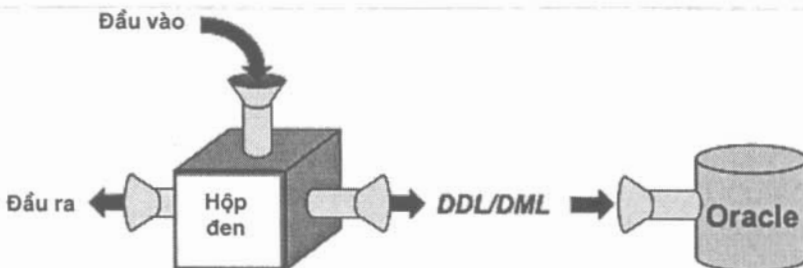
Mặc dù thuật ngữ “biểu thức” có thể gây bối rối, nhưng nói chung nó nói đến giá trị trả về từ một lệnh gọi hàm trong PL/SQL. Trong SQL, bạn có thể gặp phải một biểu thức SQL, đây là một nhãn khác cho một subquery vô hướng. Các subquery vô hướng trả về một hàng với một giá trị cột đơn. Quy tắc đơn giản là một biểu thức luôn có nghĩa là một giá trị hoặc một thứ gì đó có một giá trị hoặc trả về một giá trị.



Hình 6.1 Gán một kết quả hàm

Các thủ tục không thể được sử dụng làm các toán hạng phải. Các thủ tục cũng phải có phạm vi thời gian chạy (run-time scope) được xác lập bên trong một khối PL/SQL gọi. Bạn không thể gọi các thủ tục trong các câu lệnh SQL. Tuy nhiên, bạn có thể sử dụng các câu lệnh CALL hoặc EXECUTE để chạy các thủ tục trong SQL*Plus. Các thủ tục cũng là những đơn vị độc lập, trong khi các hàm chỉ có thể chạy như là một mục của một phép gán, một sự lượng giá so sánh hoặc câu lệnh SQL.

Các hàm hoặc thủ tục PL/SQL cũng có thể chạy các câu lệnh SQL bên trong các hộp đen của chúng. Những hành động này không được biểu thị trong các sơ đồ trước. Hình 6.2 minh họa một hàm chuyển theo giá trị được chỉnh sửa vốn thật sự cập nhật cơ sở dữ liệu. Điều này trở nên phức tạp hơn cho các hàm chuyển theo tham chiếu bởi vì chúng có một đầu ra, đầu ra tham chiếu và hành động cơ sở dữ liệu như là kết quả của một hàm. Cũng có những giới hạn về cách bởi vì sử dụng các hàm vốn thực thi các câu lệnh DML. Ví dụ, bạn không thể sử dụng một hàm thực thi câu lệnh DML bên trong một query, nếu không bạn đưa ra một lỗi ORA-14551.



Hình 6.2 Các hàm chuyển theo giá trị với đọc-ghi sang cơ sở dữ liệu

Ghi chú

Đạn có thể đưa các câu lệnh SQL vào các hàm.

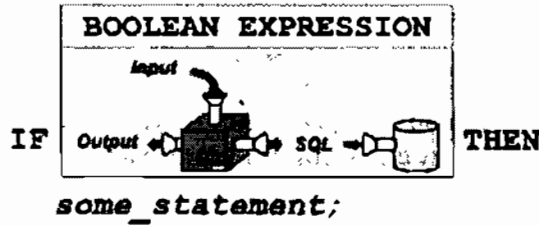
Nhiều nhà phát triển sử dụng các hàm để thực hiện các hành động cơ sở dữ liệu nhưng giới hạn các hàm chỉ trong các query hoặc phép tính. Họ làm điều này hầu như bởi vì các thủ tục trước đây đã là cách duy nhất để ghi những thay đổi sang cơ sở dữ liệu. Oracle 10g trở về sau hỗ trợ các giao tác độc lập vốn phải thay đổi các quy tắc. Bây giờ bạn chỉ nên gọi một thủ tục khi bạn có thể bảo đảm rằng nó không chạy một cách độc lập, không được gọi qua OCI hoặc không có chức năng như một giao tác phân phối. Nếu không bạn giả định rằng thủ tục đã làm việc khi tất cả những gì bạn biết là nó đã được gọi, đây là việc xử lý lạc quan (optimistic processing). Nếu bạn gọi một hàm để thực thi những thay đổi cơ sở dữ liệu và nó trả về một giá trị Boolean báo hiệu sự thành công hay thất bại, bạn đang sử dụng xử lý bi quan (pessimistic processing).

Hình 6.3 minh họa cách gọi một hàm để kiểm chứng việc hoàn thành. Đây là mẫu giao tác chung cho người ứng dụng bên ngoài. Bạn nên hết sức xem xét việc thực thi nó như là một chuẩn cho các wrapper phía server (phiên bản chuyển theo giá trị cũng làm việc với mã Java như được trình bày trong chương 15).

PL/SQL xem các hàm và thủ tục là những thường trình con chuyển theo giá trị hoặc chuyển theo tham chiếu bởi chế độ của các danh sách tham số hình thức của chúng. PL/SQL hỗ trợ ba chế độ: read-only (chỉ đọc), write-only (chỉ ghi) và read-write (đọc-ghi). Chế độ IN là chế độ mặc định và chỉ định một tham số hình thức là read-only. Chế độ OUT chỉ định một tham số write-only và chế độ IN OUT chỉ định một chế độ tham số read-write. Bảng 6.2 trình bày chi tiết về các chế độ tham số có sẵn này.

Theo mặc định, các chương trình Oracle 11g gửi các bản sao của tất cả tham số đến các thường trình con khi chúng gọi chúng. Có thể điều này dường như khác lạ bởi vì nó trái với khái niệm về các thường trình con chuyển theo tham chiếu. Tuy nhiên, đây chính xác là những gì mà bạn mong đợi cho một thường trình con chuyển theo giá trị.

Khi các thường trình con hoàn tất thành công, chúng sao chép các tham số chế độ OUT hoặc IN OUT trở về các biến ngoài. Phương pháp bảo đảm nội dung của một biến ngoài không thay đổi trước khi một thường trình con hoàn thành thành công. Điều này loại bỏ khả năng ghi một mục các tập hợp kết quả. Bởi vì một lỗi kết thúc một thường trình con. Khi một ngoại lệ được đưa ra bởi một thường trình con, bạn có một cơ hội cố gắng chuyển các biến phục hồi hoặc ghi sang các file log.



END IF;

Hình 6.3 Các hàm bi quan (pessimistic) bảo đảm các kết quả của những câu lệnh SQL.

Bảng 6.2 Các chế độ tham số thường trình con

Chế độ	Mô tả
IN	Chế độ IN là chế độ mặc định. Nó có nghĩa là một tham số hình thức chỉ đọc (read-only). Khi bạn xác lập một tham số hình thức dưới dạng read-only, bạn không thể thay đổi nó trong quá trình thực thi thường trình con. Bạn có thể gán một giá trị mặc định vào một tham số, làm cho tham số trở nên tùy chọn. Bạn sử dụng chế độ IN cho tất cả tham số hình thức khi bạn muốn định nghĩa một thường trình con chuyển theo giá trị.
OUT	Chế độ OUT nghĩa là một tham số hình thức thì chỉ ghi (write-only). Khi bạn xác lập một tham số hình thức là write-only, không có kích cỡ vật lý ban đầu được cấp phát cho biến. Bạn cấp phát kích cỡ vật lý và giá trị bên trong thường trình con. Bạn không thể gán một giá trị mặc định, điều này làm cho một tham số hình thức chế độ OUT trở nên tùy chọn. Nếu bạn cố gắng điều đó, bạn đưa ra một lỗi PLS-00230. Lỗi nói rằng một biến chế độ OUT hoặc IN OUT không thể có một giá trị mặc định. Tương tự, bạn không thể chuyển một trực kiện (literal) dưới dạng một tham số thật sự đến một biến chế độ OUT bởi vì điều đó ngăn ghi biến đầu ra. Nếu bạn cố gửi một trực kiện, bạn sẽ đưa ra một lỗi ORA-06577 với một lệnh gọi từ SQL*Plus và một lỗi PLS-00363 bên trong một khối

PL/SQL. Thông báo lỗi SQL*Plus cho biết tham số đầu ra không phải là biến liên kết (bind variable), mà là một biến session SQL*Plus. Lỗi PL/SQL cho biết biểu thức (hoặc rõ ràng hơn là trực kiện) không thể là một đích gán. Bạn sử dụng một chế độ OUT với một hoặc nhiều tham số hình thức khi bạn muốn một thường trình con chuyển theo tham chiếu chỉ ghi (write-only).

IN OUT

Chế độ IN OUT nghĩa là một tham số hình thức là đọc-ghi (read-write). Khi bạn xác lập một tham số hình thức là read-write, tham số thật sự cung cấp kích cỡ vật lý của tham số thật sự. Trong khi bạn có thể thay đổi nội dung của biến bên trong thường trình con, bạn không thể thay đổi hoặc vượt quá kích cỡ cấp phép của tham số thật sự. Bạn không thể gán một giá trị mặc định làm cho một tham số chế độ IN OUT trở thành tùy chọn. Nếu bạn cố làm điều đó, bạn đưa ra một lỗi PLS-00230. Lỗi nói rằng một biến chế độ OUT hoặc IN OUT không thể có một giá trị mặc định. Tương tự, bạn không thể chuyển một trực kiện dưới dạng một tham số thật sự đến một biến chế độ OUT bởi vì điều đó ngăn ghi biến đầu ra. Nếu bạn cố gởi một trực kiện, bạn sẽ đưa ra một lỗi ORA-06577 với một lệnh gọi từ SQL*Plus và một lỗi PLS-00363 bên trong một khối PL/SQL. Thông báo lỗi SQL*Plus cho biết tham số đầu ra không phải là biến liệt kê mà là một biến session SQL*Plus. Lỗi PL/SQL cho biết ngoại lệ (hoặc cụ thể hơn là trực kiện) không thể là một đích gán. Bạn sử dụng chế độ IN OUT với một hoặc nhiều tham số hình thức khi bạn muốn một thường trình con chuyển theo tham chiếu đọc-ghi (read-write).

Bạn có thể ghi đè hành vi mặc định là chuyển các bản sao của những biến khi gọi các hàm và thủ tục cho các giao tác cục bộ. Điều này có nghĩa bạn sử dụng ít nguồn tài nguyên hơn và thật sự chuyển một tham chiếu chứ không phải một bản sao dữ liệu. Bạn không thể ghi đè hành vi mặc định đó khi gọi đơn vị chương trình qua một liên kết cơ sở dữ liệu

hoặc lệnh gọi thủ tục ngoài. Bạn ghi đè hành vi copy bằng cách sử dụng gợi ý NOCOPY.

Gợi ý NOCOPY không ghi đè quy tắc copy khi

- Một tham số thật sự là một phần tử của một mảng kết hợp (associative array). Gợi ý NOCOPY làm việc khi bạn chuyển một mảng kết hợp hoàn chỉnh chứ không phải một phần tử đơn.
- Một tham số thật sự không bị ràng buộc NOT NULL.
- Một tham số thật sự được ràng buộc bởi thang.
- Một tham số thật sự là một cấu trúc record được định nghĩa ngầm định, nghĩa là bạn đã sử dụng anchor (neo) %ROWTYPE hoặc %TYPE.
- Một tham số thật sự là một cấu trúc record được định nghĩa ngầm định từ một vòng lặp FOR thất bại bởi vì index riêng đã giới hạn phạm vi chỉ trong cấu trúc vòng lặp.
- Một tham số thật sự đòi hỏi type casting (gán kiểu) ngầm định.

Những ví dụ trong chương này và trong suốt quyển sách sử dụng mô hình các quyền định nghĩa. Nó là giải pháp phổ biến hơn, nhưng bạn sẽ tìm thấy một sự phân tích so sánh đầy đủ về cả hai mô hình trong chương 4 của Expert Oracle PL/SQL. Chương 9 thảo luận những hàm ý thiết kế của việc sử dụng các mô hình quyền định nghĩa và quyền gọi ra.

Oracle 11g Database mang lại những thay đổi trong cách làm việc của ký hiệu tên và ký hiệu vị trí trong cả SQL và PL/SQL. Bây giờ chúng thật sự làm việc theo cùng một cách trong cả SQL và PL/SQL. Điều này sửa chữa một tật xấu đã tồn tại từ lâu trong cơ sở dữ liệu.

Dữ liệu cục bộ là gì?

Oracle phân loại dữ liệu cục bộ là các view cụ thể hóa, từ đồng nghĩa, table hoặc view. Các table và view được cụ thể hóa là dữ liệu được lưu trữ vật lý. Các view là các run-time query được tạo từ các table. View được cụ thể hóa và những view khác. Các từ đồng nghĩa (synonym) cho dữ liệu là các pointer dẫn sang các view được cụ thể hóa, từ đồng nghĩa, table hoặc view.

Bạn có thể ghi sang một view được cụ thể hóa, table, view hoặc từ đồng nghĩa cục bộ từ một chương trình con lưu trữ được đồng định vị trong cùng một schema. Các từ đồng nghĩa có thể trở vào những đối tượng trong cùng một schema hoặc một schema khác. Khi đối tượng được định nghĩa trong một schema khác, bạn phải có các đặc quyền để đọc hoặc ghi sang chúng để một từ đồng nghĩa biên dịch chính xác thành đối tượng. Một từ đồng nghĩa cục bộ có thể phân giải một schema, component selector (hoặc dấu chấm), và một tên đối tượng thành một tên schema cục bộ.

Phạm vi giao tác

Như được thảo luận trong chương 2, phạm vi giao tác (transaction scope) là một luồng thực thi - một tiến trình (process). Bạn thiết lập một session khi kết nối với cơ sở dữ liệu. Những gì bạn làm trong session chỉ nhìn thấy được đối với bạn cho đến khi commit bất kỳ thay đổi sang cơ sở dữ liệu. Sau khi bạn commit các thay đổi, những session khác có thể nhìn thấy các thay đổi mà bạn đã thực hiện.

Trong một session, bạn có thể chạy một hoặc nhiều chương trình PL/SQL. Chúng thực thi một cách nối tiếp hoặc theo trình tự. Chương trình thứ nhất có thể thay đổi dữ liệu hoặc môi trường trước khi chương trình thứ hai chạy... Điều này đúng bởi vì các session là giao tác chính. Tất cả hoạt động phụ thuộc vào một hoặc nhiều hoạt động trước. Bạn có thể commit công việc, làm cho tất cả thay đổi trở nên thường trực hoặc loại bỏ công việc, khước từ tất cả hoặc một số thay đổi.

Phạm vi giao tác khá đơn giản khi làm việc với các dòng làm việc tập trung vào tiến trình nhưng hơi phức tạp hơn khi bạn phụ thuộc vào các hàm và thủ tục. Các hàm và thủ tục có một trong hai loại phạm vi. Theo mặc định chúng được định phạm vi một cách phụ thuộc, nghĩa là chúng chạy trong phạm vi giao tác của tiến trình chính - chương trình gọi. Tuy nhiên, bạn có thể xác lập các chương trình. Điều này làm cho tất cả các thay đổi trở nên thường trực bất kể những quy tắc điều khiển chương trình chính.

Các giao tác độc lập tuyệt vời khi bạn muốn một điều gì đó xảy ra bất kể sự thành công hoặc thất bại của một điều khác nào đó. Chúng hữu dụng khi bạn muốn ghi dữ liệu trong một trigger trước khi đưa ra một ngoại lệ vốn gây ra sự thất bại của chương trình chính. Tuy nhiên, chúng nguy hiểm vì cùng một lý do. Bạn không thể vô ý ghi các trạng thái dữ liệu khi bạn không muốn chúng được ghi.

Bạn nên chú ý rằng phạm vi giao tác được điều khiển bằng việc sử dụng các lệnh SAVEPOINT, ROLLBACK, và COMMIT. Cả hai mục sau "Các hàm" và "Các thủ tục" trình bày các thường trình con độc lập.

Gọi các thường trình con

Trước Oracle 11g, bạn đã có thể sử dụng cả ký hiệu vị trí lẫn ký hiệu định danh khi gọi các thường trình con trong các đơn vị chương trình PL/SQL, nhưng không thể sử dụng ký hiệu định dạng trong các lệnh gọi SQL đến các hàm. Oracle 11g đã sửa chữa khuyết điểm đó và cũng đưa ra các lệnh gọi ký hiệu hỗn hợp.

Ký hiệu vị trí nghĩa là bạn cung cấp một giá trị cho mỗi biến trong danh sách tham số hình thức. Các giá trị phải nằm theo trình tự và cũng phải khớp với kiểu dữ liệu. Ký hiệu định danh nghĩa là bạn chuyển các

tham số thật sự bằng cách sử dụng tên tham số hình thức của chúng, toán tử kết hợp ($=>$) và giá trị. Ký hiệu định danh cho phép bạn chỉ chuyển các giá trị đến các tham số bắt buộc, nghĩa là bạn chấp nhận các giá trị mặc định cho bất kỳ tham số tùy chọn.

Các ký hiệu hỗn hợp mới nghĩa là bây giờ bạn có thể gọi các thường trình con bằng cách kết hợp ký hiệu vị trí và ký hiệu định danh. Điều này trở nên rất tiện lợi khi các danh sách tham số được định nghĩa với tất cả tham số bắt buộc trước tiên và các tham số tùy chọn tiếp theo. Nó cho bạn đặt tên hoặc tránh đặt tên các tham số bắt buộc và nó cho bạn bỏ qua các tham số tùy chọn nơi những giá trị mặc định của chúng có tác dụng. Nó không giải quyết các vấn đề ký hiệu loại trừ. Các vấn đề loại trừ xảy ra với ký hiệu vị trí khi các tham số tùy chọn được xen kẽ với các tham số bắt buộc và khi bạn gọi một số nhưng không phải tất cả tham số tùy chọn.

Hàm sau đây cho bạn thử nghiệm với những phương pháp khác nhau này. Hàm chấp nhận ba tham số tùy chọn và trả về tổng của ba số:

```
CREATE OR REPLACE FUNCTION add_three_numbers
(a NUMBER := 0, b NUMBER := 0, c NUMBER := 0) RETURN NUMBER IS
BEGIN
    RETURN a + b + c;
END;
/
```

Ba mục nhỏ đầu tiên trình bày cho bạn cách thực hiện các lệnh gọi hàm ký hiệu vị trí, định danh và hỗn hợp. Mục nhỏ cuối cùng minh họa cách thực hiện các lệnh gọi ký hiệu loại trừ.

Ký hiệu vị trí

Bạn sử dụng ký hiệu vị trí để gọi hàm như sau:

```
BEGIN
    dbms_output.put_line(add_three_numbers(3,4,5));
END;
/
```

Ký hiệu định danh

Bạn gọi hàm bằng cách sử dụng ký hiệu định danh bằng

```
BEGIN
    dbms_output.put_line(add_three_numbers(c => 4,b => 5,c => 3));
END;
/
```

Ký hiệu hỗn hợp

Bạn gọi hàm bằng cách kết hợp ký hiệu vị trí và ký hiệu định danh bằng

```
BEGIN
    dbms_output.put_line(add_three_numbers(3,c => 4,b => 5));
END;
/
```

Có một giới hạn về ký hiệu hỗn hợp. Tất cả tham số thật sự của ký hiệu vị trí phải xảy ra trước tiên và theo cùng một thứ tự như chúng được định nghĩa bởi chữ ký hàm. Bạn không thể cung cấp một giá trị vị trí sau mỗi giá trị định danh mà không đưa ra một ngoại lệ.

Ký hiệu loại trừ

Như được đề cập, bạn cũng có thể loại trừ một hoặc nhiều tham số thật sự khi các tham số hình thức được định nghĩa là tùy chọn. Tất cả tham số trong hàm `add_three_numbers` thì tùy chọn. Ví dụ sau đây chuyển một giá trị đến tham số thứ nhất bằng tham chiếu vị trí và tham số thứ ba bằng tham chiếu định danh:

```
BEGIN
    dbms_output.put_line(add_three_numbers(3,c => 4));
END;
/
```

Khi bạn chọn không cung cấp một tham số thật sự, như thể bạn đang chuyển một giá trị rỗng. Đây được gọi là ký hiệu loại trừ. Trong nhiều năm Oracle đã đề nghị bạn nên liệt kê các tham số tùy chọn sau cùng trong các chữ ký hàm và thủ tục. Họ cũng đã đề nghị rằng bạn xếp trình tự các biến tùy chọn để bạn không bao giờ phải bỏ qua một tham số tùy chọn trong danh sách. Những đề nghị này nhằm tránh các lỗi khi thực hiện các lệnh gọi ký hiệu vị trí.

Bạn không thể bỏ qua một tham số thật sự trong một lệnh gọi ký hiệu vị trí. Điều này đúng bởi vì tất cả lệnh gọi vị trí nằm theo trình tự theo kiểu dữ liệu, nhưng bạn có thể cung cấp một giá trị rỗng được phân cách bằng dấu phẩy khi bạn bỏ qua một tham số tùy chọn trong danh sách. Tuy nhiên, bây giờ Oracle 11g cho bạn sử dụng các lệnh gọi ký hiệu hỗn hợp. Bây giờ bạn có thể sử dụng ký hiệu vị trí cho danh sách các tham số bắt buộc và ký hiệu định danh cho các tham số tùy chọn. Điều này cho bạn bỏ qua các tham số tùy chọn mà không cần đặt tên cho các tham số một cách tường minh.

Ký hiệu lệnh gọi SQL

Trước đó, bạn đã chỉ có một lựa chọn. Bạn đã phải liệt kê tất cả tham số theo thứ tự vị trí của chúng bởi vì bạn không thể sử dụng tham chiếu định danh trong SQL. Điều này được sửa chữa trong Oracle 11g; bây giờ bạn có thể gọi chúng như bạn gọi từ một khối PL/SQL. Dòng sau đây minh họa ký hiệu hỗn hợp trong một lệnh gọi SQL:

```
SELECT add_three_numbers(3,c => 4,b => 5) FROM dual;
```

Như khi sử dụng các phiên bản trước, bạn chỉ có thể gọi các hàm nào có các biến chỉ chế độ IN từ các câu lệnh SQL. Bạn không thể gọi một hàm từ SQL khi bất kỳ tham số hình thức của chúng được định nghĩa là các biến chỉ chế độ IN OUT hoặc OUT mà không xử lý tham số thật sự trong SQL*Plus dưới dạng một biến liên kết session. Điều này đúng bởi vì bạn phải chuyển một tham chiếu biến khi một tham số có một chế độ OUT.

Các hàm

Như đã được đề cập, bạn có các hàm chuyển theo giá trị và chuyển theo tham chiếu trong PL/SQL. Cả hai loại hàm trả về các giá trị đầu ra. Các giá trị đầu ra của hàm có thể là bất kỳ kiểu dữ liệu SQL hoặc PL/SQL. Bạn có thể sử dụng các hàm trả về các kiểu dữ liệu SQL bên trong các câu lệnh SQL. Các hàm trả về các kiểu dữ liệu PL/SQL chỉ làm việc bên trong các khối PL/SQL.

Một ngoại lệ cho những quy tắc chung này là bạn không thể gọi một hàm lưu trữ vốn chứa một thao tác DML từ bên trong một query. Nếu bạn làm như vậy, nó đưa ra một lỗi ORA-14551 nói rằng nó không thể thực thi một DML bên trong một query. Tuy nhiên, bạn có thể gọi một hàm vốn thực thi một thao tác DML bên trong các thao tác insert, update và delete.

Các hàm cũng có thể chứa các khối định danh xếp lồng vốn là các hàm và thủ tục cục bộ. Bạn định nghĩa các khối định danh trong khối khai báo của hàm. Tương tự, bạn có thể xếp lồng các khối nặc danh trong khối thực thi.

Dòng mã sau đây minh họa một nguyên mẫu hàm khối định danh:

```
FUNCTION function_name
  [( parameter1 [IN][OUT] [NOCOPY] sql_datatype | plsql_datatype
  , parameter2 [IN][OUT] [NOCOPY] sql_datatype | plsql_datatype
  , parameter(n+1) [IN][OUT] [NOCOPY] sql_datatype | plsql_datatype )]
RETURN { sql_data_type | plsql_data_type }
  [ AUTHID [ DEFINER | CURRENT_USER ]]
```



```

[ DETERMINISTIC | PARALLEL_ENABLED ]
[ PIPELINED ]
[ RESULT_CACHE [ RELIES_ON table_name ] ] IS
declaration_statements
BEGIN
    execution_statements
    RETURN variable;
[EXCEPTION]
    exception_handling_statements
END [function_name];
/

```

Bạn gọi các hàm bằng cách cung cấp bất kỳ tham số bắt buộc dưới dạng một danh sách các đối số bên trong các dấu ngoặc đơn mở và đóng. Các dấu ngoặc đơn không bắt buộc khi các hàm không được định nghĩa với những tham số bắt buộc. Điều này khác với hầu hết những ngôn ngữ lập trình khác. Các lệnh gọi trong những ngôn ngữ khác đòi hỏi một tập hợp các dấu ngoặc đơn mở và đóng rỗng.

Nguyên mẫu cho một lệnh gọi hàm với những tham số thật sự từ SQL*Plus là

```

CALL function_name(parameter1, parameter2, parameter(n+1))
INTO target_variable_name;

```

Khi không có bất kỳ tham số hình thức bắt buộc, nguyên mẫu khác như được trình bày:

```

CALL function_name INTO target_variable_name;

```

Các phép gán bên trong các khối PL/SQL với những tham số bắt buộc trông như sau

```

target_variable_name := function_name(parameter1, parameter2, parameter(n+1));

```

Nguyên mẫu gán loại bỏ các dấu ngoặc đơn khi không cần thiết:

```

target_variable_name := function_name;

```

Bạn cũng có thể trả về một giá trị hàm dưới dạng một biểu thức và sau đó sử dụng nó làm tham số thật sự cho một hàm khác. Điều này được thực hiện bằng cách sử dụng nguyên mẫu sau đây:

```

external_function_name(function_name(parameter1, parameter2,
parameter(n+1)));

```

Có một số cấu hình tùy chọn mà bạn có thể sử dụng khi tạo các hàm. Mô hình hoạt động mặc định là các quyền định nghĩa. Bạn có thể định nghĩa một hàm để hỗ trợ một mô hình quyền gọi ra bằng cách đưa vào AUTHID as CURRENT_USER. Mô hình quyền định nghĩa chạy với các quyền của schema sở hữu và thích hợp nhất trong một mô hình điện toán tập trung. Mô hình quyền gọi ra đòi hỏi bạn duy trì nhiều bản sao của các table hoặc view trong các schema hoặc cơ sở dữ liệu khác nhau. Mô hình quyền gọi ra hỗ trợ tốt nhất các mô hình điện toán phân phối. Chương 9 thảo luận các mô hình quyền định nghĩa và quyền gọi ra.

Bạn cũng có thể bảo đảm hành vi của một hàm, điều này làm cho có thể sử dụng chúng trong các câu lệnh SQL, các index dựa vào hàm và các view được cụ thể hóa. Bạn cũng có thể cấu hình các hàm để trả về các table được pipeline trong Oracle 11g, các tập hợp kết quả chia sẻ từ cache trong SGA.

Như được thảo luận, bạn có thể định nghĩa các tham số hình thức ở một trong ba chế độ. Chúng là chế độ IN cho các tham số read-only, chế độ OUT cho các tham số write-only, và chế độ IN OUT cho các tham số read-write. Bạn xây dựng một hàm chuyển theo giá trị khi bạn định nghĩa tất cả tham số là chế độ IN, và một hàm chuyển theo tham chiếu khi bạn định nghĩa một hoặc nhiều tham số là các tham số chế độ OUT hoặc IN OUT.

Ba mục tiếp theo thảo luận cách bạn tạo các hàm. Mục thứ nhất kiểm tra các mệnh đề tùy chọn vốn cho bạn tạo các hàm cho những mục đích khác nhau. Mục thứ hai kiểm tra các hàm chuyển theo giá trị và mục thứ ba thảo luận các hàm chuyển theo giá trị.

Các tùy chọn tạo

Bạn tạo các hàm cho những câu lệnh SQL, index dựa vào hàm và view được cụ thể hóa bằng cách sử dụng các mệnh đề DETERMINISTIC hoặc PARALLEL_ENABLED. Các mệnh đề DETERMINISTIC và PARALLEL_ENABLED thay thế các chỉ lệnh tiên biên dịch RESTRICT_REFERENCES cũ hơn giới hạn những gì mà các hàm có thể làm khi chúng nằm trong các PL/SQL. Những mệnh đề mới cho bạn gán những giới hạn đó vào các hàm trong các PL/SQL và chúng cũng cho bạn gán chúng vào các hàm lưu trữ độc lập.

Các vấn đề tương thích ngược cho các hàm

Các hàm đã là những thường trình con giới hạn trước Oracle 8i (8.1.6). Bạn đã phải đơn giản chúng với sự bảo đảm hiệu suất, đây đã được gọi là mức độ thuần túy của chúng. Những bảo đảm giới hạn việc các hàm có thể đọc hoặc ghi sang các biến PL/SQL hoặc sang cơ sở dữ liệu hay không.

Những giới hạn này vẫn có thể được áp đặt trên các hàm bên trong các package bằng cách sử dụng những tùy chọn `RESTRICT_REFERENCES PRAGMA` được liệt kê trong bảng 6.3. Một `PRAGMA` là một chỉ lệnh tiền biên dịch. Bất kỳ nỗ lực nhằm sử dụng một `RESTRICT_REFERENCES PRAGMA` bên trong một hàm độc lập sẽ đưa ra một lỗi PLS-00708.

Bạn phải định nghĩa các giới hạn `PRAGMA` trong các thông số package, không phải trong các mục thân package. Chỉ nên có một `PRAGMA` mỗi hàm. Bạn có thể đưa nhiều tùy chọn vào bất kỳ chỉ lệnh tiền biên dịch `RESTRICT_REFERENCES`. Tùy chọn `TRUST` có thể được thêm vào việc giới hạn các chỉ lệnh `PRAGMA` khi bạn muốn cho phép một hàm giới hạn gọi những hàm không giới hạn khác. Tùy chọn `TRUST` tắt việc kiểm toán (auditing) cho dù các hàm được gọi có tuân theo các giới hạn của đơn vị chương trình gọi hay không - chia sẻ cùng một cấp độ thuần túy hoặc bảo đảm hiệu suất.

Ghi chú

Bạn nên xem xét thay thế những chỉ lệnh tiền biên dịch giới hạn này trong các thông số package cũ hơn bằng một mệnh đề `DETERMINISTIC` hoặc `PARALLEL_ENABLED`.

Khả năng tương thích ngược thì tốt nhưng hiếm khi kéo dài mãi mãi. Bạn nên thay thế những chỉ lệnh tiền biên dịch cũ này bằng cách định nghĩa các hàm với cú pháp mới. Điều này có nghĩa là làm cho các hàm trở thành `DETERMINISTIC` khi chúng được sử dụng bởi các index dựa vào hàm. Tương tự, bạn nên định nghĩa các hàm là `PARALLEL_ENABLED` khi chúng có thể chạy trong các thao tác được song song hóa.

Mệnh đề `PIPELINED` cho phép tạo các hàm trả về các table pipelined. Các table pipelined hành động như các cursor giả tham chiếu và được tạo bằng cách sử dụng các loại tập hợp PL/SQL được chỉnh sửa. Chúng cho bạn làm việc với các tập hợp PL/SQL gồm các cấu trúc record mà không định nghĩa chúng là các loại đối tượng do người dùng định nghĩa có thể thể hiện cụ thể (instantiable). Bạn cũng có thể đọc các tập hợp trong những câu lệnh SQL như bạn thường đọc một inline view.

Oracle 11g giới thiệu cache kết quả session chéo. Bạn thực thi tính năng này bằng cách định nghĩa các hàm với mệnh đề `RESULT_CACHE`. Cache kết quả session chéo lưu trữ những tham số thật sự và kết quả cho mỗi lệnh gọi đến những hàm này. Một lệnh gọi thứ hai đến hàm với cùng những tham số thật sự đó sẽ tìm kết quả trong cache session chéo và do đó tránh chạy lại mã. Kết quả được lưu trữ trong SGA. Khi cache kết quả hết bộ nhớ, chúng làm lão hóa các kết quả gọi hàm ít được sử dụng nhất.

Mệnh đề DETERMINISTIC

Mệnh đề DETERMINISTIC bảo đảm một hàm luôn làm việc theo cùng một cách với bất kỳ đầu vào. Loại bảo đảm này đòi hỏi một hàm không đọc hoặc ghi dữ liệu từ các nguồn bên ngoài như các package hoặc table cơ sở dữ liệu. Chỉ các hàm deterministic (tất định) làm việc trong các view được cụ thể hóa và các index dựa vào hàm. Chúng cũng là những giải pháp khuyên dùng cho các hàm do người dùng định nghĩa mà bạn dự định sử dụng trong các mệnh đề SQL, như WHERE, ORDER, BY hoặc GROUP BY; hoặc các phương thức loại đối tượng SQL như MAP hoặc ORDER.

Các hàm deterministic thường xử lý những tham số chính xác theo cùng một cách. Điều này có nghĩa cho dù những giá trị nào mà bạn gửi, hàm làm việc theo cùng một cách. Chúng cũng không có những sự phụ thuộc nội tại vào các biến package hoặc dữ liệu từ cơ sở dữ liệu. Hàm pv sau đây là hàm deterministic và tính giá trị hiện tại của một khoản đầu tư:

```
-- This is found in pv.sql on the publisher's web site.
```

```
CREATE OR REPLACE FUNCTION pv
( future_value NUMBER
,   periods NUMBER
,   interest NUMBER )
RETURN NUMBER DETERMINISTIC IS
BEGIN
RETURN future_value / ((1 + interest)**periods);
END pv;
/
```

Giả sử bạn muốn biết phải đặt bao nhiêu vào một khoản đầu tư 6% hôm nay để đạt được \$10.000 trong năm năm. Bạn test hàm này bằng cách định nghĩa một biến liên kết, sử dụng một câu lệnh CALL để đặt giá trị trong biến liên kết và truy vấn kết quả dựa vào bảng DUAL, như sau

```
VARIABLE result NUMBER
CALL pv(10000,5,6) INTO :result;
COLUMN money_today FORMAT 9,999.90
SELECT :result AS money_today FROM dual;
```

Lệnh gọi hàm sử dụng ký hiệu vị trí nhưng cũng có thể sử dụng ký hiệu định danh hoặc ký hiệu hỗn hợp. Nó in lượng giá trị hiện tại được định dạng:

MONEY_TODAY

7,472.58

Bạn sử dụng các hàm deterministic bên trong các view được cụ thể hóa và các index dựa vào hàm. Cả các view được cụ thể hóa và các index dựa vào hàm phải được tái tạo khi bạn thay đổi chi tiết hoạt động bên trong của các hàm deterministic.

Bảng 6.3 Các tùy chọn tiền biên dịch cho các hàm Package

Tùy chọn	Mô tả
RNDS	Tùy chọn RNDS bảo đảm một hàm không đọc trạng thái dữ liệu. Điều này có nghĩa bạn không thể đưa vào hàm một query SQL thuộc bất kỳ loại. Nó cũng không thể gọi bất kỳ khối định danh khác vốn bao gồm một query SQL. Một lỗi PLS-00452 được đưa ra trong quá trình biên dịch nếu có một query bên trong phạm vi chương trình của hàm vi phạm giới hạn PRAGMA.
WINDS	Tùy chọn WINDS bảo đảm một hàm không ghi trạng thái dữ liệu. Điều này có nghĩa bạn không thể đưa vào các câu lệnh SQL vốn chèn, cập nhật hoặc xóa dữ liệu. Nó cũng không thể gọi bất kỳ khối định danh khác vốn bao gồm một query SQL. Một lỗi PLS-00452 được đưa ra trong quá trình biên dịch nếu có một câu lệnh DML bên trong phạm vi chương trình của hàm vi phạm giới hạn PRAGMA.
RNPS	Tùy chọn RNPS bảo đảm một hàm không đọc trạng thái package, nghĩa là nó không đọc bất kỳ biến package. Điều này có nghĩa bạn không thể truy cập một biến package trong hàm. Nó cũng không thể gọi bất kỳ biến package đọc khối định danh khác. Một lỗi PLS-00452 được đưa ra trong quá trình biên dịch nếu có một query bên trong phạm vi chương trình của hàm vi phạm giới hạn PRAGMA.
WNPS	Các tùy chọn WNPS bảo đảm một hàm không ghi trạng thái dữ liệu, nghĩa là nó không ghi bất kỳ dữ liệu sang các biến package. Điều này có nghĩa bạn không thể

thay đổi các biến package hoặc gọi một khối định danh khác vốn thay đổi chúng. Một lỗi PLS-00452 được đưa ra trong quá trình biên dịch nếu có một câu lệnh bên trong phạm vi chương trình của hàm vi phạm giới hạn PRAGMA.

TRUST

Tùy chọn TRUST ra lệnh hàm không kiểm tra xem các chương trình được gọi có thi hành các tùy chọn RESTRICT_REFERENCES khác hay không. Lợi ích của tùy chọn này là bạn có thể từ từ di trú mã sang chuẩn mới. Những rủi ro bao gồm thay đổi hành vi hoặc hiệu suất của những câu lệnh SQL. Để tham khảo, những tùy chọn khác cũng bảo đảm các điều kiện cần thiết để hỗ trợ các index dựa vào hàm và các thao tác query song song.

Mệnh đề PARALLEL_ENABLE

PARALLEL_ENABLE cho phép chỉ định một hàm để hỗ trợ các khả năng query song song. Loại bảo đảm này đòi hỏi một hàm không đọc hoặc ghi dữ liệu từ những nguồn bên ngoài như các package hoặc table cơ sở dữ liệu. Bạn nên xem xét ẩn định các hàm là an toàn cho các thao tác song song để cải thiện thông lượng, nhưng optimizer Oracle 11g có thể chạy các hàm không được ẩn định khi nó tin là chúng an toàn cho các thao tác song song. Các phương thức Java và những chương trình C bên ngoài không bao giờ được xem là an toàn cho các thao tác song song.

Hàm sau đây hỗ trợ các thao tác SQL song song và trộn họ, tên và tên đệm vào một chuỗi đơn:

```
-- This is found in merge.sql on the publisher's web site.
CREATE OR REPLACE FUNCTION merge
( last_name VARCHAR2
, first_name VARCHAR2
, middle_initial VARCHAR2 )
RETURN VARCHAR2 PARALLEL_ENABLE IS
BEGIN
RETURN last_name || ', ' || first_name || ' ' || middle_initial;
END;
/
```

Bạn sử dụng an toàn hàm trong các query cơ sở dữ liệu như

```
SELECT merge(last_name,first_name,middle_initial) AS full_name
FROM contact
ORDER BY last_name, first_name, middle_initial;
```

Query này phụ thuộc vào mã được thảo luận trong mục giới thiệu và trả về kết quả sau đây

```
FULL_NAME
-----
Sweeney, Ian M
Sweeney, Irving M
Sweeney, Meaghan
Vizquel, Doreen
Vizquel, Oscar
Winn, Brian
Winn, Randi
```

Các thao tác song song không luôn luôn xảy ra khi bạn sử dụng gợi ý PARALLEL_ENABLE. Các thao tác song song tốn kém hơn với các tập hợp dữ liệu nhỏ. Optimizer Oracle 11g đánh giá khi nào chạy các thao tác trong chế độ song song (parallel). Đôi khi optimizer cũng chạy các hàm song song khi chúng được đánh dấu là được bật chế độ parallel. Điều này đưa ra quyết định này sau khi kiểm tra hàm có thể hỗ trợ thao tác hay không. Bật các hàm cho hoạt động song song khi chúng có đủ điều kiện là một thói quen viết mã tốt.

Mệnh đề PIPELINED

Mệnh đề PIPELINED nâng cao hiệu suất được cải thiện khi các hàm trả về các tập hợp như các table xếp lồng hoặc VARRAY. Bạn cũng sẽ nhận thấy những cải thiện hiệu suất khi trả về các cursor tham chiếu hệ thống bằng cách sử dụng mệnh đề PIPELINED. Các hàm pipelined cũng cho bạn trả về các bảng gộp (aggregate table). Các bảng gộp có chức năng như các tập hợp gồm những cấu trúc record PL/SQL. Chúng chỉ làm việc trong những câu lệnh SQL.

Mục này thảo luận các khái niệm về tập hợp (collection). Chương 6 đề cập đến các tập hợp cho những người mới làm quen với PL/SQL. Các tập hợp là các mảng (array) và danh sách các biến vô hướng và biến phức hợp. Các hàm pipelined chỉ làm việc với các tập hợp VARRAY và table xếp lồng. Hai loại tập hợp này được tạo index bằng các số trình tự. Bạn cũng có thể tạo các tập hợp gồm những loại đối tượng SQL do người dùng định nghĩa vốn được xem là các mảng một chiều gồm các số, chuỗi hoặc

ngày tháng. Chương 14 đề cập đến các loại đối tượng và bao gồm một hộp bóng (shadow box) minh họa cách sử dụng các hàm pipelined.

Mục thực thi dễ nhất của một hàm pipelined bao gồm một tập hợp các giá trị vô hướng được định nghĩa bởi một kiểu dữ liệu SQL. Bạn định nghĩa kiểu dữ liệu NUMBERS làm VARRAY của số bằng cách sử dụng lệnh sau đây:

```
CREATE OR REPLACE TYPE numbers AS VARRAY(10) OF NUMBER;
/
```

10 trong các dấu ngoặc đơn sau VARRAY xác lập số phần tử tối đa trong tập hợp. Các kiểu dữ liệu VARRAY rất tương tự như các mảng (array). Các mảng trong hầu hết các ngôn ngữ lập trình được tái tạo với một kích cỡ hoặc sự cấp phát bộ nhớ cố định.

Sau khi bạn tạo kiểu dữ liệu tập hợp, bạn mô tả nó tại dòng lệnh SQL:

```
SQL> DESCRIBE NUMBERS

NUMBERS VARRAY(10) OF NUMBER
```

Ghi chú

Khi bạn tạo các kiểu trong cơ sở dữ liệu, lệnh DDL hành động như một khối PL/SQL. Những lệnh này đòi hỏi một dấu chấm phẩy để kết thúc câu lệnh và một dấu gạch chéo tiến (/) để thực thi nó (hoặc biên dịch nó thành cơ sở dữ liệu).

Một hàm pipelined phụ thuộc vào các kiểu dữ liệu tập hợp SQL hoặc PL/SQL có sẵn. Những loại này giới hạn chỉ trong các tập hợp VARRAY hoặc table xếp lồng. Bạn có thể định nghĩa các loại tập hợp SQL của các biến vô hướng hoặc các loại đối tượng do người dùng định nghĩa.

Dòng mã sau đây định nghĩa một hàm pipelined trả về một mảng các số thứ tự:

-- This is found in create_pipelined1.sql on the publisher's web site.

```
CREATE OR REPLACE FUNCTION pipelined_numbers
RETURN NUMBERS
PIPELINED IS
list NUMBERS := numbers(0,1,2,3,4,5,6,7,8,9);
BEGIN
FOR i IN 1..list.LAST LOOP
PIPE ROW(list(i));
END LOOP;
RETURN;
```



```
END;
/
```

Hàm trả về kiểu dữ liệu SQL NUMBERS do người dùng định nghĩa từ catalog dữ liệu. Hàm khai báo một tập hợp NUMBERS bằng cách khởi tạo tập hợp. Bạn khởi tạo một tập hợp bằng cách gọi tên kiểu dữ liệu SQL do người dùng định nghĩa với một danh sách các biến vô hướng. Bên trong vòng lặp FOR, bạn gán các phần tử từ tập hợp vào pipe.

Sau đó bạn có thể truy vấn kết quả như sau:

```
SELECT * FROM TABLE(pipelined_numbers);
```

Kết quả là một cột với các số thứ tự từ 0 đến 9.

Các hàm pipelined cũng có thể sử dụng các loại tập hợp PL/SQL, miễn là bạn thực thi chúng dưới dạng các tập hợp VARRAY hoặc tập hợp table xếp lồng. Các loại tập hợp PL/SQL có thể chứa các biến vô hướng hoặc các loại đối tượng do người dùng định nghĩa như các loại SQL tương đương. Chúng cũng có thể là những tập hợp gồm các cấu trúc record. Điều này có nghĩa chúng tương tự như các cursor tham chiếu hệ thống.

Không giống như các cursor tham chiếu, chúng không thể được định nghĩa là các kiểu dữ liệu SQL hoặc PL/SQL. Chúng chỉ có thể được định nghĩa là các kiểu dữ liệu PL/SQL. Để trả về những kiểu này trong các hàm lưu trữ, chúng phải được định nghĩa bên trong một thông số package. Tối thiểu bạn phải định nghĩa loại record trong một tham số package ngay khi bạn thực thi một hàm độc lập pipelined. Chương 9 đề cập chi tiết hơn về các package.

Thông số package sau đây định nghĩa một cấu trúc record, một tập hợp của cấu trúc record và một hàm trả về loại tập hợp:

```
-- This is found in create_pipelined2.sql on the publisher's web site.
```

```
CREATE OR REPLACE PACKAGE pipelined IS
    -- Define a PL/SQL record type and Collection of the record type.
    TYPE account_record IS RECORD
    ( account VARCHAR2(10)
    , full_name VARCHAR2(42));
    TYPE account_collection IS TABLE OF account_record;

    -- Define a pipelined function.
    FUNCTION pf RETURN account_collection PIPELINED;
END pipelined;
/
```

Hàm được thực thi trong mục thân package:

-- This is found in create_pipelined2.sql on the publisher's web site.

```
CREATE OR REPLACE PACKAGE BODY pipelined IS
    -- Implement a pipelined function.
    FUNCTION pf
    RETURN account_collection
    PIPELINED IS
        -- Declare a collection control variable and collection variable.
        counter NUMBER := 1;
        account ACCOUNT_COLLECTION := account_collection();
        -- Define a cursor.
        CURSOR c IS
            SELECT          m.account_number
            , c.last_name || ' ' || c.first_name full_name
            FROM member m JOIN contact c ON m.member_id = c.member_id
            ORDER BY c.last_name, c.first_name, c.middle_initial;
        BEGIN
            FOR i IN c LOOP
                account.EXTEND;
                account(counter).account := i.account_number;
                account(counter).full_name := i.full_name;
                PIPE ROW(account(counter));
                counter := counter + 1;
            END LOOP;
            RETURN;
        END pf;
    END pipelined;
/
```

Mục thân package thực thi hàm pf. Bên trong hàm, một biến cục bộ được khai báo bằng cách sử dụng loại tập hợp PL/SQL account_collection. Các tập hợp VARRAY và tập hợp table xếp lồng là các đối tượng nội tại và chúng đòi hỏi việc xây dựng tường minh. Phương thức tạo (constructor) là tên của loại tập hợp không có bất kỳ tham số thật sự khi bạn muốn khai báo một tập hợp rỗng. Các tập hợp đòi hỏi bạn cấp phát không gian trước khi thêm các phần tử vào một tập hợp. Phương thức EXTEND cấp phát không gian cho một phần tử và sau đó những giá trị

được gán vào những thành mục của phần tử được tạo index đó. Phần tử record được thêm dưới dạng một hàng trong PIPE.

Bạn có thể gọi hàm bằng cách sử dụng tên package, component selector và tên hàm như được trình bày:

```
SELECT * FROM TABLE(pipelined.pf);
```

Câu lệnh này trả về các hàng từ cấu trúc record dưới dạng một bảng gộp:

```
ACCOUNT      FULL_NAME
-----
B293-71447   Sweeney, Ian
B293-71446   Sweeney, Irving
B293-71447   Sweeney, Meaghan
B293-71446   Vizquel, Doreen
B293-71446   Vizquel, Oscar
B293-71445   Winn, Brian
B293-71445   Winn, Randi
```

Đường như bạn bị giới hạn chỉ trong các package bởi vì đó là nơi kiểu trả về được định vị. Trong khi các kiểu dữ liệu PL/SQL không có sẵn trong từ điển dữ liệu, chúng có sẵn cho những đơn vị chương trình PL/SQL khác khi chúng được xuất trong một thông số package.

Định nghĩa hàm độc lập thực thi cùng một hàm pipelined bên ngoài package:

```
-- This is found in create_pipelined2.sql on the publisher's web site.
```

```
CREATE OR REPLACE FUNCTION pf
```

```
RETURN pipelined.account_collection
```

```
PIPELINED IS
```

```
-- Declare a collection control variable and collection variable.
```

```
counter NUMBER := 1;
```

```
account PIPELINED.ACCOUNT_COLLECTION := pipelined.account_collection();
```

```
-- Define a cursor.
```

```
CURSOR c IS
```

```
SELECT m.account_number
```

```
, c.last_name || ', ' || c.first_name full_name
```

```
FROM member m JOIN contact c ON m.member_id = c.member_id
```

```
ORDER BY c.last_name, c.first_name, c.middle_initial;
```

```

BEGIN
    FOR i IN c LOOP
        account.EXTEND;
        account(counter).account := i.account_number;
        account(counter).full_name := i.full_name;
        PIPE ROW(account(counter));
        counter := counter + 1;
    END LOOP;
    RETURN;
END pf;
/

```

Những điểm khác biệt là cách bạn tham chiếu loại tập hợp PL/SQL. Bạn phải sử dụng tên package, component selector và tên kiểu dữ liệu. Tuy nhiên, bạn có thể gọi hàm bằng cách tham chiếu chỉ tên hàm, như sau

```
SELECT * FROM TABLE(pf);
```

Các kết quả Pipelined được giới hạn chỉ trong phạm vi SQL

Có một sự ham muốn chuyển giá trị trả về từ một hàm pipelined đến một module PL/SQL khác bởi vì không biết rõ những bảng gộp (aggregate table) này được thiết kế chỉ để sử dụng trong các câu lệnh SQL hay. Bạn nhận được một lỗi PLS-00653 khi bạn cố chuyển một kết quả hàm pipelined đến một chương trình PL/SQL dưới dạng một tham số thật sự. Một lỗi PLS-00653 cho biết rằng các hàm bảng gộp không được cho phép trong phạm vi PL/SQL. Các kết quả bảng pipelined chỉ có thể truy cập trong phạm vi SQL.

Thủ tục sau đây chuyển các cuộc kiểm tra biên dịch bởi vì nó tham chiếu đến một loại tập hợp PL/SQL hợp lệ:

-- This is found in create_pipelined2.sql on the publisher's web site.

```

CREATE OR REPLACE PROCEDURE read_pipe
( pipe_in pipelined.account_collection ) IS
BEGIN
    FOR i IN 1..pipe_in.LAST LOOP
        dbms_output.put(pipe_in(i).account);
        dbms_output.put(pipe_in(i).full_name);
    END LOOP;
END read_pipe;
/

```

Dòng sau đây minh họa cách bạn gọi thủ tục bằng cách chuyển tập hợp kết quả của một lệnh gọi đến hàm pipelined pf:

```
EXECUTE read_pipe(pf);
```

Điều này đưa ra thông báo lỗi sau đây:

```
BEGIN read_pipe(pf); END;
```

```
*
```

```
ERROR at line 1:
```

```
ORA-06550: line 1, column 10:
```

```
PLS-00653: aggregate/table functions are not allowed in PL/SQL scope
```

Lỗi xảy ra bởi vì kiểu dữ liệu thật sự được chuyển đến thủ tục là một tập hợp (aggregate) hoặc table pipelined với những giá trị tương đương nhưng không phải là một kiểu dữ liệu tập hợp PL/SQL. Thật may thay, thông báo lỗi cho bạn sự hồi tiếp tuyệt vời khi bạn biết rằng một bảng gộp pipelined không phải là một loại tập hợp PL/SQL.

Bạn có thể sử dụng các hàm pipelined để xem các view như sau:

```
CREATE OR REPLACE VIEW pipelined_view AS
```

```
SELECT r.account, r.full_name FROM TABLE(pf) r;
```

Các view được tạo bằng các lệnh gọi đến những hàm pipelined đòi hỏi các trigger instead-of để quản lý các hoạt động chèn, cập nhật và xóa. Tối thiểu, bạn tạo trigger instead-of khi bạn muốn cho phép các thao tác DML. Chương 10 đề cập đến cách thực thi một trigger instead-of.

Các hàm pipelined được thiết kế để cho bạn sử dụng các tập hợp gồm những biến vô hướng hoặc cấu trúc record. Các hàm pipelined được trình bày trước đó chuyển đổi tập hợp PL/SQL thành một bảng gộp (aggregate table). Bạn không thể tái sử dụng bảng pipelined trong một phạm vi PL/SQL khác, nhưng bạn có thể sử dụng nó trong các query phạm vi SQL.

Bạn đã học cách sử dụng các hàm pipelined và những ưu điểm và khuyết điểm của chúng. Chúng là những công cụ tuyệt vời khi bạn muốn đưa dữ liệu vào một query hoặc view mà nó đòi hỏi logic thủ tục.

Mệnh đề RESULT_CACHE

Mệnh đề RESULT_CACHE mới trong Oracle 11g Database. Nó chỉ định một hàm được lưu trữ chỉ một lần trong SGA và có sẵn qua các session. Các tham số thật sự của các lệnh gọi trước và các kết quả có sẵn trong cache kết quả. Mệnh đề RESULT_CACHE ra lệnh bộ máy PL/SQL kiểm tra cache kết quả để tìm các lệnh gọi hàm có các tham số thật sự tương hợp. Một lệnh gọi hàm tương hợp cũng lưu trữ kết quả và cache

trả về kết quả và bỏ qua việc chạy lại hàm. Điều này có nghĩa là hàm chỉ chạy khi những tham số mới được gọi đến nó.

Ghi chú

Các hàm session chéo chỉ làm việc với các tham số chế độ IN.

Nguyên mẫu cho mệnh đề RESULT_CACHE có một mệnh đề RELIES_ON tùy chọn. Mệnh đề RELIES_ON quan trọng bởi vì nó bảo đảm bất kỳ thay đổi đối với table nên tảng vô hiệu hóa cache kết quả. Điều này cũng có nghĩa là bất kỳ giao tác DML vốn thay đổi các tập hợp kết quả. Mệnh đề RELIES_ON bảo đảm rằng cache động, tương trưng cho tập hợp kết quả hiện hành. Bạn có thể liệt kê bất kỳ số table độc lập trong mệnh đề RELIES_ON và chúng được liệt kê là các tên được phân tách bằng dấu phẩy.

Ví dụ tiếp theo phụ thuộc vào mã có thể download từ web site của nhà xuất bản. Bạn có thể tìm một mục mô tả về mã trong đoạn giới thiệu. Ví dụ này cũng sử dụng một tập hợp tham chiếu về phía trước nội dung trong chương 6.

Câu lệnh này cho bạn tạo một tập hợp các giá trị VARCHAR2:

```
-- This is found in result_cache.sql on the publisher's web site.
CREATE OR REPLACE TYPE strings AS TABLE OF VARCHAR2(60);
/
```

Hàm này thực thi một cache kết quả session chéo với mệnh đề RELIES_ON:

```
-- This is found in result_cache.sql on the publisher's web site.
CREATE OR REPLACE FUNCTION get_title
( partial_title VARCHAR2 ) RETURN STRINGS
RESULT_CACHE RELIES_ON(item) IS
    -- Declare a collection control variable and collection variable.
    counter NUMBER := 1;
    return_value STRINGS := strings();
-- Define a parameterized cursor.
    CURSOR get_title
    ( partial_title VARCHAR2 ) IS
    SELECT item_title
    FROM item
    WHERE UPPER(item_title) LIKE '%' || UPPER(partial_title) || '%';
BEGIN
```

```

-- Read the data and write it to the collection in a cursor FOR loop.
FOR i IN get_title(partial_title) LOOP
    return_value.EXTEND;
    return_value(counter) := i.item_title;
    counter := counter + 1;
END LOOP;
RETURN return_value;
END get_title;
/

```

Có lẽ chi tiết quan trọng nhất của hàm `get_title` ở trên là bạn nên khởi động bộ đếm (`counter`) tại 1, chứ không phải 0. Hàm sử dụng `counter` làm giá trị index tập hợp và các index tập hợp nên là các số nguyên dương. Một index có một số nguyên không dương đưa ra một lỗi ORA-06532 cho biết subscript nằm ngoài dãy (out of range).

Chú ý

Mệnh đề `RELIES_ON` có thể chấp nhận một tham số hoặc một danh sách các tham số thụt sự.

Bạn có thể test hàm `get_title` với chương trình khối nặc danh sau đây:

```

-- This is found in result_cache.sql on the publisher's web site.
DECLARE
    list STRINGS;
BEGIN
    list := get_title('Harry');
    FOR i IN 1..list.LAST LOOP
        dbms_output.put_line('list(' || i || ') : [' || list(i) || ']');
    END LOOP;
END;
/

```

Sau khi gọi hàm lưu trữ kết quả, bạn chèn, xóa hoặc cập nhật dữ liệu phụ thuộc. Sau đó, bạn sẽ tìm thấy các tập hợp kết quả mới được hiển thị. Sự thay đổi này bảo đảm dữ liệu cũ không bao giờ dẫn sai đường người dùng. Mệnh đề `RELIES_ON` bảo đảm tính toàn vẹn của tập hợp kết quả, nhưng nó đòi hỏi một số hao phí xử lý.

•••••
Thủ thuật

Bạn nên xem xét loại trường mệnh đề `RELIES_ON` để cải thiện hiệu suất giao tác trong các mục thực thi kho chứa dữ liệu.

Các hàm được lưu trữ kết quả cũng có một số hạn chế. Các hàm được lưu trữ kết quả phải đáp ứng tiêu chuẩn sau đây:

- Chúng không thể được định nghĩa trong một module sử dụng các quyền gọi ra hoặc trong một khối độc lập.
- Chúng không thể là một hàm table pipelined.
- Chúng không thể có các ngữ nghĩa chuyển theo tham chiếu như các tham số chế độ IN OUT hoặc OUT.
- Chúng không thể sử dụng các tham số hình thức với một BLOB, CLOB, NCLOB, REF, CURSORS, tập hợp, đối tượng hoặc kiểu dữ liệu record.
- Chúng không thể trả về một biến với một BLOB, CLOB, NCLOB, REF, CURSOR, tập hợp, đối tượng hoặc kiểu dữ liệu record.

Oracle cũng đề nghị các hàm được lưu trữ kết quả không nên chỉnh sửa trạng thái cơ sở dữ liệu, chỉnh sửa trạng thái bên ngoài (bằng cách sử dụng package `DBMS_OUTPUT`), hoặc gửi email (qua package `UTL_SMTP`). Tương tự, hàm không nên phụ thuộc vào các xác lập hoặc ngữ cảnh riêng biệt của session.

Những mục trên đã đề cập những tùy chọn có sẵn để định nghĩa các hàm. Những kỹ năng này được giả định khi thảo luận các hàm chuyển theo giá trị.

Các hàm chuyển theo giá trị

Các hàm chuyển theo giá trị nhận các bản sao của các giá trị khi chúng được gọi. Những hàm này trả về một biến đầu ra đơn sau khi hoàn thành và chúng có thể thực hiện các thao tác bên ngoài. Những thao tác bên ngoài có thể là đọc và ghi vật lý sang hệ điều hành hoặc những câu lệnh SQL trên cơ sở dữ liệu. Xem trở lại bảng 6.1 để tham khảo hàm chuyển theo giá trị.

Như được thảo luận, bạn có thể định nghĩa các hàm chuyển theo giá trị là deterministic hoặc được bật chế độ parallel khi các hàm không thay đổi các biến package hoặc các giá trị cơ sở dữ liệu. Bạn cũng có thể định nghĩa các hàm để trả về các table pipelined vốn mô phỏng các tập hợp SQL hoặc PL/SQL. Kết quả của các hàm pipelined đòi hỏi bạn sử dụng chúng trong phạm vi SQL. Tất cả hàm ngoại trừ các hàm được tạo bằng các kết quả pipelined hỗ trợ các cache kết quả.

Cấu trúc cơ bản của một chương trình chuyển theo giá trị lấy một danh sách các đầu vào còn được gọi là các tham số hình thức. Các hàm

trả về một biến đầu vào đơn. Các biến đầu ra có thể là các giá trị vô hướng, cấu trúc, tập hợp, table pipelined hoặc các loại đối tượng do người dùng định nghĩa. Điều này có nghĩa một biến đơn có thể chứa nhiều thứ khi nó là một kiểu dữ liệu phức hợp.

Cho dù các hàm tương tác với hệ thống file hoặc cơ sở dữ liệu đều không ảnh hưởng đến cách hoạt động của chúng bên trong khối mã PL/SQL. Bạn có thể sử dụng một hàm để gán một kết quả vào một biến, hoặc trả về một biến dưới dạng một biểu thức. Hình 6.1 trước minh họa sử dụng một hàm làm một toán hạng phải trong một thao tác gán.

Bạn có thể sử dụng một hàm vốn trả về một biến làm một biểu thức khi bạn đặt nó bên trong một lệnh gọi đến một hàm cài sẵn PL/SQL khác, như sau

```
EXECUTE dbms_output.put_line(TO_CHAR(pv(10000,5,6),'9,999.90'));
```

Khi SERVEROUTPUT được bật, hàm này trả về kết quả sau đây

```
7,472.58
```

Ví dụ sử dụng hàm pv được mô tả trong mục “Mệnh đề DETERMINISTIC” của chương này và nó sử dụng hàm cài sẵn TO_CHAR. Một lệnh gọi đến hàm pv trở thành một biểu thức cho hàm TO_CHAR và kết quả của hàm TO_CHAR sau đó trở thành một biểu thức và tham số thật sự cho thủ tục PUT_LINE của package DBMS_OUTPUT. Đây là những lệnh gọi điển hình và những công dụng của các hàm chuyển theo giá trị.

Các hàm chuyển theo giá trị PL/SQL được định nghĩa bằng sáu quy tắc sau đây:

- Tất cả tham số hình thức phải được định nghĩa là các biến write-only bằng cách sử dụng chế độ IN.
- Tất cả tham số hình thức là các biến được định phạm vi cục bộ vốn không thể thay đổi trong quá trình thực thi bên trong hàm.

Bất kỳ tham số hình thức có thể sử dụng bất kỳ kiểu dữ liệu SQL hoặc PL/SQL hợp lệ. Chỉ các hàm có những danh sách tham số sử dụng các kiểu dữ liệu SQL mới có thể làm việc trong các câu lệnh SQL.

- Bất kỳ tham số hình thức có thể có một giá trị ban đầu mặc định.
- Biến trả về hình thức có thể sử dụng bất kỳ kiểu dữ liệu SQL hoặc PL/SQL hợp lệ, nhưng các bảng trả về pipelined phải được sử dụng trong các câu lệnh SQL. Bạn không thể truy cập các kết quả bảng pipelined trong một phạm vi PL/SQL khác.
- Bất kỳ việc gán (cast) tham chiếu hệ thống từ một query SQL vào một hàm không thể ghi được và do đó phải được chuyển qua một tham số chế độ IN.

Các Cursor tham chiếu hệ thống

Tất cả tập hợp kết quả cursor là những cấu trúc tĩnh được lưu trữ trong Oracle SGA. Các biến cursor thật sự là những tham chiếu hoặc handle. Handle trở vào một tập hợp kết quả được lưu trữ bên trong từ một query. Bạn tập hợp lại các biến cursor bằng cách truy cập các record thường bằng cách sử dụng một

```
OPEN cursor_name FOR select_statement;
```

Bạn truy cập các cursor bằng cách sử dụng một tham chiếu hoặc handle vốn cho bạn cuộn nội dung của chúng. Bạn cuộn qua chúng bằng cách sử dụng cú pháp biến INTO cursor FETCH. Một khi bạn khai báo một cấu trúc cursor ngầm định hoặc tường minh, bạn có thể gán tham chiếu của nó sang một kiểu dữ liệu cursor SQL. Bạn cũng có thể trả về các biến cursor này dưới dạng các kiểu trả về hàm hoặc dưới dạng các biến tham chiếu IN OUT hoặc OUT trong các chữ ký hàm và thủ tục. Các tập hợp kết quả là những cấu trúc read-only (chỉ đọc).

Dòng mã sau đây trình bày cách trả về một cursor bằng cách sử dụng một hàm:

```
-- This is found in cursor_management.sql on the publisher's web site.
```

```
CREATE OR REPLACE FUNCTION get_full_titles
```

```
RETURN SYS_REFCURSOR IS
```

```
    titles SYS_REFCURSOR;
```

```
BEGIN
```

```
    OPEN titles FOR
```

```
        SELECT item_title, item_subtitle
```

```
        FROM item;
```

```
    RETURN titles;
```

```
END;
```

```
/
```

Hàm này sử dụng SYS_REFCURSOR được định nghĩa sẵn, đây là một cursor tham chiếu hệ thống được định kiểu yếu. Một cursor tham chiếu được định kiểu yếu có thể mang bất kỳ cấu trúc record vào thời gian chạy, trong khi một cursor tham chiếu được định kiểu mạnh được neo vào một đối tượng catalog cơ sở dữ liệu.

Mệnh đề OPEN tạo một tham chiếu trong SGA cho cursor. Sau đó bạn có thể chuyển tham chiếu cho một khối PL/SQL khác dưới dạng một biến cursor, như được minh họa trong khối nặc danh sau đây:

```
-- This is found in cursor_management.sql on the publisher's web site.
```

```
DECLARE
```

```

-- Define a type and declare a variable.
TYPE full_title_record IS RECORD
( item_title item.item_title%TYPE
, item_subtitle item.item_subtitle%TYPE);
full_title FULL_TITLE_RECORD;
-- Declare a system reference cursor variable.
titles SYS_REFCURSOR;

BEGIN
-- Assign the reference cursor function result.
titles := get_full_titles;
-- Print one element of one of the parallel collections.

LOOP
    FETCH titles INTO full_title;
    EXIT WHEN titles%NOTFOUND;
    dbms_output.put_line('Title {' || full_title.item_title || '}');
END LOOP;

END;
/

```

Ghi chú

Không bao giờ có một câu lệnh open trước vòng lặp khi một cursor được chuyển vào một thường trình con bởi vì chúng đã mở. Các biến cursor thật sự là những tham chiếu teo vào một vùng làm việc cursor chuyên dụng trong SGA.

Khối nhận hoặc xử lý cần biết loại record nào được lưu trữ trong cursor. Một số sử dụng yêu cầu này để tranh luận rằng bạn chỉ nên sử dụng các cursor tham chiếu được định kiểu mạnh. Trong các giải pháp chỉ PL/SQL, chúng có một điểm.

Khía cạnh khác của sự tranh cãi có thể là đối với các cursor tham chiếu được định kiểu yếu khi bạn truy vấn chúng qua các chương trình bên ngoài sử dụng các thư viện OCI. Trong những ngôn ngữ bên ngoài này, bạn có thể phát hiện động cấu trúc của các cursor và quản lý chúng một cách riêng biệt qua các thuật toán chung chung.

Tính giá trị tương lai của một khoản tiền gửi ngân hàng minh họa cách ghi một hàm chuyển theo giá trị. Dòng mã sau đây tạo hàm pv tính lãi suất hàng năm được gộp hàng ngày:

```
-- This is found in fv.sql on the publisher's web site.
CREATE OR REPLACE FUNCTION fv
( current_value      NUMBER := 0
, periods           NUMBER := 1
, interest          NUMBER)
RETURN NUMBER DETERMINISTIC IS
BEGIN
-- Compounded Daily Interest.
RETURN current_value * (1 + ((1 + ((interest/100)/365))**365 - 1)*periods);
END fv;
/
```

Hàm định nghĩa ba tham số hình thức. Hai tham số là các tham số tùy chọn bởi vì chúng có những giá trị mặc định. Những giá trị mặc định là số dương hiện tại của tài khoản và 365 ngày của năm (đối với các năm không nhuận). Tham số thứ ba thì bắt buộc bởi vì giá trị không được cung cấp. Như được thảo luận, chế độ IN là chế độ mặc định và bạn không cần phải xác định nó khi định nghĩa các hàm.

Là một thói quen chung, các tham số bắt buộc đứng sau các tham số tùy chọn. Điều này quan trọng khi các tham số thật sự được gọi theo thứ tự vị trí. Oracle 11g hỗ trợ thứ tự vị trí, thứ tự ký hiệu định danh và ký hiệu hỗn hợp.

Sau khi định nghĩa một biến đầu ra, bạn sử dụng câu lệnh CALL để chạy hàm bằng cách sử dụng ký hiệu định danh:

```
VARIABLE future_value NUMBER
CALL fv(current_value => 10000, periods => 5, interest => 4)
INTO :future_value
/
```

Sau đó bạn có thể chọn giá trị tương lai \$10,000 sau năm năm với lãi suất hàng năm 4% được gộp hàng ngày bằng cách sử dụng:

```
SELECT :future_value FROM dual;
```

Hoặc, bạn có thể định dạng với SQL*Plus và gọi hàm trong SQL bằng câu lệnh sau đây:

```
COLUMN future_value FORMAT 99,999.90

SELECT fv(current_value => 10000, periods => 5, interest => 4) FROM dual;
```

Cả câu lệnh CALL và query SQL trả về một kết quả \$12,040.42. Việc gộp lãi sẽ mang lại hơn một lãi suất hàng năm \$40.42. Có thể có dư một

hoặc hai xu phụ thuộc vào nơi năm nhuận rơi vào trong năm năm, nhưng hàm không thay đổi sắc thái đó trong phép tính.

Các hàm chuyển theo giá trị không cho phép bất kỳ nỗ lực gán lại một giá trị vào một tham số hình thức trong thời gian chạy. Bạn đưa ra một lỗi PLS-00363, lỗi này cho bạn biết biểu thức (tham số hình thức) không thể được sử dụng làm đích gán.

Các hàm cũng cho bạn xử lý các câu lệnh DML bên trong chúng. Có một số người cảm thấy bạn không nên sử dụng các hàm để thực thi các câu lệnh DML bởi vì trước đây các thủ tục đã được sử dụng. Khuyết điểm duy nhất của việc nhúng một câu lệnh DML bên trong một hàm là bạn không thể gọi hàm đó bên trong một query. Nếu bạn gọi một hàm thực thi một câu lệnh DML từ một query, bạn đưa ra một lỗi ORA-14551. Thông báo lỗi cho biết bạn không thể có một thao tác DML bên trong một query.

Hàm sau đây chèn một hàm bằng cách gọi hàm `add_user` độc lập:

-- This is found in `create_add_user.sql` on the publisher's web site.

```
CREATE OR REPLACE FUNCTION add_user
( system_user_id      NUMBER
, system_user_name   VARCHAR2
, system_group_id    NUMBER
, system_user_type   NUMBER
, last_name          VARCHAR2
, first_name         VARCHAR2
, middle_initial     VARCHAR2
, created_by         NUMBER
, creation_date      DATE
, last_updated_by    NUMBER
, last_update_date   DATE ) RETURN BOOLEAN IS
    -- Set function to perform in its own transaction scope.
    PRAGMA AUTONOMOUS_TRANSACTION;
    -- Set default return value.
    retval BOOLEAN := FALSE;
BEGIN
    INSERT INTO system_user
    VALUES
    ( system_user_id, system_user_name, system_group_id, system_user_type
    , last_name, first_name, middle_initial
```

```

, created_by, creation_date, last_updated_by, last_update_date );
-- Save change inside its own transaction scope.
    COMMIT;
-- Reset return value.
    retval := TRUE;
    RETURN retval;
END;
/

```

Các đơn vị chương trình độc lập thực thi những hoạt động của chúng trong một phạm vi giao tác riêng biệt, nghĩa là hành vi của chúng tách biệt khỏi phạm vi giao tác gọi. Chương trình khối nặc danh này minh họa cách bạn sử dụng một hàm làm biểu thức trong một câu lệnh IF khi nó thi hành một thao tác DML trong một hàm độc lập:

-- This is found in create_add_user.sql on the publisher's web site.

```

BEGIN
    IF add_user( 6,'Application DBA', 1, 1
                , 'Brown', 'Jerry', ''
                , 1, SYSDATE, 1, SYSDATE) THEN
        dbms_output.put_line('Record Inserted');
    ROLLBACK;
    ELSE
        dbms_output.put_line('No Record Inserted');
    END IF;
END;
/

```

Việc rollback (phục hồi trở lại) không undo việc chèn bởi vì nó chỉ áp dụng vào phạm vi giao tác hiện hành. Hàm add_user là một giao tác độc lập và do đó ghi các thay đổi trong một phạm vi giao tác độc lập. Khi hàm trả về một giá trị Boolean true, giá trị đã được ghi và được làm cho trở thành thường trực. Sau đó bạn truy vấn hàng, và bạn sẽ thấy hàng vẫn có ở đó ngay cả khi phạm vi gọi thất bại hoặc đã được roll back.

••••• Thủ thuật

Bạn không thể chuyển một cursor tham chiếu hệ thống dưới dạng một tham số thật sự chế độ IN và sau đó mở chúng, bởi vì chúng đã mở.

Các hàm đệ quy

Các hàm đệ quy (recursive function) là một công cụ hữu dụng để giải quyết một số vấn đề phức tạp như phân tích cú pháp nâng cao. Một hàm đệ quy gọi một hoặc nhiều bản sao của chính nó để giải quyết một vấn đề bằng cách đệ quy trên một kết quả. Các hàm đệ quy nhìn lùi về thời gian, trong khi các hàm không đệ quy nhìn hướng về thời gian. Các hàm đệ quy là một dạng chuyên dụng của các hàm chuyển theo giá trị.

Các chương trình không đệ quy lấy một số tham số và bắt đầu xử lý chương trình trong vòng lặp, cho đến khi chúng đạt được một kết quả. Điều này có nghĩa chúng bắt đầu với một thứ gì đó và làm việc với nó cho đến khi chúng tìm thấy một kết quả bằng cách áp dụng một tập hợp quy tắc hoặc sự lượng giá. Điều này có nghĩa các chương trình không đệ quy giải quyết các vấn đề di chuyển về phía trước trong thời gian.

Các hàm đệ quy có một trường hợp cơ sở và một trường hợp đệ quy. Trường hợp cơ sở là kết quả dự đoán trước. Trường hợp đệ quy áp dụng một công thức bao gồm một hoặc nhiều lệnh gọi trở lại cùng một hàm. Lệnh gọi đệ quy được gọi là phép đệ quy tuyến tính hoặc đường thẳng. Phép đệ quy tuyến tính nhanh hơn nhiều so với phép đệ quy không tuyến tính và các lệnh gọi càng đệ quy thì chi phí xử lý càng cao. Các hàm đệ quy sử dụng trường hợp đệ quy trừ phi trường hợp cơ sở không được đáp ứng. Một kết quả được tìm thấy khi một lệnh gọi hàm đệ quy trở về giá trị trường hợp cơ sở. Điều này có nghĩa các đơn vị chương trình đệ quy giải quyết các vấn đề di chuyển lùi trở lại thời gian hoặc một phép đệ quy này rồi đến một phép đệ quy khác.

Giải quyết các kết quả giai thừa là một vấn đề đặc trưng cho phép đệ quy tuyến tính. Hàm sau đây trả về giá trị giai thừa cho bất kỳ số:

-- This is found in recursion.sql on the publisher's web site.

```
CREATE OR REPLACE FUNCTION factorial
( n BINARY_DOUBLE ) RETURN BINARY_DOUBLE IS
BEGIN
    IF n <= 1 THEN
        RETURN 1;
    ELSE
        RETURN n * factorial(n - 1);
    END IF;
END factorial;
/
```

Trường hợp cơ sở được đáp ứng khi câu lệnh IF phân giải là true. Trường hợp đệ quy chỉ gọi đến cùng một hàm. Trường hợp đệ quy có thể

gọi nhiều lần cho đến khi nó cũng trả về giá trị trường hợp cơ sở là 1. Sau đó, nó làm việc trở lại lên cây các lệnh gọi đệ quy cho đến khi một lời giải đáp được tìm thấy bởi lệnh gọi đầu tiên.

Các số fibonacci phức tạp hơn bởi vì các phép đệ quy đòi hỏi hai lệnh gọi cho mỗi phép đệ quy. Hàm sau đây minh họa phép đệ quy không tuyến tính:

-- This is found in recursion.sql on the publisher's web site.

```
CREATE OR REPLACE FUNCTION Fibonacci
( n BINARY_DOUBLE ) RETURN BINARY_DOUBLE IS
BEGIN
    IF n <= 2 THEN
        RETURN 1;
    ELSE
        RETURN fibonacci(n - 2) + fibonacci(n - 1);
    END IF;
END fibonacci;
/
```

Toán tử cộng có một thứ tự ưu tiên thấp hơn một lệnh gọi hàm. Do đó, lệnh gọi đệ quy nằm bên trái được xử lý trước tiên cho đến khi nó trả về một biểu thức. Sau đó, lệnh gọi đệ quy nằm bên phải được phân giải thành một biểu thức. Phép cộng xảy ra sau khi cả hai lệnh gọi đệ quy trả về các biểu thức.

Các hàm chuyển theo tham chiếu

Các hàm chuyển theo tham chiếu nhận các bản sao của các giá trị khi chúng được gọi trừ phi bạn ghi đè hành vi mặc định bằng cách sử dụng gợi ý NOCOPY. Gợi ý NOCOPY chỉ làm việc với các hàm nhất định đáp ứng tiêu chuẩn giới hạn. Những hàm này trả về một biến đầu ra đơn sau khi hoàn thành. Chúng có thể thực thi các thao tác bên ngoài như các câu lệnh SQL. Vào thời gian chạy, chúng có thể trả về những giá trị mới cho những tham số thật sự trở về các đơn vị chương trình gọi. Nếu chúng sử dụng gợi ý NOCOPY và chuyển các tham số, hàm có thể thay đổi bất kỳ giá trị được trở vào bởi tham chiếu. Các thao tác bên ngoài có thể là các hoạt động đọc và ghi vật lý sang hệ điều hành hoặc những câu lệnh SQL trên cơ sở dữ liệu. Xem lại bảng 6.1 để tham khảo hàm chuyển theo tham chiếu.

Bạn sử dụng các hàm chuyển theo tham chiếu khi bạn muốn thực hiện một thao tác, trả về một giá trị từ hàm và sau đó thay đổi một hoặc nhiều tham số thật sự. Những hàm này chỉ có thể hành động bên trong phạm vi của một chẳng hạn như hoặc môi trường khác. Môi trường

SQL*Plus cho bạn định nghĩa các biến cấp session (còn được gọi là các biến liên kết) mà bạn có thể sử dụng khi bạn gọi những loại hàm này. Bạn không thể chuyển các trực kiện (như các ngày tháng, số hoặc chuỗi) hoặc những biểu thức (như các giá trị trả về hàm) vào một tham số được định nghĩa là chế độ OUT hoặc IN OUT.

Các hàm chuyển theo tham chiếu PL/SQL được định nghĩa bởi sáu quy tắc sau đây:

- Tối thiểu một tham số hình thức phải được định nghĩa là một biến read-only hoặc read-write bằng cách sử dụng chế độ OUT hoặc IN OUT.
- Tất cả tham số hình thức là các biến được định phạm vi cục bộ mà bạn có thể thay đổi trong các thao tác bên trong hàm.
- Bất kỳ tham số hình thức có thể sử dụng bất kỳ kiểu dữ liệu SQL hoặc PL/SQL hợp lệ. Chỉ các hàm có các danh sách tham số sử dụng các kiểu dữ liệu SQL mới có thể làm việc trong những câu lệnh SQL.
- Bất kỳ tham số hình thức chế độ IN có thể có một giá trị ban đầu mặc định.
- Biến trả về hình thức có thể sử dụng bất kỳ kiểu dữ liệu SQL hoặc PL/SQL hợp lệ, nhưng các bảng trả về pipelined phải được sử dụng trong các câu lệnh SQL. Bạn không thể truy cập các kết quả bảng pipelined trong một phạm vi PL/SQL khác.
- Bất kỳ việc gán (cast) cursor tham chiếu hệ thống từ một query SQL vào một hàm không thể ghi được và do đó phải được chuyển qua một tham số chế độ IN.

Hàm chuyển theo tham chiếu sau đây minh họa việc trả về một giá trị trong khi thay đổi biến đầu vào:

```
-- This is found in create_counting1.sql on the publisher's web site.
```

```
CREATE OR REPLACE FUNCTION counting
```

```
( number_in IN OUT NUMBER ) RETURN VARCHAR2 IS
```

```
-- Declare a collection control variable and collection variable.
```

```
TYPE numbers IS TABLE OF VARCHAR2(5);
```

```
ordinal NUMBERS := numbers('One','Two','Three','Four','Five');
```

```
-- Define default return value.
```

```
retval VARCHAR2(9) := 'Not Found';
```

```
BEGIN
```

```
-- Replace a null value to ensure increment.
```

```
IF number_in IS NULL THEN
```

```
number_in := 1;
```

```

END IF;
-- Increment actual parameter when within range.
IF number_in < 4 THEN
    retval := ordinal(number_in);
    number_in := number_in + 1;
ELSE
    retval := ordinal(number_in);
END IF;
RETURN retval;
END;
/

```

Hàm bảo đảm giá trị index number_in không rỗng và không vượt quá 4. Giá trị index đọc một số thứ tự từ tập hợp bảng xếp lồng. Bạn có thể test hàm bằng khối nặc danh sau đây:

```

-- This is found in create_counting1.sql on the publisher's web site.
DECLARE
    counter NUMBER := 1;
BEGIN
    FOR i IN 1..5 LOOP
        dbms_output.put('Counter [' || counter || ']');
        dbms_output.put_line([' || counting(counter) || ']);
    END LOOP;
END;
/

```

Biến counter luôn được in trước khi gọi hàm. Điều này có nghĩa bạn thấy giá trị ban đầu và chuỗi số thứ tự tương hợp. Kết quả là

```

Counter [1][One]
Counter [2][Two]
Counter [3][Three]
Counter [4][Four]
Counter [4][Four]

```

Như bạn có thể thấy trong kết quả, tham số thật sự chế độ IN OUT chỉ được tăng cho đến khi giá trị là 4. Lệnh gọi sau cùng trong khối nặc danh tái sử dụng giá trị index không thay đổi trước bởi vì hàm chỉ tăng các giá trị nhỏ hơn bốn.

Một tham số hình thức (chế độ OUT) chỉ đọc không thể làm việc trong loại lệnh gọi này bởi vì giá trị mới không bao giờ được đọc. Câu lệnh IF ban đầu xác lập `number_in` sang 1 mỗi lần bạn gọi chương trình với một tham số thật sự rỗng. Các tham số chế độ OUT luôn luôn là các giá trị rỗng lúc nhập vào.

Hàm đếm được tái tạo với một tham số chế độ OUT như sau:

-- This is found in create_counting1.sql on the publisher's web site.

```
CREATE OR REPLACE FUNCTION counting
( number_out OUT NUMBER ) RETURN VARCHAR2 IS
    TYPE numbers IS TABLE OF VARCHAR2(5);
    ordinal NUMBERS := numbers('One','Two','Three','Four','Five');
    retval VARCHAR2(9) := 'Not Found';
BEGIN
    -- Replace a null value to ensure increment.
    IF number_out IS NULL THEN
        number_out := 1;
    END IF;
    -- Increment actual parameter when within range.
    IF number_out < 4 THEN
        retval := ordinal(number_out);
        number_out := number_out + 1;
    ELSE
        retval := ordinal(number_out); -- Never run because number_out is always null.
    END IF;
    RETURN retval;
END;
/
```

Hàm gọi mới sử dụng một tham số chế độ OUT. Tham số được đổi tên thích hợp thành `number_out`. Vào thời gian gọi, giá trị `number_out` luôn là một giá trị rỗng. Điều này có nghĩa chúng luôn được xác lập lại sang 1, được tìm thấy nhỏ hơn 4 và được xác lập lại sang 2.

Khối nặc danh quen thuộc này cho bạn test hàm mới:

```
DECLARE
    counter NUMBER := 1;
BEGIN
    FOR i IN 1..5 LOOP
```

```

dbms_output.put('Counter [' || counter || ']');
dbms_output.put_line('[' || counting(counter) || ']');
END LOOP;

END;

/

```

Lúc đầu counter là 1 và luôn được trả về là 2, nghĩa là bạn có được kết quả sau đây:

```

Counter [1][One]
Counter [2][One]
Counter [2][One]
Counter [2][One]
Counter [2][One]

```

Mục này đã đề cập cách bạn định nghĩa và sử dụng một hàm chuyển theo tham chiếu. Bạn nên nhận ra rằng có hai loại tham số chuyển theo tham chiếu. Một loại có một giá trị khi đi vào và thoát: các biến chế độ IN OUT. Loại kia luôn có một giá trị rỗng lúc đi vào và luôn nên có một giá trị lúc thoát: các tham số chế độ OUT.

Các thủ tục

Các thủ tục (procedure) không thể là những toán hạng phải hoặc được gọi từ những câu lệnh SQL. Chúng hỗ trợ việc sử dụng các tham số hình thức chế độ IN, OUT và IN OUT.

Như các hàm, các thủ tục cũng có thể chứa các khối định danh xếp lồng. Các khối định danh xếp lồng là các hàm cục bộ mà bạn định nghĩa trong khối khai báo. Tương tự bạn xếp lồng các khối nặc danh trong khối thực thi hoặc các thủ tục.

Dòng mã sau đây minh họa một nguyên mẫu thủ khối định danh:

```

PROCEDURE procedure_name
([ parameter1 [IN][OUT] [NOCOPY] sql_datatype | plsql_datatype
, parameter2 [IN][OUT] [NOCOPY] sql_datatype | plsql_datatype
, parameter(n+1) [IN][OUT] [NOCOPY] sql_datatype | plsql_datatype ])
[ AUTHID DEFINER | CURRENT_USER ] IS
declaration_statements
BEGIN
    execution_statements
[EXCEPTION]
    exception_handling_statements

```

```
END [procedure_name];
```

```
/
```

Bạn có thể định nghĩa các thủ tục có hoặc không có các tham số hình thức. Các tham số hình thức trong những thủ tục có thể là các biến chuyển theo giá trị (pass-by-value) hoặc chuyển theo tham chiếu (pass-by-reference) trong các thủ tục lưu trữ. Các biến chuyển theo tham chiếu có cả hai chế độ IN và OUT. Như khi làm việc với các hàm, bạn tạo nó dưới dạng một tham chiếu chuyển theo giá trị khi bạn không xác định chế độ tham số bởi vì nó sử dụng chế độ IN mặc định.

Biên dịch (tạo hoặc thay thế) thủ tục sẽ gán ngầm định cụm từ chế độ IN khi không có gì được cung cấp. Như các hàm, các tham số hình thức trong những thủ tục cũng hỗ trợ các giá trị mặc định tùy chọn cho các tham số chế độ IN.

Mệnh đề AUTHID xác lập mô hình quyền thực thi. Mặc định là các quyền định nghĩa. Các quyền định nghĩa nghĩa là bất kỳ người nào có các đặc quyền thực thi trên thủ tục hành động như thể họ là người sở hữu cùng một schema. CURRENT_USER ghi đè quyền mặc định và xác lập quyền thực thi sang các quyền gọi ra (invoker rights). Các quyền gọi ra nghĩa là bạn gọi các thủ tục để hành động lên dữ liệu cục bộ, và nó đòi hỏi bạn sao chép các đối tượng dữ liệu trong bất kỳ schema tham gia. Chương 9 so sánh rộng hơn các quyền định nghĩa và quyền gọi ra.

Như trong các hàm, khối khai báo nằm giữa các cụm từ IS và BEGIN, trong khi những khối khác phản ánh cấu trúc của các chương trình khối nặc danh. Các thủ tục đòi hỏi một môi trường thực thi nghĩa là bạn phải gọi chúng từ SQL*Plus hoặc một đơn vị chương trình khác. Đơn vị chẳng hạn như gọi có thể là một khối PL/SQL khác hoặc một chương trình bên ngoài sử dụng OCI hoặc JDBC.

Các thủ tục được sử dụng thường xuyên nhất để thực thi các câu lệnh DML và việc quản lý giao tác. Bạn định nghĩa các thủ tục để hành động trong phạm vi giao tác hiện hành hoặc một phạm vi giao tác độc lập. Như với các hàm, bạn sử dụng PRAGMA AUTONOMOUS_TRANSACTION để xác lập một thủ tục sao cho nó chạy dưới dạng một giao tác độc lập.

Các thủ tục chuyển theo giá trị

Các thủ tục chuyển theo giá trị nhận những bản sao của các giá trị khi chúng được gọi. Những thủ tục này không trả về một biến đầu ra như một hàm. Chúng chỉ thực thi các thao tác bên ngoài, như các câu lệnh SQL cho cơ sở dữ liệu hoặc các thao tác đọc hoặc ghi file bên ngoài. Xem lại bảng 6.1 để tham khảo thủ tục chuyển theo giá trị.

Như được thảo luận, bạn có thể định nghĩa các thủ tục chuyển theo giá trị để chạy độc lập trong một phạm vi giao tác riêng biệt hoặc bạn có thể chấp nhận mặc định và chạy chúng trong phạm vi giao tác hiện hành. Các thủ tục chuyển theo giá trị thường xuyên chạy trong phạm vi giao tác hiện hành. Chúng tổ chức các câu lệnh DML cơ sở dữ liệu, như các câu lệnh insert cho nhiều table.

Các thủ tục chuyển theo giá trị PL/SQL được định nghĩa bằng năm quy tắc sau đây:

- Tất cả tham số hình thức phải được định nghĩa là các biến write-only bằng cách sử dụng chế độ IN.
- Tất cả tham số hình thức là các biến được định phạm vi cục bộ vốn không thể được thay đổi trong quá trình thực thi bên trong thủ tục.
- Bất kỳ tham số hình thức có thể sử dụng bất kỳ kiểu dữ liệu SQL hoặc PL/SQL hợp lệ.
- Bất kỳ tham số hình thức có thể có một giá trị ban đầu mặc định.
- Bất kỳ việc cast (gán) cursor tham chiếu hệ thống từ một query SQL vào một hàm không thể thi được và do đó phải được chuyển qua một tham số chế độ IN. Điều này bao gồm những tham số được chuyển dưới dạng các biến cursor tường minh và các tham số được cast bằng cách sử dụng hàm CURSOR. Như được đề cập trong mục trước "Các Cursor tham chiếu hệ thống" của chương này, các biến cursor thật sự là bằng tham chiếu hoặc handle. Những handle trở vào các tập hợp kết quả được lưu trữ nội tại vốn là các cấu trúc chỉ đọc.

Thủ tục add_contact trình bày một thủ tục chèn những giá trị vào một hoặc ba table. Thủ tục sử dụng các tham số gọi để quyết định các bảng chèn đích. Tất cả bảng bên trong cửa hàng video sử dụng các khóa chính đại diện cùng với những bản sao của các giá trị khóa chính làm các khóa ngoại. add_contact không chấp nhận các khóa chính đại diện, nhưng nó sử dụng chúng bên trong thủ tục. Đây là một trong những ưu điểm của việc đặt các hoạt động chèn liên quan vào một thủ tục đơn.

Thủ tục add_contact hướng dẫn bạn cách sử dụng một tham chiếu chuyển theo giá trị để quản lý nhiều câu lệnh DML qua một phạm vi giao tác:

-- This is found in create_add_contact1.sql on the publisher's web site.

```
CREATE OR REPLACE procedure add_contact
( member_id          NUMBER
, contact_type       NUMBER
, last_name           VARCHAR2
, first_name         VARCHAR2
, middle_initial     VARCHAR2 := NULL
```

```

, address_type          NUMBER := NULL
, street_address       VARCHAR2 := NULL
, city                 VARCHAR2 := NULL
, state_province       VARCHAR2 := NULL
, postal_code          VARCHAR2 := NULL
, created_by           NUMBER
, creation_date        DATE := SYSDATE
, last_updated_by     NUMBER
, last_update_date    DATE := SYSDATE) IS
-- Declare surrogate key variables.
contact_id            NUMBER;
address_id            NUMBER;
street_address_id    NUMBER;
-- Define autonomous function to secure any surrogate key values.
FUNCTION get_sequence_value (sequence_name VARCHAR2) RETURN NUM-
BER IS
    PRAGMA AUTONOMOUS_TRANSACTION;
    id_value NUMBER;
    statement VARCHAR2(2000);
BEGIN
    -- Build and run dynamic SQL in a PL/SQL block.
    statement := 'BEGIN' || CHR(10)
        || ' SELECT ' || sequence_name || '.nextval' || CHR(10)
        || ' INTO :id_value' || CHR(10)
        || ' FROM dual;' || CHR(10)
        || 'END;';
    EXECUTE IMMEDIATE statement USING OUT id_value;
    RETURN id_value;
END get_sequence_value;
BEGIN
    -- Set savepoint to guarantee all or nothing happens.
    SAVEPOINT add_contact;
    -- Assign next value from sequence and insert record.
    contact_id := get_sequence_value('CONTACT_S1');

```

```
INSERT INTO contact VALUES
( contact_id
, member_id
, contact_type
, last_name
, first_name
, middle_initial
, created_by
, creation_date
, last_updated_by
, last_update_date);
-- Check before inserting data in ADDRESS table.
IF address_type IS NOT NULL          AND
   city IS NOT NULL                  AND
   state_province IS NOT NULL        AND
   postal_code IS NOT NULL           THEN
-- Assign next value from sequence and insert record.
address_id := get_sequence_value('ADDRESS_S1');
INSERT INTO address VALUES
( address_id
, contact_id
, address_type
, city
, state_province
, postal_code
, created_by
, creation_date
, last_updated_by
, last_update_date);
-- Check before inserting data in STREET_ADDRESS table.
IF street_address IS NOT NULL THEN
-- Assign next value from sequence and insert record.
street_address_id := get_sequence_value('STREET_ADDRESS_S1');
INSERT INTO street_address VALUES
( street_address_id
```



```

        , address_id
        , street_address
        , created_by
        , creation_date
        , last_updated_by
        , last_update_date);
    END IF;
END IF;
EXCEPTION
    WHEN others THEN
        ROLLBACK TO add_contact;
        RAISE_APPLICATION_ERROR(-20001,SQLERRM);
END add_contact;
/

```

Bạn gửi dữ liệu đến thủ tục `add_contact` và nó chèn dữ liệu vào một hoặc ba bảng. Tất cả tham số hình thức sử dụng các tham số chế độ IN. Điều này có nghĩa bạn không thể gán bất cứ thứ gì vào các biến này bên trong thủ tục. Đây là một lý do tại sao các biến thủ tục cục bộ quản lý khóa chính (primary key) và khóa ngoại (foreign key).

Thủ tục quản lý tất cả khóa chính và khóa ngoại bảo đảm chúng có sẵn khi được yêu cầu trong quá trình thực thi thủ tục. Nó xác lập một SAVEPOINT ở đầu và roll back bất kỳ thành mục giao tác nếu có bất kỳ lỗi được đưa ra. Nó đưa ra một lỗi do người dùng định nghĩa khi một ngoại lệ xuất hiện. Hàm độc lập không tác động đến tính toàn vẹn giao tác bởi vì việc truy vấn một trình tự trong cùng một phạm vi giao tác hoặc một phạm vi giao tác khác sẽ tăng lượng một trình tự. Các trình tự không bao giờ được xác lập lại bởi một câu lệnh ROLLBACK.

Hàm `get_sequence_value` cục bộ sử dụng Native Dynamic SQL (NDS) sao cho một hàm có thể truy cập các trình tự hỗ trợ các khóa chính. Thủ tục nhận các khóa chính mới trước khi thử bất kỳ câu lệnh INSERT.

Bạn có thể test thủ tục bằng cách gọi nó, như được minh họa trong chương trình khối nặc danh sau đây:

```
-- This is found in create_add_contact1.sql on the publisher's web site.
```

```

DECLARE
    -- Declare surrogate key variables.
    member_id NUMBER;
    -- Declare local function to get type.
    FUNCTION get_type

```

```

( table_name VARCHAR2
, column_name VARCHAR2
, type_name VARCHAR2) RETURN NUMBER IS
retval NUMBER;
BEGIN
    SELECT    common_lookup_id
    INTO      retval
    FROM      common_lookup
    WHERE     common_lookup_table = table_name
    AND       common_lookup_column = column_name
    AND       common_lookup_type = type_name;
    RETURN    retval;
END get_type;
-- Define autonomous function to secure surrogate key values.
FUNCTION get_member_id RETURN NUMBER IS
    PRAGMA AUTONOMOUS_TRANSACTION;
    id_value NUMBER;
BEGIN
    SELECT member_s1.nextval INTO id_value FROM dual;
    RETURN id_value;
END;
BEGIN
-- Set savepoint to guarantee all or nothing happens.
SAVEPOINT add_member;
-- Declare surrogate key variables.
member_id := get_member_id;
INSERT INTO member VALUES
( member_id
,(SELECT    common_lookup_id
FROM      common_lookup
WHERE     common_lookup_table = 'MEMBER'
AND       common_lookup_column = 'MEMBER_TYPE'
AND       common_lookup_type = 'GROUP')
, '4563-98-71'
, '5555-6363-1212-4343'

```

```

,(SELECT      common_lookup_id
FROM          common_lookup
WHERE common_lookup_table = 'MEMBER'
AND common_lookup_column = 'CREDIT_CARD_TYPE'
AND common_lookup_type = 'VISA_CARD')
, 3
, SYSDATE
, 3
, SYSDATE);
-- Call procedure to insert records in related tables.
add_contact( member_id => member_id
, contact_type => get_type('CONTACT','CONTACT_TYPE','CUSTOMER')
, last_name => 'Rodriguez'
, first_name => 'Alex'
, address_type => get_type('ADDRESS','ADDRESS_TYPE','HOME')
, street_address => 'East 161st Street'
, city => 'Bronx'
, state_province => 'NY'
, postal_code => '10451'
, created_by => 3
, last_updated_by => 3);
EXCEPTION
  WHEN others THEN
    ROLLBACK TO add_member;
    RAISE_APPLICATION_ERROR(-20002,SQLERRM);
END;
/

```

Khởi nặc danh chèn một hàng vào bảng member và sau đó gọi thủ tục để chèn dữ liệu vào các bảng contact, address, street_address. Lệnh gọi thủ tục sử dụng ký hiệu định danh gọi thủ tục add_contact.

Các thủ tục chuyển theo giá trị cho bạn thực hiện các tác vụ trong cơ sở dữ liệu hoặc các nguồn tài nguyên bên ngoài. Chúng cũng cho bạn quản lý các khóa chính và khóa ngoại trong phạm vi chương trình.

Các thủ tục chuyển theo tham chiếu

Các thủ tục chuyển theo tham chiếu nhận những tham chiếu dẫn sang các biến khi chúng được gọi. Các thủ tục không trả về các biến đầu ra. Loại thủ tục này có thể thay đổi những giá trị của các tham số thật sự. Chúng trả những tham chiếu tham số thật sự của chúng sau khi hoàn thành trở về chương trình gọi. Chúng cũng có thể thực thi các thao tác bên ngoài, như các câu lệnh SQL cho cơ sở dữ liệu. Xem lại bảng 6.1 để tham khảo một thủ tục chuyển theo tham chiếu.

Như được thảo luận, bạn có thể định nghĩa các thủ tục chuyển theo tham chiếu để chạy độc lập. Sau đó, chúng thực thi trong một phạm vi giao tác riêng biệt. Bạn cũng có thể chấp nhận mặc định và chạy chúng trong phạm vi giao tác hiện hành. Chúng tổ chức các câu lệnh DML cơ sở dữ liệu để di chuyển dữ liệu giữa chương trình và cơ sở dữ liệu hoặc chúng gửi dữ liệu đến các đơn vị chương trình bên ngoài.

Các thủ tục chuyển theo tham chiếu PL/SQL được định nghĩa bằng năm quy tắc sau đây:

- Tối thiểu một tham số hình thức phải được định nghĩa là một biến read-only hoặc read-write bằng cách lần lượt sử dụng chế độ OUT hoặc IN OUT.
- Tất cả tham số hình thức là các biến được định phạm vi cục bộ mà bạn có thể thay đổi trong các hoạt động bên trong thủ tục.
- Bất kỳ tham số hình thức có thể sử dụng bất kỳ kiểu dữ liệu SQL hoặc PL/SQL hợp lệ.
- Bất kỳ tham số hình thức chế độ IN có thể có một giá trị ban đầu mặc định.
- Bất kỳ việc cast (gán) cursor tham chiếu hệ thống từ một query SQL vào một thủ tục không thể ghi được và do đó phải được chuyển qua một tham số chế độ IN.

Các chương trình chuyển theo giá trị cho bạn đặt các dãy gồm nhiều câu lệnh DML vào một giao tác và phạm vi chương trình. Bạn có thể chia sẻ các giá trị như các khóa chính và khóa ngoại bên trong hộp đen (black box) khi bạn sử dụng chúng. Như được ghi chú trong mục trước, thủ tục `add_contact` trình bày cách bạn thực thi một tập hợp các câu lệnh `INSERT` có điều kiện.

Đôi khi bạn muốn xây dựng các đơn vị chương trình nhỏ hơn có thể tái sử dụng. Ví dụ, mỗi câu lệnh `insert` có thể được đặt vào thủ tục lưu trữ riêng của nó. Bạn hoàn thành điều đó bằng cách thực thi các thủ tục chuyển theo tham chiếu. Những thủ tục mới này mở rộng các danh sách tham số bằng cách sử dụng các tham số khóa chính và khóa ngoại. Sự thay đổi danh sách tham số làm cho những thủ tục có thể trao đổi các giá trị giữa những chương trình.

Ví dụ tái thực thi giải pháp mục chuyển theo giá trị (pass-by-value) trước dưới dạng một tập hợp thủ tục chuyển theo tham chiếu. Bước đầu tiên loại bỏ hàm `get_sequence_value` cục bộ và tạo nó dưới dạng một hàm độc lập trong cơ sở dữ liệu, như được minh họa:

```
-- This is found in create_add_contact2.sql on the publisher's web site.
CREATE OR REPLACE FUNCTION get_sequence_value
(sequence_name VARCHAR2) RETURN NUMBER IS
    PRAGMA AUTONOMOUS_TRANSACTION;
    id_value NUMBER;
    statement VARCHAR2(2000);
BEGIN
    -- Build dynamic SQL statement as anonymous block PL/SQL unit.
    statement := 'BEGIN' || CHR(10)
                || ' SELECT ' || sequence_name || '.nextval' || CHR(10)
                || ' INTO :id_value' || CHR(10)
                || ' FROM dual;' || CHR(10)
                || 'END;';
    -- Execute dynamic SQL statement.
    EXECUTE IMMEDIATE statement USING OUT id_value;
    RETURN id_value;
END get_sequence_value;
/
```

Phiên bản của hàm này sử dụng SQL động riêng (NDS) để cơ sở dữ liệu và chạy câu lệnh `SELECT` vốn nhận được giá trị dãy. Chương 11 đề cập đến NDS.

Sau khi xây dựng hàm độc lập, bạn cần xây dựng một thủ tục để thêm một hàng vào bảng `contact`. Thủ tục `add_contact` mới chỉ thêm một hàng vào bảng `contact`. Nó cũng có một danh sách tham số hình thức khác. Khóa chính cho bảng được trả về dưới dạng một biến chế độ `OUT` (write-only) vốn cho bạn tái sử dụng khóa chính làm một khóa ngoại, đây là những gì mà bạn sẽ làm trong một thủ tục tiếp theo. Bạn cũng nên chú ý rằng khóa ngoại `member_id` được chuyển dưới dạng một giá trị.

Inlining các lệnh gọi thường trình con

Inlining là một hành vi của trình biên dịch, sao chép một thường trình con bên ngoài vào một chương trình khác. Điều này được thực hiện để tránh hao phí là thường xuyên gọi một thường trình con bên ngoài. Trong khi quyết định cho trình biên dịch luôn là một tùy chọn, bạn có thể chỉ định khi

nào bạn muốn đề nghị một lệnh gọi bên ngoài được sao chép inline.

Bạn chỉ định một lệnh gọi thường trình con để inlining bằng cách sử dụng nguyên mẫu sau đây:

```
PRAGMA INLINE(subroutine_name, 'YES'|'NO')
```

Cuối cùng trình biên dịch quyết định có inline thường trình con hay không bởi vì các chỉ lệnh tiền biên dịch chỉ là những gợi ý. Có những yếu tố khác làm cho việc inlining một số thường trình con không thể mong muốn được. Pragma này ảnh hưởng đến bất kỳ lệnh gọi đến hàm hoặc thủ tục khi nó đứng trước lệnh gọi. Nó cũng tác động đến mọi lệnh gọi đến các câu lệnh CASE, CONTINUE-WHEN, EXECUTE IMMEDIATE, EXIT-WHEN, LOOP và RETURN.

Hành vi của gợi ý tiền biên dịch PRAGMA INLINE thay đổi phụ thuộc vào xác lập của biến session PLSQL_OPTIMIZE_LEVEL. Các thường trình con được inline khi PLSQL_OPTIMIZE_LEVEL được xác lập sang 2 và chỉ được cho một quyền ưu tiên cao khi được xác lập sang 3. Nếu PLSQL_OPTIMIZE_LEVEL được xác lập sang 1, các chương trình con chỉ được inline khi trình biên dịch xem nó là cần thiết.

Sau đây là thủ tục add_contact chuyển theo tham chiếu:

-- This is found in create_add_contact2.sql on the publisher's web site.

```
CREATE OR REPLACE procedure add_contact
( contact_id      OUT  NUMBER -- Primary key after insert.
, member_id      IN   NUMBER -- Foreign key preceding insert.
, contact_type    IN   NUMBER
, last_name       IN   VARCHAR2
, first_name      IN   VARCHAR2
, middle_initial  IN   VARCHAR2 := NULL
, created_by      IN   NUMBER
, creation_date   IN   DATE := SYSDATE
, last_updated_by IN   NUMBER
, last_update_date IN  DATE := SYSDATE) IS
BEGIN
    -- Set savepoint so that all or nothing happens.
    SAVEPOINT add_contact;
    -- Suggest inlining the get_sequence_value function.
    PRAGMA INLINE(get_sequence_value,'YES');
    -- Assign next value from sequence and insert record.
```

```

contact_id := get_sequence_value('CONTACT_S1');
INSERT INTO contact VALUES
( contact_id
, member_id
, contact_type
, last_name
, first_name
, middle_initial
, created_by
, creation_date
, last_updated_by
, last_update_date);
EXCEPTION
WHEN others THEN
    ROLLBACK TO add_contact;
    RAISE_APPLICATION_ERROR(-20001,SQLERRM);
END add_contact;
/

```

Thủ tục `add_contact` cung cấp một gợi ý `PRAGMA INLINE` để gợi ý rằng trình biên dịch inline hàm `get_sequence_value`. Đây là điều mà bạn nên xem xét khi các đơn vị chương trình gọi những thường trình con lưu trữ khác. Nó không được đưa vào các ví dụ tiếp theo, nhưng có thể bạn đưa nó vào mã sản xuất.

Thủ tục tiếp theo điều khiển việc insert vào các bảng `address` và `street_address`. Nó định nghĩa giá trị khóa ngoại là một biến chế độ `IN` (read-only). Tương tự như thủ tục `add_contact` đã định nghĩa khóa ngoại `number_id`.

Thủ tục `add_address` là

```

-- This is found in create_add_contact2.sql on the publisher's web site.
CREATE OR REPLACE procedure add_address
( address_id          OUT  NUMBER — Primary key after insert.
, contact_id         IN   NUMBER — Foreign key preceding insert.
, address_type       IN   NUMBER := NULL
, street_address     IN   VARCHAR2 := NULL
, city               IN   VARCHAR2 := NULL
, state_province     IN   VARCHAR2 := NULL

```

```

, postal_code      IN      VARCHAR2 := NULL
, created_by      IN      NUMBER
, creation_date   IN      DATE := SYSDATE
, last_updated_by IN      NUMBER
, last_update_date IN      DATE := SYSDATE) IS
    -- Declare surrogate key variables.
    street_address_id NUMBER;
BEGIN
    -- Set savepoint so all or nothing happens.
    SAVEPOINT add_address;
    -- Check data is present for insert to ADDRESS table.
    IF address_type IS NOT NULL      AND
       city IS NOT NULL              AND
       state_province IS NOT NULL    AND
       postal_code IS NOT NULL       THEN
    -- Assign next value from sequence and insert record.
    address_id := get_sequence_value('ADDRESS_S1');
    INSERT INTO address VALUES
    ( address_id
    , contact_id
    , address_type
    , city
    , state_province
    , postal_code
    , created_by
    , creation_date
    , last_updated_by
    , last_update_date);
    -- Check data is present for insert to ADDRESS table.
    IF street_address IS NOT NULL THEN
    -- Assign next value from sequence and insert record.
    street_address_id := get_sequence_value('STREET_ADDRESS_S1');
    INSERT INTO street_address VALUES
    ( street_address_id
    , address_id

```



```

        , street_address
        , created_by
        , creation_date
        , last_updated_by
        , last_update_date);
    END IF;
END IF;
EXCEPTION
    WHEN others THEN
        ROLLBACK TO add_address;
        RAISE_APPLICATION_ERROR(-20001,SQLERRM);
END add_address;
/

```

Sau khi tạo hàm độc lập và hai thủ tục, bạn viết lại khối nặc danh để thực hiện các lệnh gọi độc lập đến các thủ tục `add_contact` và `add_address`. Khối nặc danh như sau:

-- This is found in `create_add_contact2.sql` on the publisher's web site.

```

DECLARE
    -- Declare surrogate key variables.
    member_id NUMBER;
    contact_id NUMBER;
    address_id NUMBER;

    -- Declare local function to get type.
    FUNCTION get_type
    ( table_name VARCHAR2
    , column_name VARCHAR2
    , type_name VARCHAR2) RETURN NUMBER IS
    retval NUMBER;
BEGIN
    SELECT      common_lookup_id
    INTO        retval
    FROM        common_lookup
    WHERE       common_lookup_table = table_name
    AND         common_lookup_column = column_name

```

```

AND          common_lookup_type = type_name;
RETURN      retval;
END get_type;
-- Define autonomous function to secure surrogate key values.
FUNCTION get_member_id RETURN NUMBER IS
  PRAGMA AUTONOMOUS_TRANSACTION;
  id_value NUMBER;
BEGIN
  SELECT member_s1.nextval INTO id_value FROM dual;
  RETURN id_value;
END;
BEGIN
  -- Declare surrogate key variables.
  member_id := get_member_id;
  INSERT INTO member VALUES
  ( member_id
  ,(SELECT      common_lookup_id
  FROM          common_lookup
  WHERE         common_lookup_table = 'MEMBER'
  AND           common_lookup_column = 'MEMBER_TYPE'
  AND           common_lookup_type = 'GROUP')
  , '4563-98-71'
  , '5555-6363-1212-4343'
  ,(SELECT      common_lookup_id
  FROM          common_lookup
  WHERE         common_lookup_table = 'MEMBER'
  AND           common_lookup_column = 'CREDIT_CARD_TYPE'
  AND           common_lookup_type = 'VISA_CARD')
  , 3
  , SYSDATE
  , 3
  , SYSDATE);
  -- Call procedure to insert records in related tables.
  add_contact( member_id => member_id
  , contact_id => contact_id — This is an OUT mode variable.

```

```

        , contact_type
get_type('CONTACT','CONTACT_TYPE','CUSTOMER')
        , last_name => 'Rodriguez'
        , first_name => 'Alex'
        , created_by => 3
        , last_updated_by => 3);
-- Call procedure to insert records in related tables.
add_address( address_id => address_id
        , contact_id => contact_id — This is an OUT mode variable.
        , address_type => get_type('ADDRESS','ADDRESS_TYPE','HOME')
        , street_address => 'East 161st Street'
        , city => 'Bronx'
        , state_province => 'NY'
        , postal_code => '10451'
        , created_by => 3
        , last_updated_by => 3);

END;
/

```

Lệnh gọi đến `add_contact` trả về một giá trị cho cột khóa chính. Lệnh gọi tiếp theo đến thủ tục `add_address` sử dụng giá trị đó làm một giá trị khóa ngoại. Cho dù bạn thực thi một thủ tục chuyển theo giá trị hoặc chuyển theo tham chiếu đều tùy thuộc vào nhiều yếu tố. Lựa chọn thường là giữa khả năng tái sử dụng và khả năng quản lý.

Các đơn vị nhỏ hơn như các thủ tục chuyển theo tham chiếu có thể tái sử dụng nhiều hơn, nhưng chúng khó quản lý hơn. Chúng có thể tồn tại cho mọi table hoặc view trong ứng dụng. Các đơn vị lớn, như những thủ tục chuyển theo giá trị cho bạn quản lý các tiến trình phức tạp trong một hộp đen (black box). Chúng có khuynh hướng thực thi những gì đôi khi được gọi là các đơn vị dòng làm việc (workflow unit). Các thủ tục chuyển theo giá trị thường tập trung vào tiến trình hơn so với các wrapper tập trung vào dữ liệu và duy trì ít tốn kém hơn. Tuy nhiên, bạn nên chú ý rằng các thủ tục chuyển theo tham chiếu lý tưởng cho việc hỗ trợ các ứng dụng trên web không trạng thái.

Quy tắc chung tốt nhất có lẽ là tất cả thủ tục nên tập trung vào các hoạt động xoay quanh tiến trình. Sau đó, bạn chọn thường trình con nào thích hợp tốt nhất cho tác vụ của bạn trên cơ sở ngoại lệ.

Bây giờ bạn đã xem bốn loại thường trình con được hỗ trợ trong PL/SQL. Các ví dụ đã được trình bày nhằm hướng dẫn cách sử dụng mỗi loại thường trình con. Bây giờ thách thức là làm thế nào bạn thiết kế các ứng dụng.

Tóm tắt

Bây giờ bạn đã có một kiến thức về phạm vi giao tác (transaction scope) và cách thực thi các hàm và thủ tục. Điều này bao gồm đánh giá khi nào chọn một hàm so với một thủ tục và ngược lại.

CHƯƠNG 7

CÁC TẬP HỢP

Có ba loại tập hợp trong họ sản phẩm Oracle Database 11g. Chúng là các kiểu dữ liệu varray, nested table và associative array (mảng kết hợp). Các tập hợp là những cấu trúc mạnh bởi vì chúng cho phép phát triển những chương trình quản lý các tập hợp dữ liệu lớn trong bộ nhớ.

Bạn có thể xây dựng các tập hợp của bất kỳ kiểu dữ liệu SQL hoặc PL/SQL. Các tập hợp của các kiểu dữ liệu SQL làm việc trong các môi trường SQL và PL/SQL nhưng các tập hợp của các kiểu dữ liệu PL/SQL thì không. Chúng chỉ làm việc trong PL/SQL.

Chương này giải thích cách định nghĩa và làm việc với các tập hợp trong PL/SQL. Chương cũng đề cập đến việc sử dụng các tập hợp làm các cột cơ sở dữ liệu. Chương này bao gồm những chủ điểm sau đây:

- Các loại tập hợp
- Varray
- Nested table
- Associative array
- Các toán tử tập hợp
- Collection API

Các tập hợp (collection) là những cấu trúc lập trình chứa các tập hợp những thứ tương tự. Các tập hợp rơi vào hai hạng mục: array (mảng) và list (danh sách). Các mảng thường có một kích cỡ vật lý được cấp phát khi bạn định nghĩa chúng, trong khi các danh sách không có giới hạn vật lý được áp đặt. Hiển nhiên, bộ nhớ có sẵn để xử lý trong SGA đòi hỏi kích cỡ tối đa của một số danh sách rất lớn.

Những danh sách này thường được tạo index bằng một loạt các số trình tự bắt đầu với 0 hoặc 1 và tăng mỗi lần một giá trị. Sử dụng các giá trị index số trình tự bảo đảm bạn có thể sử dụng index để truyền ngang qua một danh sách hoàn chỉnh bằng cách tăng hoặc giảm mỗi lần một giá trị trong một vòng lặp. Hoặc, các danh sách có thể được tạo index bằng các số không trình tự hoặc các chuỗi duy nhất. Các danh sách được gọi là các associative array (mảng kết hợp) khi chúng có thể được tạo index bằng các số không trình tự hoặc các chuỗi duy nhất.

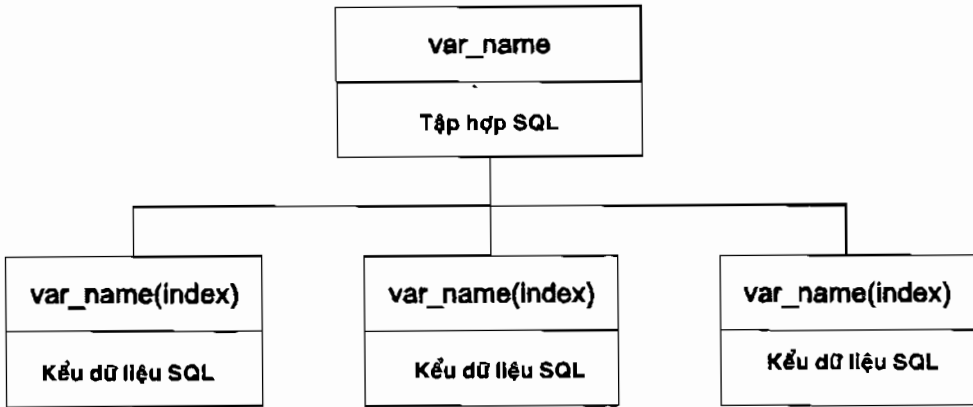
Hình 7.1 minh họa một tập hợp các chuỗi dưới dạng một cây đảo ngược tượng trưng cho một tập hợp một chiều. Nó sử dụng một index được đánh số trình tự và làm việc với bất kỳ kiểu dữ liệu SQL hoặc kiểu vô hướng PL/SQL hoặc kiểu đối tượng do người dùng định nghĩa. Điều kiện về các kiểu dữ liệu PL/SQL là chúng có thể được sử dụng trong ngữ cảnh của các khối PL/SQL.

Các giá trị index trở thành định danh (identifier) để truy cập các phần tử riêng lẻ bên trong một biến tập hợp. Như được thảo luận trong chương 3, các tên biến là những định danh và bao gồm các tên biến bao gồm những giá trị index.

Bạn có thể tạo các tập hợp giả đa chiều khi bạn sử dụng một loại đối tượng SQL do người dùng định nghĩa làm phần tử cơ sở của một tập hợp. Tuy nhiên, các loại đối tượng do người dùng định nghĩa đòi hỏi các phương thức tạo (constructor) đặc biệt và các phương thức truy cập static và instance. Chương 14 đề cập đến các loại đối tượng và trình bày cách xây dựng các tập hợp của các loại đối tượng.

Các tập hợp đa chiều không được hỗ trợ dưới dạng các kiểu dữ liệu SQL. Tuy nhiên, bạn có thể xây dựng các tập hợp đa chiều dưới dạng các kiểu dữ liệu PL/SQL. Các phần tử tập hợp đa chiều là các cấu trúc record. Bạn có thể truy cập những cấu trúc record bên trong PL/SQL, hoặc bạn có thể xây dựng các hàm pipelined để truy cập nội dung của chúng trong SQL. Chương 6 minh họa cách sử dụng các hàm pipelined chuyển đổi các tập hợp đa chiều thành các bảng gộp (aggregate table) để sử dụng trong những câu lệnh SQL.

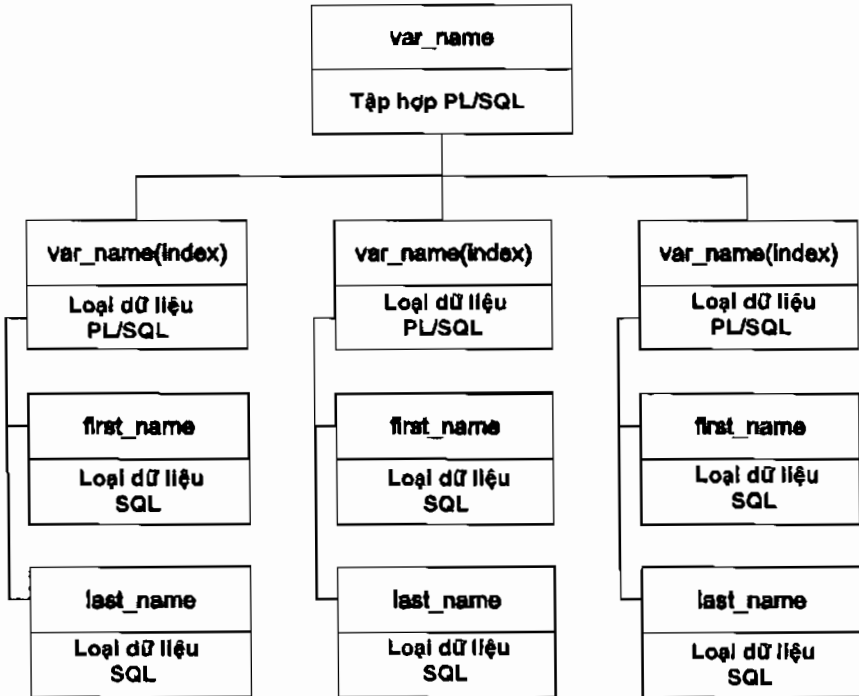
Trong khi hình 7.2 minh họa các phần tử record dưới dạng các kiểu dữ liệu SQL, bạn cũng có thể sử dụng các kiểu dữ liệu PL/SQL. Các loại record PL/SQL cũng có thể là những tập hợp của các loại record PL/SQL khác. Cú pháp khi bạn xếp lồng các tập hợp trở nên phức tạp hơn. Bạn nên xem xét tại sao bạn đòi hỏi các tập hợp xếp lồng và so sánh những chiến lược khác trước khi chấp nhận chúng làm giải pháp của bạn.



Hình 7.1 Một sơ đồ cây đảo ngược của một tập hợp kiểu dữ liệu SQL một chiều

Bạn cũng có thể tạo các mảng nhiều chiều được gọi là các tập hợp đa cấp. Bạn làm điều này bằng cách đưa vào các tập hợp dưới dạng các phần tử bên trong những tập hợp.

Các phần được tổ chức để dựa vào những khái niệm khi bạn làm việc qua chương. Nếu bạn muốn nhảy về phía trước, hãy xem qua các phần trước đó để dễ dàng nắm vững nội dung tiếp theo.



Hình 7.2 Một sơ đồ cây đảo ngược của một tập hợp loại record PL/SQL đa chiều

Các loại tập hợp

Các tập hợp VARRAY và NESTED TABLE có thể được định nghĩa là các kiểu dữ liệu SQL và PL/SQL. Là các kiểu dữ liệu SQL, chúng là các mảng một chiều gồm những giá trị vô hướng hoặc loại đối tượng. Chúng cũng định nghĩa các kiểu dữ liệu cột do người dùng xác định. Các kiểu dữ liệu VARRAY và NESTED TABLE là những cấu trúc được tạo index bởi các số nguyên trình tự (sử dụng các số nguyên dựa vào 1). Các cấu trúc được tạo index theo trình tự không cho phép các khoảng hở trong những giá trị index và còn được gọi là các cấu trúc được tập hợp dày đặc. Trong khi VARRAY có một số phần tử cố định khi được định nghĩa, NESTED TABLE thì không.

Mảng kết hợp (associative array), trước đó được gọi là một bảng (table) PL/SQL, chỉ là một kiểu dữ liệu PL/SQL. Các kiểu dữ liệu associative array chỉ được tham chiếu trong một phạm vi PL/SQL. Chúng thường được định nghĩa trong các thông số PL/SQL khi chúng sẽ được sử dụng bên ngoài từ một chương trình nặc danh hoặc chương trình khối định danh. Các kiểu dữ liệu associative array hỗ trợ các index số và index chuỗi. Các index số cho các mảng kết hợp không cần theo trình tự và là những cấu trúc không trình tự. Những cấu trúc không trình tự có thể có những khoảng hở trong các dãy index và được gọi là những cấu trúc được tập hợp thưa. Các mảng kết hợp được định kích cỡ động và như kiểu dữ liệu NESTED TABLE, không có kích cỡ cố định.

Tất cả ba loại truy cập Oracle Collection API, nhưng mỗi loại sử dụng một tập hợp phương thức khác nhau. Những thay đổi gần đây đối với OCI8 cho phép nó hỗ trợ biến vô hướng, các mảng biến vô hướng và biến cursor tham chiếu cho những ngôn ngữ bên ngoài như C, C++, C#, Java và PHP. Các kiểu dữ liệu VARRAY và NESTED TABLE đòi hỏi bạn sử dụng lớp OCI-Collection để truy cập chúng bên ngoài từ môi trường SQL*Plus. OCI8 cũng có một hàm mới hỗ trợ việc chuyển theo tham chiếu một table PL/SQL.

Bảng 7.1 so sánh các loại tập hợp. Bạn nên chú ý rằng trong khi kích cỡ thì động, các vùng bộ nhớ SGA và PGA bị ràng buộc bởi các tham số khởi tạo cơ sở dữ liệu. Khi làm việc với những loại tập hợp này, bạn đạt được thông lượng đáng kể miễn là bạn không làm cạn kiệt các nguồn tài nguyên bộ nhớ.

Quyết định chọn loại tập hợp đáp ứng tốt nhất nhu cầu lập trình là điều quan trọng. Bạn nên cẩn thận xem xét những ưu điểm và khuyết điểm của mỗi loại tập hợp. Sau đây là một hướng dẫn chung về việc chọn tập hợp thích hợp:

- Sử dụng một varray khi kích cỡ vật lý của tập hợp thì tĩnh và tập hợp có thể được sử dụng trong các table. Các varray là thứ gần giống

nhất với các mảng (array) trong những ngôn ngữ lập trình, chẳng hạn như Java, C, C++ hoặc C#.

- Sử dụng nested table (bảng xếp lồng) khi kích cỡ vật lý không được biết do những thay đổi run-time và khi kiểu có thể được sử dụng trong các table. Các nested tables giống như các list và bag trong những ngôn ngữ lập trình khác.
- Sử dụng các mảng kết hợp (associative array) khi kích cỡ vật lý không được biết do những thay đổi run-time và khi kiểu sẽ không được sử dụng trong các table. Các mảng kết hợp lý tưởng cho những giải pháp lập trình chuyển, chẳng hạn như sử dụng các ánh xạ (map) và tập hợp (set).

Bảng 7.1 So sánh loại tập hợp

Loại tập hợp	Mô tả	Chỉ số dưới (subscript)	Kích cỡ
Associative array (index-by-table)	Associative array là tên được giới thiệu trong Oracle 10g cho một cấu trúc quen thuộc. Có lẽ bạn đã biết đây là các index-by-tables trong Oracle 8 đến Oracle 9i và có lẽ là các table PL/SQL trong Oracle 7. Chúng đã biến đổi về phía trước trong Oracle 11g và xứng đáng có một tên mới. Chúng vẫn là các mảng được tập hợp thưa, nghĩa là việc đánh số không cần phải theo trình tự, chỉ duy nhất. Bây giờ chúng hỗ trợ các subscript (chỉ số dưới) vốn là những số nguyên hoặc chuỗi duy nhất. Sự thay đổi này di chuyển một cấu trúc quen thuộc và mạnh mẽ từ một giả mảng hoặc danh sách mảng được tập hợp thưa sang một kiểu dữ liệu ngôn ngữ lập trình có cấu trúc chuẩn được gọi là các list hoặc map.	Các số nguyên trình tự hoặc không trình tự hoặc các chuỗi duy nhất	Động (dynamic)
NESTED TABLE	NESTED TABLE đã được giới thiệu trong Oracle 8. Ban đầu chúng được định nghĩa là các mảng được tập hợp dày đặc nhưng có thể được tập hợp thưa khi các record bị xóa. Chúng có thể được lưu trữ trong các table thường trực và được	Các số nguyên trình tự	Động

truy cập bởi SQL. Chúng cũng có thể được mở rộng động và hành động giống nhiều như các bag và tập hợp lập trình truyền thống hơn là các mảng. Một hệ quả khác là lớp ArrayList được giới thiệu trong Java 5. Chúng có thể chứa một biến vô hướng hoặc loại đối tượng do người dùng định nghĩa khi chúng được sử dụng làm các kiểu dữ liệu SQL. Các tập hợp phạm vi SQL là các danh sách một chiều của các kiểu dữ liệu SQL hợp lệ. Chúng cũng chứa một danh sách của một hoặc nhiều kiểu dữ liệu phức hợp (các cấu trúc record PL/SQL) khi chúng làm việc độc quyền trong một phạm vi PL/SQL.

Varray

Varray đã được giới thiệu trong Oracle 8. Chúng là các mảng được tập hợp dày đặc và có hành vi như các mảng lập trình truyền thống. Chúng có thể được lưu trữ trong các table thường trực và được truy cập bởi SQL. Lúc tạo chúng có một kích cỡ cố định vốn không thể thay đổi. Như Nested Table, varray có thể chứa một biến vô hướng hoặc loại đối tượng do người dùng định nghĩa khi chúng được sử dụng làm các kiểu dữ liệu SQL. Như được đề cập trước đó, các tập hợp phạm vi SQL là các danh sách một chiều của các kiểu dữ liệu SQL hợp lệ. Varray cũng có thể chứa một danh sách của một hoặc nhiều kiểu dữ liệu phức hợp (các cấu trúc record PL/SQL) khi chúng làm việc độc quyền trong một phạm vi PL/SQL.

Các số nguyên Cố định
trình tự

Các mục dưới đây đề cập đến các kiểu dữ liệu VARRAY, NESTED TABLE và các kiểu dữ liệu associative array, và Oracle Collection API. Những mục này được thiết kế để đọc theo thứ tự nhưng hỗ trợ nhà phát triển có kinh nghiệm tìm kiếm những lời giải thích chi tiết.

Varrays

Varrays là những cấu trúc một chiều của các kiểu dữ liệu Oracle 11g SQL hoặc PL/SQL. Bạn có thể sử dụng các varray trong các định nghĩa table, record và đối tượng, và sau đó bạn có thể truy cập chúng trong SQL hoặc PL/SQL. Chúng là các mảng theo nghĩa truyền thống của các ngôn ngữ lập trình bởi vì chúng có một kích cỡ cố định và sử dụng một index số trình tự. Chúng không giống như các mảng trong Java, C, C++ và C#.

Định nghĩa và sử dụng Varray làm những cấu trúc chương trình PL/SQL

Cú pháp để định nghĩa một varray trong một đơn vị chương trình PL/SQL là

```
TYPE type_name IS (VARRAY | VARYING ARRAY) (size_limit)
OF element_type [ NOT NULL ];
```

Type name thường là một chuỗi theo sau là một dấu gạch dưới và từ array. Nhiều nhà lập trình và những người quản lý cấu hình thấy nó là một mẫu hữu dụng để cải thiện khả năng đọc mã. Nó cũng là quy ước được sử dụng trong chương.

Một cú pháp VARRAY hoặc VARYING ARRAY có thể được sử dụng, nhưng cái trước phổ biến hơn nhiều. Giới hạn kích cỡ là một giá trị bất buộc. Nó là một số nguyên dương đưa ra số phần tử tối đa trong varray. Element type có thể là bất kỳ kiểu dữ liệu Oracle 11g hoặc một kiểu dữ liệu do người dùng định nghĩa. Cho phép các giá trị rỗng trong các varrays là lựa chọn mặc định. Nếu các giá trị rỗng không được phép, bạn phải loại trừ chúng bằng mệnh đề NOT NULL.

Chương trình mẫu sau đây minh họa việc định nghĩa, khai báo và khởi tạo một varray của các số nguyên trong một đơn vị chương trình PL/SQL. Một số nguyên là một kiểu con (subtype) của kiểu dữ liệu số Oracle 11g.

Các giá trị index subscript bắt đầu tại 1, chứ không phải 0. Điều này nhất quán với hành vi của các table index-by trong Oracle 8 đến Oracle 9 và các table PL/SQL trong Oracle 7. Hầu hết các ngôn ngữ lập trình bao gồm Java, C, C++ và C# sử dụng các giá trị index subscript (chỉ số dưới) bắt đầu với 0.

```

-- This is in create_varray1.sql on the publisher's web site.
DECLARE
-- Define a varray with a maximum of 3 rows.
TYPE Integer_varray IS VARRAY(3) OF INTEGER;

-- Declare the varray with null values.
varray_integer INTEGER_VARRAY := integer_varray(NULL,NULL,NULL);
BEGIN
-- Print initialized null values.
dbms_output.put_line('Varray initialized as nulls. ');
dbms_output.put_line('-----');
FOR i IN 1..3 LOOP
    dbms_output.put ('Integer Varray [ ' || i || ' ] ');
    dbms_output.put_line(' [ ' || varray_integer(i) || ' ] ');
END LOOP;
-- Assign values to subscripted members of the varray.
varray_integer(1) := 11;
varray_integer(2) := 12;
varray_integer(3) := 13;

-- Print initialized null values.
dbms_output.put (CHR(10)); -- Visual line break.
dbms_output.put_line('Varray initialized as values. ');
dbms_output.put_line('-----');
FOR i IN 1..3 LOOP
    dbms_output.put_line('Integer Varray [ ' || i || ' ] '
        || ' [ ' || varray_integer(i) || ' ] ');
END LOOP;
END;
/

```

Chương trình mẫu định nghĩa một tập hợp vô hướng cục bộ, khai báo một biến tập hợp được khởi tạo, in các phần tử tập hợp giá trị rỗng, gán những giá trị vào các phần tử, và in lại các giá trị phần tử tập hợp. Bạn cũng sẽ thấy cách khởi tạo một tập hợp, truy cập nội dung của một phần tử tập hợp và gán một giá trị vào phần tử tập hợp trong chương trình mẫu.

Sau đây là kết quả từ chương trình:

```
Varray initialized as nulls.
```

```
Integer Varray [1] []
```

```
Integer Varray [2] []
```

```
Integer Varray [3] []
```

```
Varray initialized as values.
```

```
Integer Varray [1] [11]
```

```
Integer Varray [2] [12]
```

```
Integer Varray [3] [13]
```

Nếu bỏ qua bất kỳ bước nào, bạn sẽ gặp phải các ngoại lệ. Ngoại lệ mà hầu hết nhà phát triển mới gặp phải là một tập hợp không được khởi tạo, ORA-06531. Nó xảy ra bởi vì tối thiểu bạn phải khởi tạo một varray phần tử rỗng bằng cách gọi loại tập hợp như được trình bày ở đây:

```
Varray_integer INTEGER_VARRAY := integer_varray();
```

Ngoại lệ này không cấp phát không gian cho bất kỳ phần tử trong varray. Chương trình mẫu khởi tạo varray với những giá trị rỗng bởi vì các giá trị rỗng (null) được cho phép. Cũng có thể khởi tạo biến với những giá trị. Bạn khởi tạo biến bằng cách sử dụng tên loại varray và các dấu ngoặc đơn xung quanh những giá trị. Khi bạn khởi tạo một varray, bạn xác lập số hàng khởi tạo thật sự. Sử dụng phương thức Collection API COUNT sẽ trả về số phần tử với không gian cấp phát. Việc sử dụng phương thức này sẽ được minh họa trong chương trình mẫu tiếp theo.

Số phần tử tối đa trong varray là ba. Chương trình cấp phát bộ nhớ và một giá trị index chỉ khi bạn khởi tạo các phần tử. Bạn có thể test điều này bằng cách biên tập chương trình và thay đổi việc khởi tạo từ ba giá trị rỗng thành hai giá trị rỗng. Khi bạn chạy chương trình, bạn đưa ra một ngoại lệ ORA-06533 bên trong vòng lặp FOR đầu tiên. Thông báo nói rằng bạn đã cố gắng truy cập một subscript ngoài số phần tử ra. Ngoại lệ muốn nói rằng chỉ số 3 không có sẵn. Nó không hiện hữu. Trong khi bạn đã định nghĩa varray là có kích cỡ ba phần tử, bạn đã khởi tạo nó là chỉ kích cỡ hai phần tử. Do đó, variable chỉ có hai chỉ số dưới hợp lệ, 1 và 2.

Nếu gặp phải lỗi này, bạn có thể kiểm tra tài liệu Oracle 11g. Bạn sẽ thấy có một phương thức Collection API EXTEND cho các tập hợp và nó quá tải. Collections API đòi hỏi bạn khởi tạo một hàng và sau đó gán một giá trị.

Bạn thêm một hàng bằng cách sử dụng phương thức Collection API `EXTEND` không có một tham số thật sự hoặc có một tham số thật sự. Nếu bạn sử dụng tham số, nó là số phần tử để khởi tạo. Nó không thể vượt quá hiệu giữa số phần tử có thể có và số phần tử thật sự được định nghĩa bởi `varray`. Bạn sẽ đọc thêm về việc sử dụng những phương thức này trong mục “Oracle 11g Collection API” ở cuối chương.

Chương trình sau đây minh họa việc khởi tạo không có các hàng trong phần khai báo. Sau đó, nó minh họa việc khởi tạo và gán động trong phần thực thi:

```
-- This is in create_varray2.sql on the publisher's web site.
DECLARE
    -- Define a varray of integer with 3 rows.
    TYPE integer_varray IS VARRAY(3) OF INTEGER;

    -- Declare an array initialized as a no-element collection.
    varray_integer INTEGER_VARRAY := integer_varray();
BEGIN
    -- Allocate space as you increment the index.
    FOR i IN 1..3 LOOP
        varray_integer.EXTEND; -- Allocates space in the collection.
        varray_integer(i) := 10 + i; -- Assigns a value to the indexed value.
    END LOOP;

    -- Print initialized array.
    dbms_output.put_line('Varray initialized as values. ');
    dbms_output.put_line('-----');
    FOR i IN 1..3 LOOP
        dbms_output.put ('Integer Varray [ ' || i || ' ' || i || ' ' || i || ' ' || ' ' || ' ');
        dbms_output.put_line(' [ ' || varray_integer(i) || ' ' || ' ' || ' ' || ' ');
    END LOOP;
END;
/
```

Như ví dụ trước đó, chương trình định nghĩa một loại tập hợp cục bộ. Sự khác biệt là chương trình không cấp phát không gian và tập hợp lại các giá trị rỗng trong quá trình khai báo biến. Nó thật sự tạo một tập hợp không phần tử. Bạn phải cấp phát không gian bằng phương thức `EXTEND` của Collection API trước khi bạn thêm một phần tử vào loại tập hợp này. Đây là những gì được thực hiện bên trong vòng lặp `FOR` đây.

Kết quả từ chương trình là

Varray initialized as values.

Integer Varray [1] [11]

Integer Varray [2] [12]

Integer Varray [3] [13]

Bây giờ bạn có những điểm cơ bản để xây dựng những cấu trúc varray bên trong các đơn vị chương trình PL/SQL. Sức mạnh và những tiện ích quản lý của các phương thức tập hợp sẽ nâng cao khả năng sử dụng những cấu trúc này. Trong khi mục này đã đề cập đến các phương thức Collection API để minh họa những vấn đề khởi tạo, chúng được đề cập chuyên sâu ở mục sau trong chương. Bằng cách làm việc qua những ví dụ ở đó, bạn sẽ có thể thấy bạn áp dụng những phương thức này qua các loại tập hợp.

Định nghĩa và sử dụng Varray làm các loại đối tượng trong SQL

Cú pháp để định nghĩa một loại đối tượng của varray trong cơ sở dữ liệu là

```
CREATE OR REPLACE TYPE type_name AS (VARRAY | VARYING ARRAY)
(size_limit)
OF element_type [ NOT NULL ];
```

Như được thảo luận, tên kiểu (type name) thường là một chuỗi theo sau là một dấu gạch dưới và từ varray. Nhiều nhà lập trình và những người quản lý cấu hình thường thấy đây là một mẫu hữu dụng để cải thiện khả năng đọc mã. Nó cũng là quy ước được sử dụng trong chương cho các loại cấu trúc và đối tượng PL/SQL.

Như với một cấu trúc kiểu PL/SQL, cú pháp VARRAY hoặc VARYING ARRAY có thể được sử dụng. Cái trước phổ biến hơn nhiều. Giới hạn kích cỡ là một giá trị bắt buộc. Nó là một số nguyên dương, số phần tử tối đa trong varray. Kiểu phần tử có thể là bất kỳ kiểu dữ liệu Oracle 11g hoặc một kiểu dữ liệu do người dùng định nghĩa. Cho phép các giá trị rỗng trong varray là lựa chọn mặc định. Nếu các giá trị rỗng không được cho phép, bạn phải loại trừ chúng bằng mệnh đề NOT NULL.

Dòng mã sau đây tạo một loại đối tượng do người dùng định nghĩa của varray với giới hạn ba phần tử:

```
-- This is in create_varray3.sql on the publisher's web site.
```

```
CREATE OR REPLACE TYPE integer_varray AS VARRAY(3) OF INTEGER;
/
```

Sau đó chương trình PL/SQL khối nặc danh sau đây sử dụng loại đối tượng varray bằng cách khai báo và khởi tạo một biến dựa vào kiểu dữ liệu SQL:

-- This is in create_varray3.sql on the publisher's web site.

```

DECLARE
    varray_integer INTEGER_VARRAY := integer_varray(NULL,NULL,NULL);
BEGIN
    -- Assign values to replace the null values.
    FOR i IN 1..3 LOOP
        varray_integer(i) := 10 + i;
    END LOOP;
    -- Print the initialized values.
    dbms_output.put_line('Varray initialized as values.');
```

```

    dbms_output.put_line('-----');
    FOR i IN 1..3 LOOP
        dbms_output.put ('Integer Varray [' || i || ' ] ');
        dbms_output.put_line([' || varray_integer(i) || ' ] ');
    END LOOP;
END;
/
```

Ví dụ phản ánh khái niệm chương trình trước cho việc biến bây giờ là một kiểu dữ liệu tập hợp SQL do người dùng định nghĩa. Nó in kết quả sau đây:

```
Varray initialized as values.
```

```
Integer Varray [1] [11]
```

```
Integer Varray [2] [12]
```

```
Integer Varray [3] [13]
```

Lợi ích của việc định nghĩa loại đối tượng varray là nó có thể được tham chiếu từ bất kỳ chương trình nào được phép sử dụng nó, trong khi một cấu trúc kiểu PL/SQL giới hạn chỉ trong đơn vị chương trình. Các đơn vị chương trình có thể là những chương trình khối nặc danh như ví dụ hoặc các thủ tục lưu trữ hoặc package trong cơ sở dữ liệu. Chỉ cái sau cho phép các tham chiếu bởi những chương trình PL/SQL khác vốn có các quyền đối với package. Xem chương 9 để biết chi tiết về việc tạo và sử dụng các package.

Tất cả varray cho đến điểm này tận dụng hành vi mặc định cho phép các giá trị rỗng. Bắt đầu với hành vi mặc định thì luôn hơi rõ ràng hơn. Sau khi bạn nắm vững cú pháp cơ bản và lựa chọn mặc định để dễ dàng, khai báo và khởi tạo các varray, có một câu hỏi cần được giải quyết. Khi nào, tại sao và làm thế nào bạn cho phép hoặc không cho phép các hàng rỗng?

Đây là một câu hỏi. Trong các chương trình mẫu nhỏ trong sách, dường như nó có thể không quan trọng quá nhiều. Thật ra, nó quan trọng rất nhiều. Các varray là cấu trúc gần gũi nhất liên quan đến các mảng (array) ngôn ngữ lập trình chuẩn. Các mảng là những cấu trúc đòi hỏi việc quản lý tập trung. Theo quy tắc chung, các mảng luôn nên dày đặc. Dày đặc nghĩa là sẽ không có bất kỳ khoảng hở trong dãy giá trị index. Nó cũng có nghĩa là không nên có các khoảng hở. Bạn không nên cho phép các giá trị rỗng khi bạn muốn một varray hành động như một cấu trúc mảng chuẩn.

Cho phép các giá trị rỗng (null) trong varray bảo đảm rằng bạn có thể gặp phải chúng trong luồng dữ liệu. Oracle 11g không cho phép tạo các khoảng hở trong những giá trị index. Nếu bạn không muốn viết vô số thường trình xử lý lỗi cho các mảng với dữ liệu thiếu, bạn nên xem xét ghi đè hành vi mặc định. Không cho phép các giá trị rỗng trong varrays để đơn giản hóa việc truy cập dữ liệu và xử lý lỗi.

Bây giờ bạn sẽ học cách không cho phép các giá trị rỗng trong các varray. Tác động chính của việc làm như vậy sẽ được cảm nhận khi bạn khởi tạo chúng. Ví dụ, nếu bạn đã tái định nghĩa loại đối tượng varray được trong chương trình trước để không cho phép các giá trị rỗng, chương trình thất bại. Bạn đưa ra một ngoại lệ PLS-00567 bởi vì bạn đang cố chuyển một giá trị đến một cột bị ràng buộc not-null (không rỗng).

Bạn nên tạo một loại tập hợp SQL không cho phép các giá trị rỗng cho ví dụ kế tiếp. Ví dụ sau đây tạo kiểu dữ liệu bắt buộc:

-- This is in create_varray4.sql on the publisher's web site.

```
CREATE OR REPLACE TYPE integer_varray
  AS VARRAY(100) OF INTEGER NOT NULL;
```

/

Ví dụ sau đây cấp phát 100 record. Nó làm như vậy mà không khởi tạo dữ liệu dưới dạng các giá trị rỗng bằng cách tận dụng phương thức LIMIT của Collection API. Bạn sẽ thấy Collection API được đề cập trong chương.

-- This is in create_varray4.sql on the publisher's web site.

```
DECLARE
  varray_integer INTEGER_VARRAY := integer_varray();
```

```

BEGIN
    FOR i IN 1..varray_integer.LIMIT LOOP
        varray_integer.EXTEND;
    END LOOP;
    dbms_output.put ('Integer Varray Initialized ');
    dbms_output.put_line([' | | varray_integer.COUNT | | ']);
END;
/

```

Chương trình tạo một tập hợp varray bằng cách cấp phát không gian mà không gán các giá trị rỗng một cách tường minh. Tuy nhiên, khi bạn đọc các phần tử trong varray, chúng được xem là những giá trị rỗng. Nó in kết quả sau đây:

```
Integer Varray Initialized [100]
```

Bạn đã phát triển những kỹ năng với việc sử dụng các loại đối tượng varray. Mục tiếp theo sẽ sử dụng những loại đối tượng varray đó để định nghĩa các table vốn sử dụng chúng làm các kiểu dữ liệu cột (column).

Định nghĩa và sử dụng Varray làm các kiểu dữ liệu Column trong các Table

Sức mạnh của các varray không giới hạn trong chỉ lập trình thủ tục. Các varray cung cấp cho Oracle 8 đến Oracle 11g những tính năng độc đáo để biểu diễn dữ liệu. Đây là lý do tại sao cơ sở dữ liệu của Oracle được gọi là một hệ thống quản lý cơ sở dữ liệu quan hệ đối tượng (ORDBMS). Nó là một chuẩn mà nhiều người đã lựa chọn.

Các cơ sở dữ liệu quan hệ làm việc trên một nguyên lý chuẩn hóa. Chuẩn hóa (normalization) là tiến trình kết nhóm dữ liệu liên quan thành những tập hợp duy nhất. Nó phụ thuộc vào hai giả thuyết cơ bản. Một là dữ liệu có thể được đặt bởi việc đánh giá ngữ nghĩa vào dạng chuẩn thứ ba hoặc cao hơn. Một giả thuyết khác là dữ liệu có thể được đặt bởi dạng chuẩn khóa miền (domain key). Đối với những mục đích thảo luận về các tập hợp Oracle 11g, sách này tối thiểu đề nghị mỗi table đáp ứng dạng chuẩn thứ ba, nghĩa là

- Các table nên có một khóa chính (primary key) nhận dạng duy nhất mỗi hàng.
- Các table không nên chứa bất kỳ cột nhiều phần như các tập hợp trong các chuỗi được phân cách bằng dấu phẩy.
- Các table không nên chứa bất kỳ sự phụ thuộc bắt cầu, nghĩa là bạn đã thiết kế một table nơi bạn có tối thiểu một table khác cho mỗi sự phụ thuộc.

Sự phụ thuộc bắc cầu (transitive dependency) nghĩa là các cột dữ liệu phụ thuộc vào một hoặc nhiều cột trước khi chúng phụ thuộc vào khóa chính. Kiểu phụ thuộc này có nghĩa là bạn có thể đặt chủ thể (miền) vào một table và có thể tạo những bất thường xóa. Quy tắc chung là thiết kế và tạo các table chứa một chủ thể đơn, có một khóa duy nhất và đặt tất cả dữ liệu trong một hàng.

Sách sử dụng một khóa đại diện làm khóa chính. Khóa đại diện là một khóa giả, nghĩa là một khóa hoặc cột vốn không phải là một phần của dữ liệu trong hàng. Lựa chọn khác là một khóa tự nhiên vốn là một cột hoặc tập hợp cột nhận dạng duy nhất mọi hàng trong một table. Các bản sao của khóa chính được đặt trong các cột của các table được tạo quan hệ dưới dạng những giá trị khóa ngoại. Các đường nối (join) giữa các table sử dụng những giá trị trong khóa chính và khóa ngoại để tương hợp với các hàng trong những câu lệnh SQL.

Ưu điểm của việc sử dụng một khóa đại diện là một sự tìm hiểu tiến hóa về table có thể thay đổi cột hoặc các cột vốn định nghĩa duy nhất một khóa tự nhiên. Khi một khóa tự nhiên phát triển, nó thay đổi table sở hữu và chỉ các table được tạo quan hệ chứa một khóa ngoại tự nhiên. Nó cũng thay đổi mọi đường nối câu lệnh SQL giữa table và những table khác. Một khóa đại diện giúp tránh điều này bởi vì điều duy nhất mà bạn sẽ cần thay đổi là index duy nhất của table chứa khóa đại diện trước tiên, sau đó là tất cả cột của khóa tự nhiên.

Sơ đồ lớp tính UML trong hình 7.3 tượng trưng cho mô hình dữ liệu của chương này. Các sơ đồ lớp tính UML thay thế Entity Relation Diagrams (ERD) truyền thống. Nhiều sơ đồ ERD truyền thống sử dụng một mô hình kỹ thuật thông tin thường được gọi thông tục là vết rạn chân chim. Một mô hình IE có vết rạn chân chim trên phía addresses của mối quan hệ.

Bản vẽ trình bày ba table hỗ trợ mã trong chương này nhưng không thích hợp trong ví dụ của hàng video. Bạn có thể download mã này từ web site của nhà sản xuất.

Kiểu dữ liệu strings là tập hợp các giá trị VARCHAR2. Có hai lựa chọn điển hình để đặt dữ liệu vào một cột tập hợp. Bạn có thể giả định các khách hàng trong mô hình dữ liệu không bao giờ có hơn hai địa chỉ đường và sau đó tạo hai cột cho địa chỉ đường được ghi nhận là street_address1 và street_address2. Đây là một dạng hủy chuẩn hóa. Hoặc, bạn có thể tuân theo việc chuẩn hóa và xây dựng một table riêng biệt cho các địa chỉ đường.

Khả năng đưa danh sách vào một table cơ sở giảm sự phức tạp thực thi vật lý. Nó loại bỏ nhu cầu nối table cơ sở với table thứ cấp. Điều này thay đổi bởi vì cái sau trở thành một danh sách trong một hàng của table cơ sở. Đây thật sự là một cách chắc chắn để thực thi những gì được

gọi là mối quan hệ phụ thuộc ID, hỗ trợ mối quan hệ nhị phân một đối nhiều.

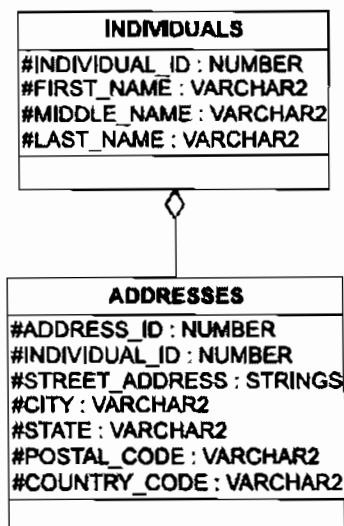
Định nghĩa các Varray trong các Table cơ sở dữ liệu

create_addressbook.sql thay đổi định nghĩa table sang mô hình mới. Định nghĩa loại đối tượng varray sau đây được cung cấp, vốn hỗ trợ việc toàn cầu hóa (globalization) bằng cách sử dụng một chuẩn Unicode.

-- This is in create_addressbook.sql on the publisher's web site.

```
CREATE OR REPLACE TYPE strings
AS VARRAY(3) OF VARCHAR2(30 CHAR);
```

/



Hình 7.3 Sơ đồ lớp tĩnh UML (mô hình ERD trong UML)

Nếu bạn tạo loại đối tượng, table addresses được tái định nghĩa để phù hợp với sơ đồ lớp tĩnh UML mới. Như bạn có thể, sơ đồ thực thi một varray của một loại catalog cơ sở dữ liệu đã biết. Table cũng duy trì tính toàn vẹn tham chiếu qua các ràng buộc cơ sở dữ liệu. Điều này đã được thực hiện trong trường hợp cơ bản.

-- This is in create_addressbook.sql on the publisher's web site.

```
CREATE TABLE individuals
( individual_id          INTEGER          NOT NULL
, first_name           VARCHAR2(30 CHAR)  NOT NULL
, middle_name          VARCHAR2(30 CHAR)
, last_name            VARCHAR2(30 CHAR)  NOT NULL
, title                VARCHAR2(10 CHAR)
```

```

, CONSTRAINT indiv_pk PRIMARY KEY(individual_id));

CREATE TABLE addresses
( address_id          INTEGER          NOT NULL
, individual_id       INTEGER NOT NULL
, street_address      STRINGS NOT NULL
, city                VARCHAR2(20 CHAR) NOT NULL
, state               VARCHAR2(20 CHAR) NOT NULL
, postal_code         VARCHAR2(20 CHAR) NOT NULL
, country_code        VARCHAR2(10 CHAR) NOT NULL
, CONSTRAINT addr_pk  PRIMARY KEY(address_id)
, CONSTRAINT addr_indiv_fk FOREIGN KEY(individual_id)
REFERENCES individuals (individual_id));

```

Cột `street_address` sử dụng loại tập hợp string. Varray là một mảng một chiều gồm ba chuỗi có chiều dài khả biến. Các chuỗi có chiều dài khả biến được định nghĩa như được ghi chú để hỗ trợ Unicode.

Sử dụng Varray trong các Table cơ sở dữ liệu

Sau khi tạo một table với một cột của một kiểu dữ liệu varray, bạn cần biết cách sử dụng nó như thế nào. Sử dụng nó đòi hỏi hiểu các phương thức truy cập ngôn ngữ xử lý dữ liệu (DML) cho các varray. Các varray không đưa ra những điều kiện duy nhất cho việc xóa vì việc xóa nằm tại cấp hàng (row). Tuy nhiên, có những điểm khác biệt đáng kể khi nói về việc sử dụng các câu lệnh insert và update.

Ghi chú

Việc truy cập DML bao gồm chèn, cập nhật và xóa dữ liệu từ các table

Các câu lệnh insert có một loại truy cập. Nó là một phương pháp đòi hỏi nhiều nỗ lực đối với kiểu dữ liệu. Các câu lệnh insert cấp phát không gian cần thiết cho việc xây dựng varray. Ví dụ, trong một mảng ba phần tử cho `street_address`, có thể chèn một đến ba hàng dữ liệu. Khi chèn hàng, một instance của loại tập hợp được tạo với số hàng được sử dụng.

-- This is in varray_dml1.sql on the publisher's web site.

```

INSERT INTO individuals VALUES
( individuals_s1.nextval, 'John', 'Sidney', 'McCain', 'Mr. ');
INSERT INTO addresses VALUES
( 1

```

```
, individuals_s1.currval
, strings
  ('Office of Senator McCain'
  , '450 West Paseo Redondo'
  , 'Suite 200')
, 'Tucson'
, 'AZ'
, '85701'
, 'USA');
```

Chương trình mẫu chèn một tập hợp đầy đủ gồm ba hàng vào kiểu dữ liệu varray. Điều quan trọng là phải chú ý trong mệnh đề values, tên kiểu dữ liệu varray được sử dụng làm tên constructor (phương thức tạo). Phương thức tạo sử dụng cú pháp được trình bày trước đó với một danh sách các tham số phân cách nhau bằng dấu phẩy cho một tập hợp dấu ngoặc đơn.

Nếu bạn truy vấn cột street_address từ table, bạn thấy một tập hợp trả về của phương thức tạo với những tham số thật sự của nó. Điều này được minh họa bằng cách chạy một query như sau:

-- This is in varray_dml1.sql on the publisher's web site.

```
SELECT street_address
FROM addresses;
```

Kết quả rút ngắn từ query được ghi chú.

-- This is found running varray_dml1.sql from the publisher's web site.

```
STREET_ADDRESS
```

```
ADDRESS_VARRAY('Office of Senator McCain', '450 West Paseo ...
```

Loại kết quả này không hữu dụng cho lắm. Nó cũng rất khác với những gì bạn có thể mong đợi. Sử dụng ngôn ngữ truy vấn dữ liệu (DQL) để chọn một kết quả từ một kiểu dữ liệu varray đòi hỏi cú pháp đặc biệt. Bạn cần định nghĩa một cấu trúc tập hợp nested table để thật sự truy cập dữ liệu varray một cách có ý nghĩa. Nếu bạn không quen thuộc với khái niệm về các nested table, có thể bạn muốn chuyển nhanh đến mục "Nested Tables"

Ví dụ sau đây minh họa cách bạn tạo một tập hợp nested table cho vấn đề đang có. Ở mục sau trong chương điều này sẽ được đề cập thêm chi tiết khi nghiên cứu các nested table. Ở đây, nó minh họa một cú pháp một phần không trực giác để truy vấn dữ liệu.

```
-- This is in varray_dml1.sql on the publisher's web site.
-- Create a PL/SQL table datatype.
CREATE OR REPLACE TYPE varray_nested_table IS TABLE OF VARCHAR2(30
CHAR);
/

-- Use SQL*Plus to format the output.
COL      column_value FORMAT A30
-- Print  formatted elements from aggregate table.
SELECT  nested.column_value
FROM    addresses a
        , TABLE(CAST(a.street_address AS VARRAY_NESTED_TABLE)) nested
WHERE   address_id = 1;
```

Từ khóa **TABLE** có thể được trao đổi với từ khóa **THE** cũ hơn, nhưng Oracle đề nghị bạn sử dụng **TABLE**. Trong chương trình mẫu, một tập hợp **nested table** được tạo để phản ánh định nghĩa phần tử cho **varray**. Các **nested table** không được liên kết hướng lên như các **varray** nhưng có thể được sử dụng để tạm thời chứa nội dung của các **varray**. Sử dụng một **nested table** là cách duy nhất để hiển thị có ý nghĩa các nội dung của **varray** sử dụng câu lệnh **select**. Hàm **CAST** cho phép bạn chuyển đổi **varray** thành một **nested table** mà sau đó có thể được quản lý dưới dạng một bảng gộp (**aggregate table**).

Kết quả được định dạng từ query là

```
-- This is found running varray_dml1.sql from the publisher's web site.
COLUMN_VALUE
```

```
-----
Office of Senator McCain
450 West Paseo Redondo
Suite 200
```

Bạn phải bảo đảm **varray** là một gương của cấu trúc **table** xếp lồng. Nếu chúng không phải là các gương kiểu dữ liệu, bạn sẽ gặp phải một lỗi **ORA-00932**. Lỗi than phiền rằng nguồn cho **CAST** là kiểu sai được chuyển đổi sang một **nested table**.

Bạn cũng có thể cập nhật các cột **varray** và **nested table** như được minh họa trong câu lệnh này:

```
-- This is in varray_dml2.sql on the publisher's web site.
UPDATE  addresses
SET     street_address =
```

```

strings('Office of Senator McCain'
        , '2400 E. Arizona Biltmore Cir.'
        , 'Suite 1150')
WHERE    address_id = 1;

```

Câu lệnh update gán giá trị của một loại tập hợp strings mới được tạo. Sử dụng cùng một câu lệnh select phức tạp để truy vấn dữ liệu mới, bạn sẽ thấy kết quả sau đây:

-- This is found running varray_dml2.sql from the publisher's web site.

```
COLUMN_VALUE
```

```

Office of Senator McCain
2400 E. Arizona Biltmore Cir.
Suite 1150

```

Bạn không thể cập nhật một phần của một cột varray bằng bất kỳ phương thức trực tiếp hoặc gián tiếp trong SQL. Bạn phải cập nhật các phần của các tập hợp varray bằng cách những chương trình PL/SQL. Chương trình khối nặc danh sau đây cho phép cập nhật phần tử đầu tiên của tập hợp varray:

-- This is in varray_dml3.sql on the publisher's web site.

```

DECLARE
    TYPE address_type IS RECORD
    ( address_id      INTEGER
    , individual_id   INTEGER
    , street_address  STRINGS
    , city            VARCHAR2(20 CHAR)
    , state           VARCHAR2(20 CHAR)
    , postal_code     VARCHAR2(20 CHAR)
    , country_code    VARCHAR2(10 CHAR));
    address          ADDRESS_TYPE;

CURSOR get_street_address
    (address_id_in INTEGER) IS
    SELECT *
    FROM    addresses
    WHERE   address_id = address_id_in;

BEGIN

```



```

-- Access the cursor.
OPEN get_street_address(1);
FETCH get_street_address INTO address;
CLOSE get_street_address;
-- Reset the first element of the varray type variable.
address.street_address(1) := 'Office of Senator John McCain';
-- Update the entire varray column value.
UPDATE addresses
SET street_address = address.street_address
WHERE address_id = 1;
END;
/

```

Chương trình mẫu đọc đầy đủ và varray xếp lổng. Sau đó nó cập nhật chỉ phần tử đầu tiên của tập hợp rồi ghi lại tập hợp sang cùng một hàng.

Bạn có thể thấy rằng nó chỉ thay đổi phần tử đầu tiên của cột tập hợp varray. Điều này thực hiện bằng cách sử dụng cú pháp nested table vốn đã được trình bày trong một ví dụ trước. Kết quả nằm trong một file kết quả sau đây:

```

-- This is found running varray_dml3.sql from the publisher's web site.
COLUMN_VALUE
-----
Office of Senator John McCain
2400 E. Arizona Biltmore Cir.
Suite 1150

```

Vẫn còn một tình huống update khác để bạn kiểm tra. Ví dụ này trình bày cách một cột tập hợp varray có thể phát triển từ một phần tử lên thành hai hoặc nhiều phần tử như thế nào. Thêm các phần tử vào một cột tập hợp varray đòi hỏi PL/SQL. Điều này giống như trường hợp cập nhật một phần tử đơn của cột tập hợp varray. Từ phần thảo luận trước, bạn nên nhớ lại rằng một câu lệnh insert xây dựng một cột tập hợp varray.

Câu lệnh insert trong ví dụ này chèn chỉ một phần tử vào cột street_address, khởi tạo chỉ một phần tử trong tập hợp varray cho hàng. Ví dụ sau đây minh họa câu lệnh insert:

```

-- This is in varray_dml4.sql on the publisher's web site.
INSERT INTO individuals VALUES
( individuals_s1.nextval, 'John', 'Sidney', 'McCain', 'Mr.');
```

```

INSERT INTO addresses VALUES
( 2
, individuals_s1.currval
, strings('Office of Senator Kennedy')
, 'Boston'
, 'MA'
, '02203'
, 'USA');

```

Bạn có thể sử dụng giải pháp sau đây để thêm các phần tử thiếu vào cột tập hợp varray:

-- This is in varray_dml4.sql on the publisher's web site.

```

DECLARE
TYPE address_type IS RECORD
( address_id          INTEGER
, individual_id      INTEGER
, street_address     STRINGS
, city               VARCHAR2(20 CHAR)
, state              VARCHAR2(20 CHAR)
, postal_code        VARCHAR2(20 CHAR)
, country_code       VARCHAR2(10 CHAR));
address              ADDRESS_TYPE;
--Define a cursor to return the %ROWTYPE value.
CURSOR get_street_address
(address_id_in INTEGER) IS
SELECT *
FROM      addresses
WHERE     address_id = address_id_in;
BEGIN
-- Access the cursor.
OPEN get_street_address(2);
FETCH get_street_address INTO address;
CLOSE get_street_address;
-- Add elements.
address.street_address.EXTEND(2);

```

```

address.street_address(2) := 'JFK Building';
address.street_address(3) := 'Suite 2400';
-- Update the varray column value.
UPDATE addresses
SET street_address = address.street_address
WHERE address_id = 2;

END;
/

```

Chương trình mẫu đọc hàng đầy đủ và varray xếp lồng. Sau đó nó cập nhật chỉ phần tử thứ hai và phần tử thứ ba của tập hợp xếp lồng.

Bây giờ cột có ba phần tử, sử dụng loại cú pháp nested table. File kết quả sau đây trình bày kết quả:

```
-- This is found running varray_dml4.sql from the publisher's web site.
```

```
COLUMN_VALUE
```

```

Office of Senator Kennedy
JFK Building
Suite 2400

```

Các Nested Table

Giống như các varray, các nested table (bảng xếp lồng) là những cấu trúc một chiều của cơ sở dữ liệu Oracle 11g SQL hoặc PL/SQL. Bạn có thể sử dụng chúng làm các định nghĩa table, record và đối tượng và truy cập chúng trong SQL và PL/SQL. Bạn cũng có thể sử dụng các nested table trong các định nghĩa table, record và đối tượng. Chúng có thể truy cập trong cả SQL và PL/SQL. Không giống các varray, chúng khác với các mảng (array) truyền thống trong những ngôn ngữ lập trình chẳng hạn như Java, C, C++ và C#. Sự khác biệt là chúng không có kích cỡ tối đa ban đầu và do đó kích cỡ của chúng không bị ràng buộc ngoại trừ bộ nhớ có sẵn trong SGA. Hệ quả gần giống với các ngôn ngữ lập trình chuẩn là các bag và set.

Định nghĩa các Nested Table dưới dạng các loại đối tượng là những cấu trúc chương trình PL/SQL

Cú pháp để định nghĩa một loại tập hợp nested table PL/SQL là

```
TUPE type_name IS TABLE OF element_type [NOT NULL];
```

Như được thảo luận, tên kiểu (type name) thường là một kiểu theo sau là một dấu gạch dưới và từ table. Một số nhà lập trình thích hậu tố của tab hơn của table. Bạn chọn cái gì thì không quan trọng. Nhưng quan trọng là bạn phải chọn nó một cách nhất quán.

Chương trình mẫu sau đây minh họa việc định nghĩa, khai báo và khởi tạo một nested table của các card trong một đơn vị chương trình PL/SQL. Các card sẽ được giới hạn chỉ trong một bộ. Chúng sẽ được định nghĩa là các chuỗi có chiều dài khả biến:

```
-- This is in create_nestedtable1.sql on the publisher's web site.
DECLARE
    -- Define a nested table type.
    TYPE card_table IS TABLE OF VARCHAR2(5 CHAR);
    -- Declare a nested table with null values.
    cards CARD_TABLE := card_table(NULL,NULL,NULL);
BEGIN
    -- Print initialized null values.
    dbms_output.put_line('Nested table initialized as null values.');
```

```
    dbms_output.put_line('-----');
```

```
    FOR i IN 1..3 LOOP
        dbms_output.put ('Cards Varray [ ' || i || ' ] ');
        dbms_output.put_line([' ' || cards(i) || '']);
    END LOOP;

    -- Assign values to subscripted members of the nested table.
    cards(1) := 'Ace';
    cards(2) := 'Two';
    cards(3) := 'Three';

    -- Print initialized null values.
    dbms_output.put (CHR(10)); -- Visual line break.
    dbms_output.put_line('Nested table initialized as 11, 12 and 13.');
```

```
    dbms_output.put_line('-----');
```

```
    FOR i IN 1..3 LOOP
        dbms_output.put_line('Cards [ ' || i || ' ] ' || [' ' || cards(i) || '']);
    END LOOP;

END;
/
```

Chương trình mẫu định nghĩa một tập hợp nested table cục bộ, khai báo một biến tập hợp khởi tạo, in các phần tử tập hợp giá trị rỗng, gán những giá trị vào các phần tử và in lại các giá trị phần tử tập hợp. Sau đây là kết quả từ chương trình create_nestedtable1.sql:

```
-- This is found running create_nestedtable1.sql from the publisher's web
site.
```

```
Nested table initialized as nulls.
```

```
Cards Varray [1] []
```

```
Cards Varray [2] []
```

```
Cards Varray [3] []
```

```
Nested table initialized as Ace, Two and Three.
```

```
Cards [1] [Ace]
```

```
Cards [2] [Two]
```

```
Cards [3] [Three]
```

Nếu không khởi tạo tập hợp, bạn đưa ra một ngoại lệ ORA-06531 cho biết một tập hợp không được khởi tạo. Khi bạn khởi tạo một varray, bạn xác lập số hàng được khởi tạo thật sự. Bạn có thể xem phương thức Collection API COUNT để thấy bao nhiêu hàng đã được khởi tạo để bạn không đọc qua số phần tử. Các nested table có chức năng như các varray khi bạn cố truy cập một phần tử trước khi cấp phát cho nó không gian và một giá trị index, và chúng đưa ra một ngoại lệ ORA-06533. Ngoại lệ nghĩa là chỉ số dưới (subscript) không có sẵn bởi vì nó không tồn tại. Khi bạn đã định nghĩa nested table là có kích cỡ ba hàng, bạn xác lập kích cỡ của nó. Do đó, biến có ba chỉ số dưới hợp lệ 1, 2 và 3.

Nếu bạn gặp phải lỗi, bạn có thể tham khảo tài liệu Oracle 11g. Bạn sẽ thấy có phương thức Collection API EXTEND để cấp phát không gian, và nó quá tải. Nó cũng được đề cập trong mục sau “Collection API”.

Như được thảo luận trong mục về varray, việc sử dụng phương thức Collection API EXTEND(n, i) để chèn một hàng ngoài dãy được tạo chỉ số dưới sẽ thất bại. Nó sẽ đưa ra lỗi subscript beyond count.

Bạn thêm một hàng bằng cách sử dụng phương thức Collection API EXTEND mà không có hoặc có một tham số thật sự. Nếu bạn sử dụng tham số, nó là số hàng để khởi tạo. Nó không thể vượt quá hiệu giữa số hàng có thể có và số hàng thật sự cho varray. Để biết thêm thông tin về việc sử dụng những phương thức này, hãy xem mục “Collection API”.

Chương trình sau đây minh họa việc khởi tạo không có các hàng trong phần khai báo. Sau đó, nó minh họa việc khởi tạo và gán động trong phần thực thi.

```
-- This is in create_nestedtable2.sql on the publisher's web site.
DECLARE
    -- Define a nested table.
    TYPE card_suit IS TABLE OF VARCHAR2(5 CHAR);

    -- Declare a no-element collection.
    cards CARD_SUIT := card_suit();
BEGIN
    -- Allocate space as you increment the index.
    FOR i IN 1..3 LOOP
        cards.EXTEND;
        IF i = 1 THEN
            cards(i) := 'Ace';
        ELSIF i = 2 THEN
            cards(i) := 'Two';
        ELSIF i = 3 THEN
            cards(i) := 'Three';
        END IF;
    END LOOP;

    -- Print initialized collection.
    dbms_output.put_line('Nested table initialized as Ace, Two and Three.');
```

```
dbms_output.put_line('-----');
```

```
FOR i IN 1..3 LOOP
    dbms_output.put ('Cards [' || i || ' ']);
    dbms_output.put_line([' ' || cards(i) || ' ']);
END LOOP;

END;

/
```

Chương trình mẫu định nghĩa một tập hợp nested table cục bộ và khai báo một tập hợp không phần tử. Bên trong khối thực thi, chương trình khởi tạo và in các giá trị phần tử.

Kết quả được minh họa ở đây là:

```
-- This is found running create_nestedtable2.sql from the publisher's web
site.
```

```
Nested table initialized as Ace, Two and Three.
```

```
Cards [1] [Ace]
```

```
Cards [2] [Two]
```

```
Cards [3] [Three]
```

Bây giờ bạn có những điểm cơ bản để xây dựng các cấu trúc nested table bên trong những đơn vị chương trình PL/SQL. Sức mạnh và những tiện ích quản lý của các phương thức tập hợp sẽ nâng cao khả năng sử dụng những cấu trúc này. Mục này đã đề cập hơn nữa về các phương thức Collection API đó được sử dụng trong việc thảo luận varray. Chúng giúp minh họa những vấn đề khởi tạo và được đề cập chuyên sâu hơn ở mục sau trong chương. Bằng cách sử dụng những cấu trúc này trong các ví dụ đơn giản, bạn sẽ thấy những cơ hội áp dụng các phương thức qua các loại tập hợp.

Định nghĩa và sử dụng các Nested Table làm các loại đối tượng trong PL/SQL

Cú pháp để định nghĩa một loại tập hợp SQL của các nested table trong cơ sở dữ liệu là

```
CREATE OR REPLACE TYPE type_name
AS TABLE OF element_type [ NOT NULL ];
```

Tên kiểu (type name) thường là một chuỗi theo sau là một dấu gạch dưới và từ table. Như được thảo luận, nhiều nhà lập trình và những người quản lý cấu hình thấy nó là một mẫu hữu dụng để nâng cao khả năng đọc mã. Nó cũng là quy ước được sử dụng trong chương cho các loại cấu trúc và đối tượng PL/SQL.

Kiểu phần tử (element type) có thể là bất kỳ kiểu dữ liệu Oracle 11g SQL, kiểu con (subtype) do người dùng định nghĩa, hoặc loại đối tượng. Cho phép các giá trị rỗng trong các nested table là lựa chọn mặc định. Nếu các giá trị rỗng không được cho phép, nó phải được xác định khi chúng được định nghĩa.

Chương trình mẫu sau đây minh họa việc định nghĩa một loại tập hợp nested table. Bước đầu tiên là tạo một loại tập hợp nested table SQL trong schema:

```
-- This is in create_nestedtable3.sql on the publisher's web site.
```

```
CREATE OR REPLACE TYPE card_table
```

```
AS TABLE OF VARCHAR2(5 CHAR);
```

```
/
```

Sau đó chương trình PL/SQL khởi nặc danh sử dụng nó một cách khai báo và khởi tạo một biến.

-- This is in `create_nestedtable3.sql` on the publisher's web site.

```
DECLARE
```

```
-- Declare a nested table with null values.
```

```
cards CARD_TABLE := card_table(NULL,NULL,NULL);
```

```
BEGIN
```

```
-- Print initialized null values.
```

```
dbms_output.put_line('Nested table initialized as nulls.');
```

```
dbms_output.put_line('-----');
```

```
FOR i IN 1..3 LOOP
```

```
    dbms_output.put ('Cards Varray [ | | i | | ] ');
```

```
    dbms_output.put_line([' | | cards(i) | | ']);
```

```
END LOOP;
```

```
-- Assign values to subscripted members of the table.
```

```
cards(1) := 'Ace';
```

```
cards(2) := 'Two';
```

```
cards(3) := 'Three';
```

```
-- Print initialized values.
```

```
dbms_output.put (CHR(10)); -- Visual line break.
```

```
dbms_output.put_line('Nested table initialized as Ace, Two and Three.');
```

```
dbms_output.put_line('-----');
```

```
FOR i IN 1..3 LOOP
```

```
    dbms_output.put_line('Cards [ | | i | | ] ' || [' | | cards(i) | | ']);
```

```
END LOOP;
```

```
END;
```

```
/
```

Chương trình mẫu khai báo một biến tập hợp được khởi tạo, in các phần tử tập hợp giá trị rỗng, gán những giá trị vào các phần tử và in lại các giá trị phần tử tập hợp. Sau đây là kết quả từ chương trình `create_nestedtable1.sql`:


```
-- This is found running create_nestedtable3.sql from the publisher's web
site.
```

```
Nested table initialized as null values.
```

```
Cards Varray [1] []
```

```
Cards Varray [2] []
```

```
Cards Varray [3] []
```

```
Nested table initialized as Ace, Two and Three.
```

```
Cards [1] [Ace]
```

```
Cards [2] [Two]
```

```
Cards [3] [Three]
```

Ưu điểm của việc định nghĩa loại đối tượng nested table là nó có thể được tham chiếu từ bất kỳ chương trình nào có các quyền sử dụng nó trong khi một cấu trúc loại nested table PL/SQL được giới hạn chỉ trong đơn vị chương trình. Các đơn vị chương trình có thể là các chương trình khối nặc danh như ví dụ hoặc các thủ tục lưu trữ hoặc package trong cơ sở dữ liệu. Chỉ cái sau cho phép tham chiếu bởi những chương trình PL/SQL khác vốn có các quyền đối với package. Xem chương 9 để biết chi tiết về việc tạo và sử dụng các package.

Bây giờ bạn sẽ học cách không cho phép những giá trị rỗng trong các nested table. Tác động chính của việc không cho phép chúng xuất hiện khi khởi tạo chúng. Đây là một sự phản ánh vấn đề mà bạn đã thấy trong các varray trước đó. Ví dụ, nếu bạn đã tái định nghĩa loại đối tượng nested table được sử dụng trong chương trình trước để không cho phép các giá trị rỗng, chương trình sẽ thất bại. Như khi sử dụng varray, bạn đưa ra một ngoại lệ PLS-00567 bởi vì bạn đang cố chuyển một giá trị rỗng đến một cột bị ràng buộc NOT NULL.

Khi bạn sử dụng các nested table làm các bag hoặc set, bạn sẽ định nghĩa các cấu trúc chứa hàng trăm hàng. Một số có thể được định nghĩa động bằng cách đếm các hàng trong một table trước khi được tạo dưới dạng các cấu trúc động.

Khi bạn khởi tạo các nested table chứa 100 phần trăm dữ liệu, việc làm như vậy thì đơn giản bởi vì phương thức tạo (constructor) có thể làm điều đó. Tuy nhiên, khi bạn khởi tạo các nested table chứa ít dữ liệu hơn, việc thêm các hàng sẽ đòi hỏi thêm một số kỹ thuật lập trình. Những điều này hầu như tương đương những gì bạn đã làm việc qua với các varray.

Ví dụ sau đây cấp phát một bộ bài chơi đầy đủ. Để làm như vậy, bạn sẽ làm việc với các varray vốn chứa các tập hợp dữ liệu. Bạn sẽ sử dụng các varray bởi vì vấn đề rõ ràng phù hợp với những mảng có cấu trúc truyền thống. Có mười ba lá bài (card) trong một bộ và có bốn bộ. Bạn sẽ thấy việc sử dụng những cấu trúc này như trong chương trình sau đây cùng với các vòng lặp xếp lồng. Nếu bạn không quen thuộc với những cấu trúc vòng lặp, bạn có thể xem lại chúng trong chương 4.

Bước đầu tiên bao gồm tạo ba loại tập hợp SQL:

```
-- This is in create_nestedtable4.sql on the publisher's web site.
CREATE OR REPLACE TYPE card_unit_varray AS VARRAY(13) OF VARCHAR2(5
CHAR);
/
CREATE OR REPLACE TYPE card_suit_varray AS VARRAY(4) OF VARCHAR2(8
CHAR);
/
CREATE OR REPLACE TYPE card_deck_table AS TABLE OF VARCHAR2(17
CHAR);
/
```

Sau đó bạn có thể sử dụng những loại tập hợp này trong khối nặc danh sau đây:

```
-- This is in create_nestedtable4.sql on the publisher's web site.
DECLARE
    -- Declare counter.
    counter INTEGER := 0;

    -- Declare and initialize a card suit and unit collections.
    suits CARD_SUIT_VARRAY :=
        card_suit_varray('Clubs','Diamonds','Hearts','Spades');
    units CARD_UNIT_VARRAY :=
        card_unit_varray('Ace','Two','Three','Four','Five','Six','Seven'
            ,'Eight','Nine','Ten','Jack','Queen','King');
    -- Declare and initialize a null nested table.
    deck CARD_DECK_TABLE := card_deck_table();
BEGIN
    -- Loop through the four suits, then thirteen cards.
    FOR i IN 1..suits.COUNT LOOP
        FOR j IN 1..units.COUNT LOOP
```

```

        counter := counter + 1;
        deck.EXTEND;
        deck(counter) := units(j) || ' of ' || suits(i);
    END LOOP;
END LOOP;
-- Print initialized values.
dbms_output.put_line('Deck of cards by suit.');
```

```

dbms_output.put_line('-----');
FOR i IN 1..counter LOOP
    dbms_output.put_line([' || deck(i) || ']);
END LOOP;
END;
/
```

Chương trình mẫu xây dựng hai varray mà sẽ được sử dụng để xây dựng một bộ bài. Việc cấp phát không gian động xảy ra cho nested table, trong khi các varray được cấp phát tĩnh.

Kết quả được minh họa ở đây:

```
-- This is found running create_nestedtable4.sql from the publisher's web site.
```

```
Deck of cards by suit.
```

```
-----
```

```
[Ace of Clubs]
```

```
[Two of Clubs]
```

```
[Three of Clubs]
```

```
...
```

```
The remainder is redacted to conserve space.
```

```
...
```

```
[Jack of Spades]
```

```
[Queen of Spades]
```

```
[King of Spades]
```

Bạn đã phát triển những kỹ năng sử dụng những tập hợp nested table làm các loại đối tượng. Mục tiếp theo sẽ sử dụng các tập hợp nested table và ấn định các table nào sử dụng chúng làm các kiểu dữ liệu cột.

Định nghĩa và sử dụng các Nested Table làm các kiểu dữ liệu cột trong các Table

Sau khi tạo một table với một cột kiểu dữ liệu nested table, bạn cần biết cách sử dụng nó như thế nào. Sử dụng nó đòi hỏi việc hiểu những phương thức truy cập DML và chúng làm việc như thế nào với các nested table. Các nested table, như các varray, không đưa ra những điều kiện duy nhất đối với việc xóa, vì việc xóa nằm tại cấp hàng. Tuy nhiên, có những điểm khác biệt đáng kể khi nói về việc sử dụng các câu lệnh insert và update.

Những điểm khác biệt ít hơn trên các varray trên các hoạt động update, Nested table cung cấp một tập hợp truy cập trực giác hơn cho DML. Vì ERD tượng trưng cho street_address dưới dạng một danh sách, không cần phải tái định nghĩa nó. Varray hoặc nested table là phần thực thi của một danh sách.

Trong khi DML có tính trực giác hơn, bạn mất đi một phần linh hoạt trên các ràng buộc cơ sở dữ liệu. Khi làm việc với các varray ràng buộc để không cho phép các giá trị rỗng. Đây đã là một tính năng mới trong Oracle 10g. Bây giờ varray được lưu trữ dưới dạng các cấu trúc inline, cho phép một ràng buộc NOT NULL. Trái lại, các nested table dưới dạng các giá trị cột không cho phép sử dụng một ràng buộc NOT NULL. Điều này đúng khi bạn định nghĩa loại table với lựa chọn mặc định hoặc ghi đè lựa chọn mặc định là không cho phép các giá trị rỗng (null). Khi bạn cố sử dụng một loại table trong một định nghĩa table và xác lập ràng buộc cột sang NOT NULL, nó sẽ đưa ra một lỗi ORA-02331.

Ghi chú

Nếu bạn sử dụng công cụ oerr để kiểm tra một lỗi ORA-02331, nó sẽ cho bạn biết rằng nó áp dụng vào các varray. Điều này không còn đúng nữa.

Bạn có thể dễ dàng test giới hạn trên các ràng buộc cơ sở dữ liệu. Tạo một kiểu dữ liệu nested table như sau:

```
CREATE OR REPLACE TYPE address_table AS TABLE OF VARCHAR2(30 CHAR)
NOT NULL;
```

```
/
```

Bạn sẽ thấy lỗi sau đây được đưa ra nếu bạn chạy script chèn các giá trị rỗng trong cột:

```
,street_address          ADDRESS_TABLE          NOT NULL
*
```

ERROR at line 4:

ORA-02331: cannot create constraint on column of datatype Named Table Type

Việc tạo table thất bại bởi vì kiểu kiểm tra nested table không cho phép sử dụng ràng buộc NOT NULL. Theo định nghĩa, các nested table không thể ràng buộc. Bạn nên xem xét điều này khi sử dụng một nested table. Bạn lưu trữ một table vốn chỉ được tham chiếu qua table bố. Việc đặt một ràng buộc cột NOT NULL không nhất quán với một loại nested table.

Một ràng buộc NOT NULL trên một cột nested table tương đương như việc bắt buộc một hàng được chèn trong nested table trước khi định nghĩa nó. Điều này không thể được. Một ràng buộc NOT NULL trong trường hợp này có chức năng như một ràng buộc tính toán vẹn tham chiếu cơ sở dữ liệu và do đó không được phép. Các ràng buộc NOT NULL cho các nested table trở thành những vấn đề thiết kế ứng dụng cần xem xét khi chèn hoặc cập nhật các hàng.

Sau khi đọc mục này, bạn muốn xem xét tại sao bạn sử dụng một varray trong các định nghĩa table. Bạn sẽ thấy rằng các nested table cung cấp một phương thức truy cập tự nhiên hơn đến các phần tử trong các câu lệnh update DML.

created_addressbook2.sqp xây dựng môi trường cho phần này. Bạn nên chạy nó trước khi cố sử dụng bất kỳ script sau đây.

Như các varray được đề cập trước đó, các câu lệnh insert có một loại truy cập. Nó là một phương pháp đòi hỏi nhiều nỗ lực đối với kiểu dữ liệu. Các câu lệnh insert cấp phát không gian cần thiết để tạo nested table. Ví dụ, trong một phần thực thi nested table của street_address, có thể chèn một vào bất kỳ số hàng dữ liệu. Khi chèn hàng, một instance của loại tập hợp được tạo với số hàng được chọn. Như bạn thấy, cú pháp để chèn nested table là gương (mirror) sử dụng cho một varray. Ngoại lệ là tên của loại tập hợp được sử dụng trong phương thức tạo.

-- This is found in nestedtable_dml1.sql on the publisher's web site.

```
INSERT INTO individuals VALUES
```

```
( individuals_s1.nextval, 'John', 'Sidney', 'McCain', 'Mr.');
```

```
INSERT INTO addresses VALUES
```

```
( addresses_s1.nextval
```

```
, individuals_s1.currval
```

```
, strings
```

```
('Office of Senator McCain'
```

```
, '450 West Paseo Redondo'
```

```
, 'Suite 200')
```

```
, 'Tucson'
```

```

,'AZ'
,'85701'
,'USA');

```

Chương trình mẫu chèn một tập hợp đầy đủ gồm ba hàng vào kiểu dữ liệu nested table. Điều quan trọng cần chú ý là trong mệnh đề values, tên kiểu dữ liệu nested table được sử dụng làm phương thức tạo. Phương thức tạo sử dụng cú pháp được trình bày trước đó với một danh sách các tham số thật sự được phân cách bằng dấu phẩy trong một tập hợp dấu ngoặc đơn.

Nếu bạn truy vấn cột street_address từ table, bạn sẽ thấy một tập hợp trả về của phương thức tạo với những tham số thật sự của nó. Điều này được minh họa bằng cách chạy một query như sau:

```
-- This is found in nestedtable_dml1.sql on the publisher's web site.
```

```
SELECT street_address
FROM addresses;
```

Kết quả rút ngắn từ query được ghi chú.

```
-- This is found running nestedtable_dml1.sql from the publisher's web site.
```

```
STREET_ADDRESS
```

```
ADDRESS_TABLE('Office of Senator McCain', '450 West Paseo ...
```

Loại kết quả này không hữu dụng cho lắm. Nó cũng rất khác với những gì bạn mong đợi. Sử dụng ngôn ngữ truy vấn dữ liệu (DQL) để chọn một kết quả từ kiểu dữ liệu nested table đòi hỏi cú pháp đặc biệt. Thật may thay, không giống như bạn thực thi varray bằng cách gán (cast) vào một nested table, bạn có thể truy cập trực tiếp các nested table trong DQL.

Ví dụ sau đây định dạng kết quả với SQL*Plus. Sau đó, nó chọn các giá trị cột từ nested table mỗi lần một giá trị.

```
-- This is found in nestedtable_dml1.sql on the publisher's web site.
```

```
-- Use SQL*Plus to format the output.
```

```
COL column_value FORMAT A30
```

```
-- Print formatted elements from aggregate table.
```

```
SELECT  nested.column_value
FROM    addresses a
,       TABLE(a.street_address) nested
WHERE   address_id = 1;
```

Từ khóa TABLE chuyển đổi nested table thành một table gộp như một hàm pipelined. Kết quả từ query là

```
-- This is found running nestedtable_dml1.sql from the publisher's web site.
COLUMN_VALUE
```

```
-----
Office of Senator McCain
450 West Paseo Redondo
Suite 200
```

DQL để truy cập các giá trị trong một nested table trả về một tập hợp hàng. Một vấn đề với một tập hợp hàng là trộn tập hợp hàng với một dữ liệu khác trong SQL. Vì những phần tử khác được trả về trong một lựa chọn bình thường sẽ có một biến cố trên mỗi hàng, thể hiện dữ liệu thì khó.

PL/SQL giúp bạn giảm đi những hạn chế. Bạn sẽ xây dựng một hàm để trả về một chuỗi chiều dài khả biến với các ngắt hàng. Nếu bạn cần xem lại chi tiết về việc xây dựng các hàm PL, hãy xem chương 6. Tương tự, bạn nên kiểm tra Collection API ở sau trong chương này để biết chi tiết về phương thức COUNT.

Hàm sau đây lấy kết quả hàng trả về và tạo một chuỗi có chiều dài khả biến. Bạn sẽ thấy nó là một ví dụ hữu dụng, đặc biệt trong trường hợp xây dựng các địa chỉ thư tín.

```
-- This is found in nestedtable_dml1.sql on the publisher's web site.
```

```
CREATE OR REPLACE FUNCTION many_to_one
(street_address_in ADDRESS_TABLE) RETURN VARCHAR2 IS
    retval VARCHAR2(2000) := '';
BEGIN
    -- Read all elements in the nested table, and delimit with a line break.
    FOR i IN 1..street_address_in.COUNT LOOP
        retval := retval || street_address_in(i) || CHR(10);
    END LOOP;
    RETURN retval;
END many_to_one;
```

Hàm lấy một nested table và biến đổi nó thành một chuỗi nhiều dòng. Sử dụng SQL*Plus để định dạng cột, bạn có thể truy vấn chuỗi được định dạng:

```

-- This is found in nestedtable_dml1.sql on the publisher's web site.
-- Use SQL*Plus to format the output.
COL address_label FORMAT A30

-- Print a mailing label.
SELECT      i.first_name || ' '
           ||      i.middle_initial || ' '
           ||      i.last_name || CHR(10)
           ||      many_to_one(a.street_address)
           ||      city || ', '
           ||      state || ' '
           ||      postal_code address_label
FROM        addresses a
,           individuals i
WHERE       a.individual_id = i.individual_id
AND        i.individual_id = 1;

```

Kết quả được định dạng từ query là

```

-- This is found in nestedtable_dml1.sql on the publisher's web site.
ADDRESS_LABEL

```

```

John McCain
Office of Senator McCain
450 West Paseo Redondo
Suite 200
Tucson, AZ 85701

```

Như bạn đã thấy trước đó trong chương, PL/SQL là cách duy nhất để cập nhật các varray trừ phi thay đổi toàn bộ nội dung. Điều này không phải như vậy với các nested table. Một ưu điểm chính của các nested table là bạn có thể cập nhật các phần tử hàng riêng lẻ. Những hoạt động cập nhật có thể được thực hiện trực tiếp trong các câu lệnh update DML.

Bạn sử dụng chương trình mẫu sau đây để thay thế toàn bộ nội dung của kiểu dữ liệu nested table street_address:

```

-- This is found in nestedtable_dml2.sql on the publisher's web site.
UPDATE      addresses
SET         street_address =
           address_table('Office of Senator McCain')

```



```
, '2400 E. Arizona Biltmore Cir.'
, 'Suite 1150')
```

```
WHERE address_id = 1;
```

Câu lệnh update gán giá trị của một loại tập hợp address_table mới được tạo. Nó làm như vậy bằng cách tạo một instance của nested table. Điều này được thực hiện thông qua một tiến trình tạo trong đó các tham số thật sự được chuyển bên trong các tham số và được phân cách bằng các dấu phẩy.

Sử dụng cùng một câu lệnh select phức tạp để truy vấn dữ liệu mới, bạn sẽ thấy kết quả sau đây:

```
-- This is found in nestedtable_dml2.sql on the publisher's web site.
```

```
COLUMN_VALUE
```

```
Office of Senator McCain
2400 E. Arizona Biltmore Cir.
Suite 1150
```

Bạn có thể trực tiếp cập nhật một phần của một cột nested table trong SQL. Hoặc, bạn có thể sử dụng hai phương pháp trong PL/SQL. Đây là một cải tiến so với việc thiếu tính năng cập nhật tự động cho cột varray.

Chương trình sau đây sẽ cập nhật hàng đầu tiên trong nested table street_address. Nó sẽ thêm tên của thượng nghị sĩ vào chuỗi có chiều dài khả biến:

```
--- This is found in nestedtable_dml3.sql on the publisher's web site.
```

```
UPDATE TABLE(SELECT street_address
                  FROM addresses
                  WHERE address_id = 1)
SET column_value = 'Office of Senator John McCain'
WHERE column_value = 'Office of Senator McCain';
```

Kết quả được định dạng từ query là

```
-- This is found running nestedtable_dml3.sql from the publisher's web site.
```

```
COLUMN_VALUE
```

```
Office of Senator John McCain
450 West Paseo Redondo
Suite 200
```

Hoặc, bạn có thể sử dụng PL/SQL để thực hiện việc cập nhật. Hai phương pháp mà bạn có thể chọn trong PL/SQL là

- Cập nhật trực tiếp một hàng trong nested table
- Cập nhật tất cả nội dung hàng cho một cột nested table

Cập nhật tất cả nội dung hàng là một mirror cho phương pháp được sử dụng trước đó cho các varray. Bạn nên kiểm tra ví dụ được trình bày trước đó trong chương cho phương pháp đó. Tiếp theo, bạn sẽ xem cách cập nhật trực tiếp một hàng trong một cột nested table. Ví dụ sử dụng SQL động và các biến liên kết. Cả hai được đề cập trong chương 11.

-- This is found in nestedtable_dml3.sql on the publisher's web site.

DECLARE

-- Define old and new values.

new_value VARCHAR2(30 CHAR) := 'Office of Senator John McCain';

old_value VARCHAR2(30 CHAR) := 'Office of Senator McCain';

-- Build SQL statement to support bind variables.

sql_statement VARCHAR2(100 CHAR)

:= 'UPDATE THE (SELECT street_address '

|| ' FROM addresses '

|| ' WHERE address_id = 21) '

|| 'SET column_value = :1 '

|| 'WHERE column_value = :2';

BEGIN

-- Use dynamic SQL to run the update statement.

EXECUTE IMMEDIATE sql_statement USING new_value, old_value;

END;

/

Chương trình cho bạn sử dụng các biến liên kết thay vì các biến thay thế để tạo một câu lệnh update động. Mệnh đề USING hỗ trợ các chế độ IN, OUT và IN OUT như các hàm và thủ tục được đề cập trong chương trước. Mặc định là chế độ IN gần giống với một câu lệnh UPDATE đối với những gì bạn làm khi bạn muốn chuyển các tham số vào các cursor một cách tường minh.

Ghi chú

Các biến liên kết là các placeholder được đánh số. Các biến hoặc chuỗi riêng biệt theo vị trí tham chiếu chúng với mệnh đề USING.

Kết quả được định dạng từ query giống với kết quả trong ví dụ vừa rồi.

Các hoạt động cập nhật chỉ có thể được thực hiện cho các phần tử bên trong một nested table. Nếu bạn muốn thêm một phần tử vào một giá trị cột nested table, bạn phải sử dụng PL/SQL. Chương trình sau đây trình bày cách thêm hai hàng dữ liệu.

Câu lệnh insert vẫn vậy ngoại trừ định nghĩa kiểu cho câu lệnh mà bạn đã sử dụng trong việc thảo luận cập nhật varray. Nó chèn chỉ một phần tử vào cột street_address khởi tạo chỉ một phần tử của một tập hợp nested table cho hàng. Ví dụ sau đây minh họa câu lệnh insert:

```
-- This is found in nestedtable_dml4.sql on the publisher's web site.
```

```
INSERT INTO individuals VALUES
```

```
( individuals_s1.nextval, 'Edward', 'Moore', 'Kennedy', 'Mr.');
```

```
INSERT INTO addresses VALUES
```

```
( addresses_s1.nextval
```

```
, individuals_s1.currval
```

```
, address_table('Office of Senator Kennedy')
```

```
, 'Boston'
```

```
, 'MA'
```

```
, '02203'
```

```
, 'USA');
```

Bạn có thể sử dụng giải pháp sau đây để thêm các phần tử thiếu vào cột tập hợp nested table. Bạn nên chú ý chỉ có một sự khác biệt giữa một varray và nested table. Sự khác biệt đó là kiểu dữ liệu.

```
-- This is found in nestedtable_dml4.sql on the publisher's web site.
```

```
DECLARE
```

```
TYPE address_type IS RECORD
```

```
( address_id          INTEGER
```

```
, individual_id      INTEGER
```

```
, street_address     ADDRESS_VARRAY
```

```
, city               VARCHAR2(20 CHAR)
```

```
, state              VARCHAR2(20 CHAR)
```

```
, postal_code        VARCHAR2(20 CHAR)
```

```
, country_code       VARCHAR2(10 CHAR));
```

```
address              ADDRESS_TYPE;
```

```
-- Define a cursor to return the %ROWTYPE value.
```

```

CURSOR get_street_address
  (address_id_in INTEGER) IS
  SELECT *
  FROM addresses
  WHERE address_id = address_id_in;
BEGIN
  -- Access the cursor.
  OPEN get_street_address(2);
  FETCH get_street_address INTO address;
  CLOSE get_street_address;

  -- Add elements.
  address.street_address.EXTEND(2);
  address.street_address(2) := 'JFK Building';
  address.street_address(3) := 'Suite 2400';

  -- Update the varray column value.
  UPDATE   addresses
  SET      street_address = address.street_address
  WHERE    address_id = 2;
END;
/

```

Chương trình mẫu đọc đầy đủ hàng và nested table. Sau đó, nó cập nhật chỉ phần tử thứ hai và phần tử thứ ba của tập hợp xếp lồng.

File xuất sau đây minh họa kết quả:

```

-- This is found running nestedtable_dml4.sql from the publisher's web site.
COLUMN_VALUE

```

```

Office of Senator Kennedy
JFK Building
Suite 2400

```

Các Associative Array

Các associative array (mảng kết hợp) là những cấu trúc một chiều của một kiểu dữ liệu Oracle 11g hoặc một record/loại đối tượng do người dùng định nghĩa. Như được thảo luận ở đầu mục, trước đó chúng được gọi

là các table PL/SQL. Mục này tập trung vào các cấu trúc một chiều của mảng các kết hợp.

Các mảng kết hợp không thể được sử dụng trong các table. Chúng có thể được sử dụng chỉ làm các cấu trúc lập trình. Chúng có thể được truy cập chỉ trong PL/SQL. Chúng giống như các loại tập hợp khác và khác với các mảng (array) theo nghĩa truyền thống của những nhu cầu lập trình chẳng hạn như Java, C, C++ và C#. Chúng có họ hàng gần gũi với các list và map. Chúng không có khả năng của các linked list (danh sách liệt kê) nhưng có thể được làm cho hành động theo cách đó thông qua một giao diện lập trình do người dùng định nghĩa.

Điều quan trọng là cần chú ý một số vấn đề chính được trình bày bởi các mảng kết hợp. Những vấn đề này hướng một phương pháp hơi khác sang việc minh họa sử dụng chúng. Các mảng kết hợp

- Không đòi hỏi việc khởi tạo và không có cú pháp phương thức tạo. Chúng không cần cấp phát không gian trước khi gán các giá trị, điều này loại bỏ sử dụng phương thức Collection API EXTEND.
- Có thể được tạo index bằng số và kể cả Oracle 11g. Trong Oracle 11g, chúng cũng có thể sử dụng các chuỗi có chiều dài khá biến duy nhất.
- Có thể sử dụng bất kỳ số nguyên làm giá trị index, nghĩa là bất kỳ số âm, dương hoặc số nguyên zero.
- Được chuyển đổi ngầm định từ các giá trị trả về %ROWTYPE, loại record và loại đối tượng tương đương thành các cấu trúc mảng kết hợp.
- Là chìa khóa để sử dụng câu lệnh FORALL hoặc mệnh đề BULK COLLECT vốn cho phép chỉ hàng loạt các record từ một table cơ sở dữ liệu sang một đơn vị lập trình.
- Đòi hỏi xử lý đặc biệt khi sử dụng một chuỗi ký tự làm một giá trị index trong bất kỳ cơ sở dữ liệu sử dụng các xác lập được toàn cầu hóa, chẳng hạn như các tham số khởi tạo NLS_COMP hoặc NLS_SORT.

Bạn sẽ bắt đầu bằng cách thấy các kỹ thuật định nghĩa mở rộng được cung cấp trong Oracle 11g. Sau đó kiểm tra những công dụng chính của chúng dưới dạng những cấu trúc lập trình PL/SQL.

Định nghĩa và sử dụng các mảng kết hợp làm cấu trúc chương trình PL/SQL

Cú pháp để định nghĩa một mảng kết hợp trong PL/SQL có hai khả năng. Một là

```
CREATE OR REPLACE TYPE type_name AS TABLE OF element_type [ NOT NULL ]
```

```
INDEX BY [ PLS_INTEGER | BINARY_INTEGER | VARCHAR2(size) ];
```

Những vấn đề này xoay quanh việc cho phép hoặc không cho phép các giá trị rỗng trong các nested table áp dụng vào các mảng kết hợp. Theo quy tắc, bạn nên bảo đảm dữ liệu trong một mảng không rỗng. Bạn có thể làm điều đó bằng cách cho phép ràng buộc khi định nghĩa một mảng kết hợp hoặc bạn có thể làm điều đó bằng lập trình. Đó là một quyết định mà bạn sẽ cần đưa ra trên cơ sở từng trường hợp.

Bạn có thể sử dụng số âm, dương hoặc zero làm giá trị index cho các mảng kết hợp. Cả hai kiểu PLS_INTEGER và BINARY_INTEGER là các kiểu không ràng buộc ánh xạ sang các đặc tả gọi trong C/C++, C# và Java trong Oracle 11g.

Bạn có thể sử dụng các chuỗi có chiều dài khả biến lên đến bốn ngàn ký tự dưới dạng các cột trong các table. Kiểu VARCHAR2 hỗ trợ kích cỡ vật lý Unicode để thực thi toàn cầu hóa. Điều này có nghĩa bạn có thể lưu trữ một nửa hoặc một phần ba ký tự phụ thuộc vào phần thực thi Unicode. Hãy tham khảo chéo các kiểu dữ liệu NCHAR, NCLOB và NVARCHAR2 trong chương để biết thêm thông tin về việc quản lý kích cỡ Unicode.

Cú pháp còn lại để định nghĩa một mảng kết hợp là

```
CREATE OR REPLACE TYPE type_name AS TABLE OF element_type [ NOT
NULL ]
INDEX BY key_type;
```

Lựa chọn *key_type* cho phép bạn sử dụng các kiểu dữ liệu VARCHAR2, STRING hoặc LONG ngoài các kiểu dữ liệu PLS_INTEGER và BINARY_INTEGER. Cả VARCHAR2 và STRING đòi hỏi một kích cỡ định nghĩa. Kiểu dữ liệu LONG thì không, bởi vì theo định nghĩa nó là một chuỗi có chiều dài khả biến là 32,760 byte. Bạn nên xem chương 3 để biết chi tiết về các kiểu dữ liệu LONG.

Như được thảo luận, các mảng kết hợp không đòi hỏi khởi tạo và không có một cú pháp phương thức tạo. Đây là một điểm khác biệt đáng kể so với hai loại tập hợp kia: varray và nested table. Nó là một lợi ích đáng kể đối với việc sử dụng các mảng kết hợp trong PL/SQL. Điều này đặc biệt đúng bởi vì cấu trúc cơ bản của các mảng kết hợp với một index số nguyên đã không thay đổi nhiều kể từ khi nó được thực thi trong Oracle 7, phiên bản 7.3.

Nếu bạn cố tạo một mảng kết hợp, bạn sẽ đưa ra một ngoại lệ PLS-00222. Chương trình sau đây cố tạo một mảng kết hợp:

```
-- This is found in create_assocarray1.sql on the publisher's web site.
```

```
DECLARE
```

```
-- Define an associative array.
```

```
TYPE card_table IS TABLE OF VARCHAR2(5 CHAR)
```

```

INDEX BY BINARY_INTEGER;
-- Declare and attempt to construct an associative array.
cards CARD_TABLE := card_table('A','B','C');
BEGIN
  NULL;
END;
/

```

Nó sẽ đưa ra các thông báo lỗi sau đây:

```

-- This is found running create_assocarray1.sql from the publisher's web
site.

```

```

cards CARD_TABLE := card_table('A','B','C');

```

*

```

ERROR at line 8:

```

```

ORA-06550: line 8, column 23:

```

```

PLS-00222: no function with name 'CARD_TABLE' exists in
          this scope

```

```

ORA-06550: line 8, column 9:

```

```

PL/SQL: Item ignored

```

Sự cố xảy ra bởi vì mệnh đề INDEX BY đã tạo một mảng kết hợp chứ không phải một nested table. Trong khi một định nghĩa kiểu nested table định nghĩa một phương thức tạo một cách ngầm định, một mảng kết hợp thì không.

Trong phần thảo luận trước, phương thức tạo đối tượng đã được xem là một hàm. Các loại tập hợp, varray và nested table khác, là các loại đối tượng định nghĩa các hàm khởi tạo một cách ngầm định. Mảng kết hợp là một cấu trúc chứ không phải loại đối tượng. Do đó, nó không có hàm tạo được tạo ngầm định và thất bại khi bạn cố gọi hàm.

Tương tự, bạn không thể định hướng một mảng kết hợp cho đến khi nó chứa các phần tử. Chương trình mẫu sau đây minh họa sự cố:

```

-- This is found in create_assocarray2.sql on the publisher's web site.

```

```

DECLARE

```

```

-- Define an associative array of strings.

```

```

TYPE card_table IS TABLE OF VARCHAR2(5 CHAR)

```

```

INDEX BY BINARY_INTEGER;

```

```

-- Define an associative array variable.

```

```

cards CARD_TABLE;

```

```

BEGIN
    DBMS_OUTPUT.PUT_LINE(cards(1));
END;
/

```

Nó sẽ đưa ra ngoại lệ sau đây, hoàn toàn khác với những ngoại lệ của các loại tập hợp khác. Như được mô tả trước, bạn nhận được một lỗi tập hợp không khởi tạo từ varray và nested table. Các mảng kết hợp đưa ra một ngoại lệ no data found. Lỗi no data found xuất hiện bởi vì các phần tử mảng kết hợp được tạo thông qua việc gán phần tử trực tiếp.

--This is found running create_assocarray2.sql from the publisher's web site.

```

DECLARE
*
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 13

```

Theo quy tắc chung, bạn muốn tránh khả năng xảy ra lỗi này. Chương trình sau đây cung cấp một cơ chế để tránh gặp phải lỗi:

-- This is found in create_assocarray3.sql on the publisher's web site.

```

DECLARE
    -- Define an associative array of strings.
    TYPE card_table IS TABLE OF VARCHAR2(5 CHAR)
        INDEX BY BINARY_INTEGER;
    -- Define an associative array variable.
    cards CARD_TABLE;
BEGIN
    IF cards.COUNT <> 0 THEN
        DBMS_OUTPUT.PUT_LINE(cards(1));
    ELSE
        DBMS_OUTPUT.PUT_LINE('The cards collection is empty.');
    END IF;
END;
/

```


Phương thức Collection API COUNT trả về một giá trị zero bên dưới chỉ hai điện thoại:

- Khi một tập hợp varray hoặc nested table được khởi tạo và không gian không được cấp phát cho các phần tử
- Khi một mảng kết hợp không có các phần tử được gán

Vì điều kiện thứ hai được đáp ứng, chương trình trả về thông báo từ câu lệnh else. Kết quả như sau:

```
-- This is found running create_assocarray3.sql from the publisher's web site.
```

```
The cards collection is empty.
```

Phương thức Collection API EXTEND sẽ không cấp phát không gian cho mảng kết hợp. Chương trình sau đây sẽ minh họa sự nỗ lực:

```
-- This is found in create_assocarray4.sql on the publisher's web site.
```

```
DECLARE
  -- Define an associative array of strings.
  TYPE card_table IS TABLE OF VARCHAR2(5 CHAR)
    INDEX BY BINARY_INTEGER;
  -- Define an associative array variable.
  cards CARD_TABLE;
BEGIN
  IF cards.COUNT <> 0 THEN
    DBMS_OUTPUT.PUT_LINE(cards(1));
  ELSE
    cards.EXTEND;
  END IF;
END;
/
```

Nỗ lực nhằm mở rộng một mảng kết hợp đưa ra một ngoại lệ PLS-00306. Ngoại lệ cho biết rằng bạn gọi nó với số hoặc các loại đối số không đúng. Nó thật sự có nghĩa là component select không thể tìm thấy phương thức gắn vào mảng kết hợp. Phương thức Collection API EXTEND chỉ có thể thao tác trên các varray và nested table.

Khởi tạo các mảng kết hợp

Như được thảo luận, bạn có thể tạo các mảng kết hợp với một index số hoặc một chuỗi có chiều dài khả biến duy nhất. Các index số phải là các số nguyên vốn là các số dương, âm và zero. Các chuỗi có chiều dài khả biến duy nhất có thể là các kiểu dữ liệu VARCHAR2, STRING hoặc LONG.

Bây giờ bạn thấy cách gán các phần tử vào một mảng kết hợp được tạo index bằng số trong ví dụ sau đây:

```
-- This is found in create_assocarray5.sql on the publisher's web site.
DECLARE
    -- Define a varray of twelve strings.
    TYPE months_varray IS VARRAY(12) OF STRING(9 CHAR);

    -- Define an associative array of strings.
    TYPE calendar_table IS TABLE OF VARCHAR2(9 CHAR) INDEX BY
    BINARY_INTEGER;

    -- Declare and construct a varray.
    month MONTHS_VARRAY :=
        months_varray('January', 'February', 'March', 'April', 'May', 'June'
            , 'July', 'August', 'September', 'October', 'November', 'December');

    -- Declare an associative array variable.
    calendar CALENDAR_TABLE;
BEGIN
    -- Check if calendar has no elements, then add months.
    IF calendar.COUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Assignment loop:');
        DBMS_OUTPUT.PUT_LINE('-----');
        FOR i IN month.FIRST..month.LAST LOOP
            calendar(i) := '';
            DBMS_OUTPUT.PUT_LINE('Index [ ' || i || ' ] is [ ' || calendar(i) ||
        ' ]');

            calendar(i) := month(i);
        END LOOP;

    -- Print assigned element values.
    DBMS_OUTPUT.PUT(CHR(10));
    DBMS_OUTPUT.PUT_LINE('Post-assignment loop:');
    DBMS_OUTPUT.PUT_LINE('-----');
    FOR i IN calendar.FIRST..calendar.LAST LOOP
        DBMS_OUTPUT.PUT_LINE('Index [ ' || i || ' ] is [ ' || calendar(i) || ' ]');
```

```

    END LOOP;
  END IF;
END;
/

```

Ví dụ trước minh họa việc di chuyển nội dung của một varray đến một mảng kết hợp. Trong ví dụ này, cả hai cấu trúc có một giá trị index số.

Kết quả của nó in một dòng cho mỗi tháng cho cả hai loại tập hợp. Sau đây là một bản sao rút ngắn của kết quả:

-- This is found running create_assocarray5.sql from the publisher's web site.

Assignment loop:

Index [1] is []

Index [2] is []

...

Index [11] is []

Index [12] is []

Post-assignment loop:

Index [1] is [January]

Index [2] is [February]

...

Index [11] is [November]

Index [12] is [December]

Trong Oracle 11g, nếu bạn quyết định sử dụng một chuỗi có chiều dài khả biến làm giá trị index, tiến trình thay đổi. Vòng lặp FOR đây chuẩn làm việc để gán những giá trị từ varray sang mảng kết hợp. Tuy nhiên, loại vòng lặp FOR đây sẽ không đọc mảng kết hợp. Vấn đề là phép gán bên trong vòng lặp FOR. Bạn phải thay đổi giá trị index dưới dạng

Gán Index số

calendar(i) := ' ';

Gán Index chuỗi

Calendar(month(i)) := ' ';

Biến counter là i trong chương trình trước. Biến counter được định nghĩa là một PLS_INTEGER. Do đó, giá trị index chuỗi có chiều dài khả biến không thể được cast vào một số nguyên bởi vì nó không phải là số nguyên. Do đó, nó đưa ra một lỗi chuyển đổi ORA-06502. Cùng một ví dụ đã làm việc trước đó bởi vì biến counter được cast dưới dạng một VARCHAR2 khi khởi tạo các thành viên (member) và cast trở lại một INTEGER khi đọc mảng kết hợp.

• • • • •
Thủ thuật

Các mảng kết hợp (associative array) không có cú pháp định hướng tương đương với cú pháp cùng tên với chúng trong JavaScript. Bạn không thể xem mảng kết hợp là một cursor bằng cách sử dụng một cấu trúc vòng lặp FOR cursor.

Điều này đưa ra một vấn đề. Một giá trị index không phải số đòi hỏi bạn phải biết nơi bắt đầu và cách tăng lượng. Các phương thức Collection API FIRST và NEXT cung cấp những công cụ. Chi tiết về Collection API được đề cập sau trong chương này nếu bạn muốn biết thêm về những phương thức này bây giờ.

Bạn có thể sử dụng phương pháp được minh họa trong chương trình mẫu sau đây để giải quyết vấn đề. Trong vòng lặp FOR dãy thứ hai, logic để truyền ngang một index chuỗi duy nhất được cung cấp:

-- This is found in create_assocarray6.sql on the publisher's web site.

```
DECLARE
    -- Define variables to traverse a string indexed associative array.
    current VARCHAR2(9 CHAR);
    element INTEGER;

    -- Define required collection datatypes.
    TYPE months_varray IS VARRAY(12) OF STRING(9 CHAR);
    TYPE calendar_table IS TABLE OF VARCHAR2(9 CHAR) INDEX BY
    VARCHAR2(9 CHAR);

    -- Declare a varray.
    month MONTHS_VARRAY :=
        months_varray('January', 'February', 'March', 'April', 'May', 'June'
, 'July', 'August', 'September', 'October', 'November', 'December');
    -- Declare empty associative array.
    calendar CALENDAR_TABLE;
BEGIN
    -- Check if calendar has no elements.
    IF calendar.COUNT = 0 THEN
        -- Print assignment output title.
        DBMS_OUTPUT.PUT_LINE('Assignment loop:');
        DBMS_OUTPUT.PUT_LINE('-----');
```

```

FOR i IN month.FIRST..month.LAST LOOP
    calendar(month(i)) := TO_CHAR(i);
    DBMS_OUTPUT.PUT_LINE('Index [' || month(i) || '] is [' || i || ']');
END LOOP;

-- Print assigned output title.
DBMS_OUTPUT.PUT(CHR(10));
DBMS_OUTPUT.PUT_LINE('Post-assignment loop:');
DBMS_OUTPUT.PUT_LINE('-----');
FOR i IN 1..calendar.COUNT LOOP
    IF i = 1 THEN
        -- Assign the first character index to a variable.
        current := calendar.FIRST;
        -- Use the derived index to find the next index.
        element := calendar(current);
    ELSE
        -- Check if next index value exists.
        IF calendar.NEXT(current) IS NOT NULL THEN
            -- Assign the character index to a variable.
            current := calendar.NEXT(current);
            -- Use the derived index to find the next index.
            element := calendar(current);
        ELSE
            -- Exit loop since last index value is read.
            EXIT;
        END IF;
    END IF;
    -- Print an indexed element from the array.
    DBMS_OUTPUT.PUT_LINE('Index [' || current || '] is [' || element ||
']');
END LOOP;
END IF;
END;
/

```

Ví dụ trước minh họa việc di chuyển nội dung của một varray có index số sang một mảng kết hợp có index chuỗi duy nhất.

Câu lệnh IF kiểm tra xem bộ đếm vòng lặp for đây có bằng 1 hay không. Câu lệnh này tìm record đầu tiên để bắt đầu truyền ngang mảng kết hợp. Bạn sử dụng phương thức Collection API FIRST để trả về giá trị index chuỗi duy nhất đầu tiên. Chương trình gán giá trị index chuỗi duy nhất sang biến current và sau đó nó sử dụng biến current để tìm giá trị dữ liệu rồi gán vào biến element. Vào lúc này, nó thoát câu lệnh if-then-else và in các giá trị như được mô tả sau đó.

Vào bước đi thứ hai qua vòng lặp FOR đây, cuộc kiểm tra câu lệnh IF sẽ thất bại. Sau đó, nó sẽ đi đến câu lệnh ELSE và gặp phải câu lệnh if-then-else được xếp lồng. Câu lệnh IF sử dụng Collection API NEXT để kiểm tra xem có một record khác trong mảng kết hợp hay không. Nếu có một record khác trong mảng kết hợp, nó sẽ sử dụng biến current để tìm giá trị index kế tiếp. Nó gán giá trị để thay thế giá trị trong biến current. Khi không còn có các record nữa, nó thoát.

Nó in các index và giá trị từ mảng kết hợp calendar bằng package DBMS_OUTPUT. Chương trình tạo luồng đầu ra sau đây. Lần nữa, nó đã được chỉnh sửa để bảo toàn khoảng trống:

```
-- This is found running create_assocarray6.sql from the publisher's web
site.
```

```
Assignment loop:
```

```
-----
Index [January] is [1]
Index [February] is [2]
Index [November] is [11]
Index [December] is [12]
```

```
Post-assignment loop:
```

```
-----
Index [April] is [4]
Index [August] is [8]
Index [December] is [12]
Index [February] is [2]
Index [January] is [1]
Index [July] is [7]
Index [June] is [6]
Index [March] is [3]
```

Index [May] is [5]

Index [November] is [11]

Index [October] is [10]

Index [September] is [9]

Bạn có thể thấy trình tự tập hợp của mảng kết hợp khác với cách nó được truyền ngang. Các phương thức Collection API FIRST, NEXT và PRIOR làm việc từ các hash map cho các chuỗi duy nhất. Việc phân loại phụ thuộc vào các tham số cơ sở dữ liệu NLS_COMP và NLS_SORT trong các cơ sở dữ liệu được toàn cầu hóa.

Kết quả của hành vi phân loại này, các giá trị index chuỗi duy nhất đưa ra một số vấn đề thú vị cần xem xét. Nếu bạn cần theo dõi thứ tự gốc, bạn sẽ cần sử dụng một record hoặc loại đối tượng cung cấp một khóa đại diện. Khóa đại diện có thể duy trì thứ tự gốc.

Các toán tử tập hợp

Oracle 11g cung cấp các toán tử tập hợp. Chúng hành động và có chức năng như các toán tử tập hợp SQL trong các câu lệnh select. Sự khác biệt là chúng được sử dụng trong các phép gán giữa những tập hợp của các loại chữ ký tương hợp. Chúng chỉ làm việc với các varray và các nested table bởi vì chúng đòi hỏi các giá trị index số. Bạn phải di trú các mảng kết hợp vào varray hoặc nested table trước khi sử dụng các toán tử tập hợp, và các tập hợp (collection) phải chứa các kiểu dữ liệu SQL vô hướng. Bạn sẽ đưa ra một số sai hoặc các loại lỗi đối số hoặc một ngoại lệ PLS-00306, nếu bạn sử dụng các toán tử tập hợp để so sánh các tập hợp của các loại đối tượng do người dùng định nghĩa. Bảng 7.2 mô tả các toán tử đa tập hợp.

Bảng 7.2 Các toán tử tập hợp cho các tập hợp (Collections)

Toán tử đa tập hợp	Mô tả
CARDINALITY	Toán tử CARDINALITY đếm số các phần tử trong một tập hợp. Nó không cố gắng đếm chỉ các phần tử duy nhất, nhưng bạn có thể kết hợp nó với toán tử SET để đếm những phần tử duy nhất. Nguyên mẫu là: CARDINALITY(collection)
EMPTY	Toán tử EMPTY có chức năng như một toán hạng khi bạn kiểm tra xem một biến có rỗng hoặc không rỗng. Cú pháp so sánh là: variable_name IS [NOT] EMPTY

MEMBER OF

Toán tử **MEMBER OF** cho bạn kiểm tra xem toán hạng trái có phải là một thành viên của tập hợp được sử dụng làm toán hạng phải hay không. Cú pháp so sánh là:

```
variable_name MEMBER OF
collection_name
```

MULTISET EXCEPT

Toán tử **MULTISET EXCEPT** loại bỏ một tập hợp ra khỏi một tập hợp khác. Nó làm việc như toán tử tập hợp **SQL MINUS**. Nguyên mẫu là:

```
collection MULTISET EXCEPT collection
```

MULTISET INTERSECT

Toán tử **MULTISET INTERSECT** lượng giá hai tập hợp và trả về một tập hợp. Tập hợp trả về chứa những phần tử vốn đã được tìm thấy trong cả hai tập hợp gốc. Nó làm việc như toán tử tập hợp **SQL INTERSECT**. Nguyên mẫu là:

```
collection MULTISET INTERSECT collec-
tion
```

MULTISET UNION

Toán tử **MULTISET UNION** lượng giá hai tập hợp và trả về một tập hợp. Tập hợp trả về chứa tất cả phần tử của cả hai tập hợp. Nơi các phần tử trùng lặp được tìm thấy, chúng được trả về. Nó có chức năng như toán tử tập hợp **SQL UNION ALL**.

Bạn có thể sử dụng toán tử **DISTINCT** để loại bỏ các phần tử trùng lặp. Toán tử **DISTINCT** tuân theo quy tắc toán tử **MULTISET UNION**. Nó có chức năng như toán tử **SQL UNION**. Nguyên mẫu là:

```
collection MULTISET UNION collection
```

SET

Toán tử **SET** loại bỏ các phần tử trùng lặp ra khỏi một tập hợp, và do đó tạo một tập hợp giá trị duy nhất. Nó hoạt động như một toán tử **DISTINCT** phân loại ra các phần tử trùng lặp trong một câu lệnh **SQL**. Nguyên mẫu sử dụng là:

```
SET(collection)
```

Bạn cũng có thể sử dụng toán tử **SET** làm một toán hạng khi bạn kiểm tra xem một


```

ELSE
    retval := '(' || set_in(i);
END IF;
ELSIF i <> set_in.LAST THEN
    retval := retval || ', ' || set_in(i);
ELSE
    retval := retval || ', ' || set_in(i) || ')';
END IF;
END LOOP;
END IF;
RETURN retval;
END format_list;
/

```

Hàm `format_list` chỉ làm việc với các index số bởi vì các toán tử tập hợp giới hạn chỉ trong các varray và nested table vốn chỉ được tạo index bằng các số nguyên. Các ví dụ toán tử tập hợp đều sử dụng hàm này để định dạng kết quả.

Toán tử **CARDINALITY**

Toán tử **CARDINALITY** cho phép đếm các phần tử trong một tập hợp. Nếu có các phần tử duy nhất, chúng được đếm một lần cho mỗi bản sao trong tập hợp. Ví dụ sau đây trình bày cách loại trừ các phần tử tương hợp:

```

DECLARE
    a LIST := list(1,2,3,3,4,4);
BEGIN
    dbms_output.put_line(CARDINALITY(a));
END;
/

```

Chương trình in số 6 bởi vì có bốn phần tử trong tập hợp. Bạn có thể đếm chỉ những giá trị duy nhất bằng cách kết hợp các wizard **CARDINALITY** và **SET** như được trình bày ở đây:

```

DECLARE
    a LIST := list(1,2,3,3,4,4);
BEGIN
    dbms_output.put_line(CARDINALITY(SET(a)));
END;

```

Bây giờ chương trình in số 4 bởi vì có bốn phần tử duy nhất trong tập hợp được dẫn xuất từ tập hợp phần tử thứ sáu. Phần này đã minh họa cách bạn có thể sử dụng toán tử CARDINALITY để đếm các phần tử hoặc tập hợp.

Toán tử EMPTY

Toán tử EMPTY được đề cập trong mục SET.

Toán tử MEMBER OF

Toán tử MEMBER OF cho bạn tìm xem toán hạng trái có phải là một thành viên của tập hợp được sử dụng làm toán hạng phải hay không. Như với những toán tử tập hợp khác, các tập hợp phải sử dụng những kiểu dữ liệu vô hướng chuẩn. Ví dụ này minh họa cách bạn tìm theo một phần tử có hiện hữu trong một tập hợp hay không:

```

DECLARE
    TYPE list IS TABLE OF VARCHAR2(10);
    n VARCHAR2(10) := 'One';
    a LIST := list('One','Two','Three');
BEGIN
    IF n MEMBER OF a THEN
        dbms_output.put_line("n" is member.);
    END IF;
END;
/

```

Toán tử MEMBER OF so sánh và trả về một kiểu Boolean true khi nó tìm thấy giá trị toán hạng trái trong tập hợp toán hạng phải. Kiểu dữ liệu toán hạng trái phải khớp với kiểu dữ liệu cơ sở của tập hợp vô hướng.

Toán tử MULTISSET INTERSECT

Toán tử MULTISSET INTERSECT cho phép tìm những phần tử còn lại từ tập hợp đầu tiên sau khi loại bỏ bất kỳ phần tử tương hợp ra khỏi tập hợp thứ hai. Toán tử bỏ qua bất kỳ phần tử trong tập hợp thứ hai không được tìm thấy trong tập hợp thứ nhất. Ví dụ sau đây trình bày cách loại trừ các phần tử tương hợp:

```

DECLARE
    a LIST := list(1,2,3,4);
    b LIST := list(4,5,6,7);
BEGIN

```

```

dbms_output.put_line(format_list(a MULTISET EXCEPT b));
END;
/

```

Chỉ phần tử 4 hiện hữu trong cả hai tập hợp. Do đó phép toán loại bỏ 4 ra khỏi tập hợp thứ nhất. Kết quả sau đây được tạo ra bởi khối:

```
(1, 2, 3)
```

Phần này đã trình bày cách bạn có thể sử dụng các toán tử tập hợp (set operator) để loại trừ những phần tử ra khỏi một tập hợp khi chúng nằm trong một tập hợp khác.

Toán tử MULTISET INTERSECT

Toán tử MULTISET UNION cho bạn tìm giao hoặc những giá trị tương hợp giữa hai tập hợp. Ví dụ sau đây trình bày cách tạo một tập hợp của giao giữa hai tập hợp.

```

DECLARE
  a LIST := list(1,2,3,4);
  b LIST := list(4,5,6,7);
BEGIN
  dbms_output.put_line(format_list(a MULTISET INTERSECT b));
END;
/

```

Chỉ một phần tử từ cả hai tập hợp tương hợp và đó là số 4. Kết quả sau đây được tạo ra bởi khối:

```
(1, 2, 3)
```

Phần này đã trình bày cách bạn sử dụng các toán tử tập hợp để tạo các tập hợp của giao giữa hai tập hợp.

Toán tử MULTISET UNION

Toán tử MULTISET UNION thực thi một phép toán UNION ALL trên cả hai tập hợp. Ví dụ sau đây trình bày cách kết hợp các tập hợp thành một tập hợp:

```

DECLARE
  a LIST := list(1,2,3,4);
  b LIST := list(4,5,6,7);
BEGIN

```

```

dbms_output.put_line(format_list(a MULTISET UNION b));
END;
/

```

Kết quả phép toán của **MULTISET UNION** được chuyển dưới dạng một tham số thật sự đến hàm `format_list`. Hàm chuyển đổi nó thành chuỗi

```
(1, 2, 3, 4, 4, 5, 6, 7)
```

Bạn sẽ thấy cả hai tập hợp chứa số nguyên 4 và tập hợp vừa tạo ra có hai bản sao của nó. Bạn có thể loại bỏ việc sao chép và mô phỏng một toán tử **UNION** bằng cách thêm wizard **DISTINCT**:

```

DECLARE
  a LIST := list(1,2,3,4);
  b LIST := list(4,5,6,7);
BEGIN
  dbms_output.put_line(format_list(a MULTISET UNION DISTINCT b));
END;
/

```

Hoặc, bạn có thể lấy kết quả của phép toán **MULTISET UNION DISTINCT** và chuyển nó dưới dạng một đối số đến toán tử **SET** để loại bỏ các bản sao.

```

DECLARE
  a LIST := list(1,2,3,4);
  b LIST := list(4,5,6,7);
BEGIN
  dbms_output.put_line(format_list(SET(a MULTISET UNION b)));
END;
/

```

Cả hai toán tử **DISTINCT** và **SET** tạo kết quả sau đây:

```
(1, 2, 3, 4, 5, 6, 7)
```

Phần này đã trình bày cách bạn có thể sử dụng các phép toán tập hợp với tập hợp để tạo các tập hợp bổ của hai tập hợp có hoặc không có các giá trị bản sao.

Toán tử SET

Toán tử SET hành động trên một đầu ra vốn là một tập hợp khác, nó loại bỏ bất kỳ bản sao ra khỏi tập hợp và trả về một tập hợp mới có những giá trị duy nhất. Ví dụ sau đây minh họa cách xén một tập hợp thành các phần tử duy nhất:

```
DECLARE
  a LIST := list(1,2,3,3,4,4,5,6,6,7);
BEGIN
  dbms_output.put_line(format_list(SET(a)));
END;
/
```

Tập hợp gốc chứa mười phần tử nhưng ba phần tử được sao chép. Toán tử SET loại bỏ tất cả phần tử bản sao và tạo một tập hợp mới có bảy phần tử duy nhất.

(1, 2, 3, 4, 5, 6, 7)

Bạn cũng có thể sử dụng SET làm một toán hạng trong các câu lệnh so sánh:

```
DECLARE
  a LIST := list(1,2,3,4);
  b LIST := list(1,2,3,3,4,4);
  c LIST := list();
FUNCTION isset (set_in LIST) RETURN VARCHAR2 IS
BEGIN
  IF set_in IS A SET THEN
  IF set_in IS NOT EMPTY THEN
    RETURN 'Yes - a unique collection.';
  ELSE
    RETURN 'Yes - an empty collection.';
  END IF;
ELSE
  RETURN 'No - a non-unique collection.';
END IF;
END isset;
```

```
BEGIN
    dbms_output.put_line(isset(a));
    dbms_output.put_line(isset(b));
    dbms_output.put_line(isset(c));
END;
/
```

Ghi chú

Hãy ghi nhớ sử dụng các dấu ngoặc đơn trong khi bạn xây dựng các tập hợp rỗng. Nếu bạn quên các dấu ngoặc đơn bởi vì bạn không cần chúng gọi một số hàm hoặc thủ tục, bạn sẽ đưa ra một lỗi: **ORA-00330 - invalid use of type name (sử dụng tên kiểu không hợp lệ)**.

Chương trình trả về

Yes - a unique collection.

No - a non-unique collection.

Yes - an empty collection.

Khối nặc danh này cho thấy phép so sánh **IS A SET** trả về true khi tập hợp duy nhất hoặc rỗng. Bạn phải sử dụng phép so sánh **IS EMPTY** để bắt giữ các tập hợp rỗng như đã làm trong hàm `format_set` được minh họa trước đó.

Phần này đã trình bày cách bạn có thể sử dụng các toán tử tập hợp để tạo những tập hợp của giao giữa hai tập hợp.

Toán tử **SUBMULTISET**

Toán tử **SUBMULTISET** so sánh toán hạng trái với toán hạng phải để quyết định xem toán hạng trái có phải là một tập hợp con của toán hạng phải hay không. Nó trả về một giá trị Boolean true khi nó tìm thấy tất cả phần tử trong tập hợp trái cũng nằm trong tập hợp phải.

Ví dụ sau đây minh họa cách quyết định xem một tập hợp có phải là một tập hợp con của một tập hợp khác hay không:

```
DECLARE
    a LIST := list(1,2,3,4);
    b LIST := list(1,2,3,3,4,5);
    c LIST := list(1,2,3,3,4,4);
BEGIN
    IF a SUBMULTISET c THEN
        dbms_output.put_line('[a] is a subset of [c]');
```

```

END IF;
  IF NOT b SUBMULTISET c THEN
    dbms_output.put_line('[b] is not a subset of [c]');
  END IF;
END;
/

```

Nó in

```

[a] is a subset of [c]
[b] is not a subset of [c]

```

Điều này cho thấy tất cả phần tử của một tập hợp a nằm trong tập hợp c và tất cả phần tử nằm trong tập hợp b thì không. Bạn nên chú ý rằng hàm này tìm các tập hợp con chứ không phải các tập hợp con thật sự. Một tập hợp con thật sự khác biệt bởi vì tối thiểu nó chứa ít hơn tập hợp một phần tử.

Ghi chú

Các toán tử tập hợp chỉ làm việc khi các tập hợp là những danh sách các biến và hướng. Chúng trả về một ngoại lệ PLS-00306 khi bạn cố sử dụng một loại đối tượng do người dùng định nghĩa.

Collection API

Oracle 8i đã giới thiệu Collection API. Collection API được cung cấp nhằm đơn giản hóa việc truy cập đến các tập hợp (collections). Những phương pháp này để đơn giản hóa sự truy cập trước Oracle 11g. Thật không may, nắm vững chúng thì không quan trọng. Sự dịch chuyển từ các table index-by Oracle 9i sang associative array Oracle 11g làm cho việc hiểu chúng là điều quan trọng. Bạn đã biết lý do làm việc với các associative array (mảng kết hợp). Những phương thức FIRST, LAST, NEXT và PRIOR là cách duy nhất để định hướng các index chuỗi duy nhất.

Những phương thức Collection API thật sự không phải là những phương thức theo một nghĩa hướng đối tượng thật sự. Chúng là các hàm và thủ tục. EXTEND, TRIM và DELETE là những thủ tục. Các phương thức còn lại là các hàm.

Bảng 7.3 tóm tắt Oracle 11g Collection API.

Bảng 7.3 Oracle 11g Collection API

Phương thức	Mô tả
COUNT	Phương thức COUNT trả về số phần tử có không gian cấp phát trong các kiểu dữ liệu

VARRAY và NESTED TABLE. Phương thức COUNT trả về tất cả phần tử trong các mảng kết hợp. Giá trị trả về của phương thức COUNT có thể nhỏ hơn giá trị trả về của LIMIT cho những kiểu dữ liệu VARRAY. Nó có nguyên mẫu sau đây:

pls_integer COUNT

DELETE

Phương thức DELETE cho bạn xóa những thành viên ra khỏi tập hợp. Nó có hai tham số hình thức; một là bắt buộc và cái kia thì tùy chọn. Cả hai tham số chấp nhận các kiểu biến PLS_INTEGER, VARCHAR2 và LONG. Chỉ một tham số thật sự, n được hiểu là giá trị index để xóa ra khỏi tập hợp. Khi bạn cung cấp hai tham số thật sự, hàm xóa mọi thứ ra khỏi tham số n đến m. Nó có những nguyên mẫu sau đây:

void DELETE (n)

void DELETE (n, m)

EXISTS

Phương thức EXISTS kiểm tra để tìm một phần tử có index được cung cấp trong một tập hợp. Nó trả về true khi phần tử được tìm thấy nếu không nó trả về false. Phần tử có thể chứa một giá trị hoặc một giá trị rỗng. Nó có một tham số bắt buộc, và tham số có thể là một kiểu PLS_INTEGER, VARCHAR2 hoặc LONG. Nó có nguyên mẫu sau đây:

boolean EXISTS (n)

EXTEND

Phương thức EXTEND cấp phát không gian cho một hoặc nhiều phần tử mới trong một tập hợp VARRAY hoặc NESTED TABLE. Nó có hai tham số tùy chọn. Nó thêm không gian cho một phần tử theo mặc định mà không có bất kỳ tham số thật sự. Một toán tử tùy chọn chỉ định bao nhiêu không gian vật lý sẽ được cấp phát nhưng nó bị ràng buộc bởi giá trị LIMIT cho các kiểu dữ liệu VARRAY. Khi hai tham số tùy chọn được cung cấp, tham số thứ nhất chỉ định bao nhiêu phần tử nên được cấp phát không gian và tham số thứ hai chỉ định index mà nó sẽ sử dụng để sao chép giá trị sang không gian mới được

	cấp phát. Nó có những nguyên mẫu sau đây:
	void EXTEND
	void EXTEND(<i>n</i>)
	void EXTEND(<i>n</i> , <i>i</i>)
FIRST	Phương thức FIRST trả về giá trị subscript (chỉ số dưới) thấp nhất trong một tập hợp. Nó có thể trả về một kiểu PLS_INTEGER, VARCHAR2 hoặc LONG. Nó có nguyên mẫu sau đây:
	mixed FIRST
LAST	Phương thức LAST trả về giá trị chỉ số dưới cao nhất trong một tập hợp. Nó có thể trả về một kiểu PLS_INTEGER, VARCHAR2 hoặc LONG. Nó có nguyên mẫu sau đây:
	mixed LAST
LIMIT	Phương thức LIMIT trả về giá trị chỉ số dưới cao nhất có thể có trong một tập hợp. Nó chỉ có thể trả về một kiểu PLS_INTEGER và chỉ có thể trả về một kiểu PLS_INTEGER và chỉ có thể được sử dụng bởi một kiểu dữ liệu VARRAY. Nó có nguyên mẫu sau đây:
	mixed LIMIT
NEXT (<i>n</i>)	Phương thức NEXT trả về giá trị chỉ số dưới cao nhất kế tiếp trong một tập hợp khi thành công hoặc một giá trị false. Giá trị trả về là một kiểu PLS_INTEGER, VARCHAR2 hoặc LONG. Nó đòi hỏi một giá trị index hợp lệ dưới dạng một tham số thật sự. Nó có nguyên mẫu sau đây:
	mixed NEXT (<i>n</i>)
PRIOR (<i>n</i>)	Phương thức PRIOR trả về giá trị chỉ số dưới thấp hơn kế tiếp trong một tập hợp khi thành công hoặc một giá trị false. Giá trị trả về là một kiểu PLS_INTEGER, VARCHAR2 hoặc LONG. Nó đòi hỏi một giá trị index hợp lệ dưới dạng một tham số thật sự. Nó có nguyên mẫu sau đây:
	mixed PRIOR (<i>n</i>)

TRIM

Phương thức TRIM loại bỏ một giá trị có chỉ số dưới ra khỏi một tập hợp. Nó có một tham số tùy chọn. Nếu không có một tham số thật sự, nó loại bỏ phần tử cao nhất ra khỏi mảng. Một tham số thật sự được hiểu là tham số bị loại bỏ ra khỏi cuối tập hợp. Nó có các nguyên mẫu sau đây:

void TRIM

void TRIM (n)

Bạn sẽ kiểm tra từng phương thức trong thứ tự bảng chữ cái. Một số ví dụ bao gồm nhiều phương thức Collection API. Như trong các loại tập hợp được đề cập, khó xử lý tách biệt các phương thức Collection API. Nơi một ví dụ đề cập đầy đủ nhiều phương thức, nó sẽ được tham chiếu chéo. Đôi khi nó có thể được tham chiếu trước. Bên dưới mỗi phương thức Collection API, bạn sẽ được chuyển đến mã mẫu thích hợp. Bạn sẽ kiểm tra từng phương thức Collection API trong các chương trình mẫu. Nên chú ý rằng chỉ phương thức EXISTS sẽ không đưa ra một ngoại lệ nếu tập hợp rỗng.

Có năm ngoại lệ tập hợp chuẩn được mô tả trong bảng 7.4.

Bảng 7.4 Các ngoại lệ tập hợp

Ngoại lệ tập hợp	Được đưa ra bởi
COLLECTION_IS_NULL	Một nỗ lực nhằm sử dụng một tập hợp rỗng.
NO_DATA_FOUND	Một nỗ lực nhằm sử dụng một chỉ số dưới (subscript) vốn đã bị xóa hoặc nó là một giá trị index chuỗi duy nhất không tồn tại trong một mảng kết hợp.
SUBSCRIPT_BEYOND_COUNT	Một nỗ lực nhằm sử dụng một giá trị index số cao hơn giá trị số tối đa hiện hành. Lỗi này áp dụng chỉ vào các varray và nested table. Các mảng kết hợp (associative array) không được liên kết bởi giá trị trả về COUNT khi thêm các phần tử mới.
SUBSCRIPT_OUTSIDE_LIMIT	Một nỗ lực nhằm sử dụng một giá trị index số bên ngoài giá trị trả về LIMIT. Lỗi này chỉ áp dụng vào các varray và nested tables. Giá trị LIMIT được định nghĩa một trong hai cách. Varray xác lập kích cỡ tối đa vốn trở thành giá trị giới

hạn của chúng. Các nested table và associative array không có kích cỡ tối đa cố định, do đó giá trị giới hạn được xác lập bởi không gian được cấp phát bởi phương thức EXTEND.

VALUE_ERROR

Một nỗ lực nhằm sử dụng một kiểu vốn không được chuyển đổi thành một PLS_INTEGER vốn là kiểu dữ liệu cho các chỉ số dưới dạng số.

Phương thức COUNT

Phương thức COUNT thật sự là một hàm. Nó không có danh sách tham số hình thức. Nó trả về số phần tử trong mảng. Chương trình mẫu sau đây minh họa rằng nó trả về một giá trị PLS_INTEGER:

```
DECLARE
    TYPE number_table IS TABLE OF INTEGER;
    number_list NUMBER_TABLE := number_table(1,2,3,4,5);
BEGIN
    DBMS_OUTPUT.PUT_LINE('How many elements? {' || number_list.COUNT
    || '}');
END;
/
```

Chương trình mẫu định nghĩa một tập hợp vô hướng cục bộ, khai báo một biến tập hợp và sử dụng hàm COUNT để tìm bao nhiêu phần tử nằm trong tập hợp. Nó tạo kết quả sau đây:

```
How many elements? [5]
```

Phương thức DELETE

Phương thức DELETE là một thủ tục. Nó là một thủ tục quá tải. Nếu khái niệm về quá tải (overload) mới đối với bạn, hãy xem chương 9.

Nó có một phiên bản lấy một tham số hình thức. Tham số phải là một giá trị chỉ số dưới hợp lệ trong tập hợp. Phiên bản này sẽ loại bỏ phần tử có chỉ số đó. Nó được minh họa trong chương trình mẫu của phương thức EXISTS.

Phiên bản còn lại lấy hai tham số hình thức. Cả hai tham số phải là các giá trị chỉ số dưới hợp lệ trong tập hợp. Phiên bản này xóa một dãy phần tử bao hàm gần kề ra khỏi một tập hợp. Chương trình mẫu sau đây minh họa việc xóa dãy ra khỏi một tập hợp:

```
DECLARE
    TYPE number_table IS TABLE OF INTEGER;
```

```

number_list NUMBER_TABLE;
-- Define local procedure to check and print elements.
PROCEDURE print_list(list_in NUMBER_TABLE) IS
BEGIN
    -- Check whether subscripted elements are there.
    DBMS_OUTPUT.PUT_LINE('-----');
    FOR i IN list_in.FIRST..list_in.LAST LOOP
        IF list_in.EXISTS(i) THEN
            DBMS_OUTPUT.PUT_LINE('List [' || list_in(i) || ']');
        END IF;
    END LOOP;
END print_list;
BEGIN
    -- Construct collection when one doesn't exist.
    IF NOT number_list.EXISTS(1) THEN
        number_list := number_table(1,2,3,4,5);
    END IF;
    -- Print initialized contents.
    DBMS_OUTPUT.PUT_LINE('Nested table before a deletion');
    print_list(number_list);
    -- Delete a elements from 2, 3 and 4.
    number_list.DELETE(2,4);
    -- Print revised contents.
    DBMS_OUTPUT.PUT_LINE(CHR(10) || 'Nested table after a deletion');
    print_list(number_list);
END;
/

```

Chương trình mẫu định nghĩa một tập hợp vô hướng cục bộ, định nghĩa một biến tập hợp không được khởi tạo, khởi tạo biến tập hợp và xóa ba phần tử ra khỏi giữa tập hợp. Phần hiển thị của chương trình sử dụng một thủ tục cục bộ để in nội dung hiện hành của một tập hợp.

••••• Thủ thuật

Thủ tục DBMS_OUTPUT.PUT_LINE không thể in một ký tự xuống dòng (line return) nếu bạn chuyển cho nó một chuỗi rỗng. Bạn gửi một CHR (10) hoặc line feed khi bạn muốn in một ngắt dòng (line break) trong file kết hợp.

Nó tạo kết quả sau đây:

Nested table before a deletion

List [1]

List [2]

List [3]

List [4]

List [5]

Nested table after a deletion

List [1]

List [5]

Phương thức EXISTS

Phương thức EXISTS thật sự là một hàm. Nó chỉ có một danh sách tham số hình thức mà nó hỗ trợ. Nó lấy một giá trị chỉ số dưới. Chỉ số dưới có thể là một số hoặc một chuỗi duy nhất. Index chỉ số sau chỉ áp dụng vào các mảng kết hợp Oracle 11g.

Như được đề cập, EXISTS là phương thức Collection API duy nhất không đưa ra ngoại lệ COLLECTION_IS_NULL cho tập hợp phần tử rỗng. Các tập hợp phần tử rỗng có hai loại: thứ nhất, các varray và nested table được tạo bằng một null constructor và thứ hai, các mảng kết hợp không có các phần tử được khởi tạo.

Chương trình sau đây minh họa phương thức EXISTS. Một phần của chương trình được điều chỉnh bởi vì nó được sử dụng trong một chương trình mẫu trước.

```
DECLARE
```

```
    TYPE number_table IS TABLE OF INTEGER;
```

```
    number_list NUMBER_TABLE;
```

```
-- Define local procedure to check and print elements.
```

```
    PROCEDURE print_list(list_in NUMBER_TABLE) IS
```

```
BEGIN
```

```

-- Check whether subscripted elements are there.
  DBMS_OUTPUT.PUT_LINE('-----');
  FOR i IN list_in.FIRST..list_in.LAST LOOP
    IF list_in.EXISTS(i) THEN
      DBMS_OUTPUT.PUT_LINE('List [' || list_in(i) || ']');
    END IF;
  END LOOP;
END print_list;
BEGIN
  -- Construct collection when one doesn't exist.
  IF NOT number_list.EXISTS(1) THEN
    number_list := number_table(1,2,3,4,5);
  END IF;
  -- Print initialized contents.
  DBMS_OUTPUT.PUT_LINE('Nested table before a deletion');
  print_list(number_list);
  -- Delete element 2.
  number_list.DELETE(2);
  --Print revised contents.
  DBMS_OUTPUT.PUT_LINE(CHR(10) || 'Nested table after a deletion');
  print_list(number_list);
END;
/

```

Chương trình mẫu định nghĩa một tập hợp vô hướng cục bộ, định nghĩa một biến tập hợp không được khởi tạo, khởi tạo biến tập hợp và xóa phần tử thứ hai ra khỏi tập hợp. Phần hiển thị của chương trình sử dụng một thủ tục cục bộ để in nội dung hiện hành của một tập hợp. Quan trọng nhất, phương thức EXISTS kiểm tra xem một phần tử có tồn tại mà không đưa ra một ngoại lệ hay không.

Nó tạo kết quả sau đây:

```
Nested table before a deletion
```

```
List [1]
```

```
List [2]
```

```
List [3]
List [4]
List [5]
Nested table after a deletion
```

```
List [1]
List [3]
List [4]
List [5]
```

Phương thức EXTEND

Phương thức EXTEND thật sự là một thủ tục. Nó là một thủ tục quá tải. Nếu khái niệm về quá tải (overload) mới mẻ đối với bạn, hãy xem chương 9 về các package hoặc chương 14 về các đối tượng.

Nó có một phiên bản không đòi hỏi các tham số hình thức. Khi được sử dụng không có các tham số hình thức, EXTEND cấp phát không gian cho một phần tử mới trong một tập hợp. Tuy nhiên, nếu bạn cố gắng EXTEND không gian vượt ra khỏi một LIMIT trong một varray, nó sẽ đưa ra một ngoại lệ.

Một phiên bản thứ hai đòi hỏi một tham số hình thức. Tham số phải là một giá trị số nguyên hợp lệ. EXTEND với một tham số thật sự sẽ cấp phát không gian cho số phần tử đó được xác định bởi tham số thật sự. Như với phiên bản không có một tham số, việc có EXTEND không gian vượt ra khỏi một LIMIT trong một varray sẽ đưa ra một ngoại lệ. Phương thức này được minh họa trong ví dụ sau đây.

Phiên bản cuối cùng đòi hỏi hai tham số hình thức. Cả hai tham số phải là những số nguyên hợp lệ. Tham số thứ hai cũng phải là một giá trị chỉ số dưới hợp lệ trong tập hợp. Phiên bản này cấp phát không gian phần tử bằng với tham số thật sự thứ nhất. Sau đó, nó sao chép nội dung của chỉ số dưới được tham chiếu được tìm thấy trong tham số thật sự thứ hai.

Chương trình sau đây minh họa phương thức EXTEND với một và hai tham số hình thức. Một phần của chương trình được chỉnh sửa đã được sử dụng trong một chương trình mẫu trước.

```
DECLARE
    TYPE number_table IS TABLE OF INTEGER;
    number_list NUMBER_TABLE;

    -- Define local procedure to check and print elements.
    PROCEDURE print_list(list_in NUMBER_TABLE) IS
```



```

BEGIN
    -- Check whether subscripted elements are there.
    DBMS_OUTPUT.PUT_LINE('-----');
    FOR i IN list_in.FIRST..list_in.LAST LOOP
        IF list_in.EXISTS(i) THEN
            DBMS_OUTPUT.PUT_LINE('List [' || list_in(i) || ']');
        END IF;
    END LOOP;
END print_list;
BEGIN
    -- Construct collection when one doesn't exist.
    IF NOT number_list.EXISTS(1) THEN
        number_list := number_table(1,2,3,4,5);
    END IF;
    -- Print initialized contents.
    DBMS_OUTPUT.PUT_LINE('Nested table before a deletion');
    print_list(number_list);
    -- Add two null value members at the end of the list.
    number_list.EXTEND(2);
    -- Add three members at the end of the list and copy the contents of item 4.
    number_list.EXTEND(3,4);
    -- Print revised contents.
    DBMS_OUTPUT.PUT_LINE(CHR(10) || 'Nested table after a deletion');
    print_list(number_list);
END;
/

```

Chương trình mẫu định nghĩa một tập hợp vô hướng cục bộ, định nghĩa một biến tập hợp không được khởi tạo, khởi tạo biến tập hợp, thêm hai phần tử giá trị rỗng và thêm ba phần tử có giá trị từ phần tử được tạo index bằng bốn. Phần hiển thị của chương trình sử dụng một thủ tục cục bộ để in nội dung hiện hành của một tập hợp. Phương thức **EXTEND** cấp phát không gian cho các nested table và cho phép sao chép nội dung từ một phần tử sang một tập hợp phần tử.

Nó tạo kết quả sau đây:

```

Nested table before a deletion

```

```
List [1]
List [2]
List [3]
List [4]
List [5]
Nested table after a deletion
```

```
List [1]
List [2]
List [3]
List [4]
List [5]
List []
List []
List [4]
List [4]
List [4]
```

Phương thức FIRST

Phương thức FIRST là một hàm. Nó trả về giá trị chỉ số dưới thấp nhất được sử dụng trong một tập hợp. Nếu nó là một index số, nó trả về một PLS_INTEGER. Nếu nó là một mảng kết hợp, nó trả về một kiểu dữ liệu VARCHAR2 hoặc LONG. Bạn không thể sử dụng phương thức FIRST trong một vòng lặp FOR dãy khi index không phải số.

Phương thức FIRST được minh họa trong chương trình mẫu cho phương thức DELETE. Ví dụ đó sử dụng một index số. Ví dụ sau đây minh họa phương thức FIRST với một index không số hoặc index chuỗi duy nhất. Như được thảo luận, các index không số trong các mảng kết hợp (associative array) mới trong chức năng Oracle 11g. Mệnh đề INDEX BY giúp bạn phân biệt giữa một nested table và một associative array bởi vì mệnh đề chỉ làm việc với các mảng kết hợp.

```
DECLARE
    TYPE number_table IS TABLE OF INTEGER INDEX BY VARCHAR2(9 CHAR);
    number_list NUMBER_TABLE;
BEGIN
    -- Add elements with unique string subscripts.
    number_list('One') := 1;
    number_list('Two') := 2;
```

```

number_list('Nine') := 9;
-- Print the first index and next.
DBMS_OUTPUT.PUT_LINE('FIRST Index [' || number_list.FIRST || ']');
DBMS_OUTPUT.PUT_LINE('NEXT      Index  ['      |      |
number_list.NEXT(number_list.
FIRST) || ']');
-- Print the last index and prior.
DBMS_OUTPUT.PUT_LINE(CHR(10) || 'LAST Index [' || number_list.LAST
|| ']');
DBMS_OUTPUT.PUT_LINE('PRIOR Index [' | | number_list.
PRIOR(number_list.
LAST) || ']');
END;
/

```

Chương trình mẫu định nghĩa một tập hợp vô hướng cục bộ, định nghĩa một biến cục bộ không được khởi tạo, gán những phần tử vào mảng kết hợp và in các giá trị index FIRST, NEXT, LAST và PRIOR. Nếu bạn ngạc nhiên khi nhìn vào kết quả, bạn đã không có được nó trước đó. Khi sử dụng một chuỗi duy nhất làm một giá trị index, thứ tự của các giá trị dựa vào môi trường NLS. Do đó, bạn tạo kết quả sau đây, được sắp xếp theo thứ tự bảng chữ cái:

```

FIRST Index [Nine]
NEXT Index [One]

LAST Index [Two]
PRIOR Index [One]

```

Phương thức LAST

Phương thức LAST là một hàm. Nó trả về giá trị chỉ số dưới cao nhất được sử dụng trong một tập hợp. Nếu nó là một index số, nó trả về một PLS_INTEGER. Nếu nó là một mảng kết hợp (associative array), nó trả về một kiểu dữ liệu VARCHAR2 hoặc LONG. Bạn không thể sử dụng phương thức LAST trong một vòng lặp FOR dãy khi index không phải số.

Phương thức LAST được minh họa trong chương trình mẫu cho phương thức DELETE. Ví dụ đó sử dụng một index số. Ví dụ trong phương thức FIRST cũng minh họa phương thức LAST với một index không phải số hoặc index chuỗi duy nhất. Như được thảo luận, các index không phải số trong các mảng vô hướng mới trong chức năng Oracle 11g.

Phương thức LIMIT

Phương thức LIMIT là một hàm. Nó trả về giá trị chỉ số dưới cao nhất có thể có được sử dụng trong một varray. Nó không có giá trị cho hai loại tập hợp khác. Nó trả về một PLS_INTEGER.

Chương trình mẫu sau đây minh họa phương thức LIMIT:

```

DECLARE
    TYPE number_varray IS VARRAY(5) OF INTEGER;
    number_list NUMBER_VARRAY := number_varray(1,2,3);

    -- Define a local procedure to check and print elements.
    PROCEDURE print_list(list_in NUMBER_VARRAY) IS
    BEGIN
        -- Print all subscripted elements.
        DBMS_OUTPUT.PUT_LINE('-----');
        FOR i IN list_in.FIRST..list_in.COUNT LOOP
            DBMS_OUTPUT.PUT_LINE('List Index [ ' || i || ' ] ' ||
                list_in(i));
        END LOOP;
    END print_list;
BEGIN
    -- Print initial contents.
    DBMS_OUTPUT.PUT_LINE('Varray after initialization');
    print_list(number_list);

    -- Extend with null element to the maximum limit size.
    number_list.EXTEND(number_list.LIMIT - number_list.LAST);

    -- Print revised contents.
    DBMS_OUTPUT.PUT_LINE(CHR(10));
    DBMS_OUTPUT.PUT_LINE('Varray after extension');
    print_list(number_list);
END;
/

```

Chương trình mẫu định nghĩa một tập hợp vô hướng cục bộ, định nghĩa một biến tập hợp không được khởi tạo, khởi tạo biến tập hợp và sau đó mở rộng không gian cho càng nhiều giá trị phần tử rỗng càng tốt. Nó in kết quả sau đây:

Varray after initialization

```
List Index [1] List Value [1]
List Index [2] List Value [2]
List Index [3] List Value [3]
Varray after extension
```

```
List Index [1] List Value [1]
List Index [2] List Value [2]
List Index [3] List Value [3]
List Index [4] List Value []
List Index [5] List Value []
```

Phương thức NEXT

Phương thức NEXT là một hàm. Nó trả về giá trị chỉ số dưới kế tiếp được sử dụng trong một tập hợp. Nếu không có giá trị chỉ số dưới cao hơn, nó trả về một giá trị rỗng. Nếu nó là một index số, nó trả về một PLS_INTEGER. Nếu nó là một mảng kết hợp, nó trả về một kiểu dữ liệu VARCHAR2 hoặc LONG.

Phương thức NEXT được minh họa trong chương trình mẫu cho phương thức DELETE. Ví dụ đó sử dụng một index số. Ví dụ trong phương thức FIRST cũng minh họa phương thức NEXT với một index không phải số hoặc index chuỗi duy nhất. Như được thảo luận, các index không phải số trong các mảng kết hợp mới trong chức năng Oracle 11g.

Phương thức PRIOR

Phương thức PRIOR thật sự là một hàm. Nó ví dụ giá trị chỉ số dưới trước được sử dụng trong một tập hợp. Nếu không có giá trị chỉ số dưới thấp hơn, nó trả về một giá trị rỗng. Nếu nó là một index số, nó trả về một PLS_INTEGER. Nếu nó là một mảng kết hợp, nó trả về một kiểu dữ liệu VARCHAR2 hoặc LONG.

Phương thức PRIOR được minh họa trong chương trình mẫu cho phương thức DELETE. Ví dụ đó sử dụng một index số. Ví dụ trong phương thức FIRST cũng minh họa phương thức PRIOR với một index không phải số hoặc index chuỗi duy nhất. Như được thảo luận, các index không phải số trong các mảng kết hợp là mới trong chức năng Oracle 11g.

Phương thức TRIM

Phương thức TRIM là một thủ tục. Nó là một thủ tục quá tải. Nếu khái niệm về quá tải (overload) mới mẻ đối với bạn, hãy xem chương 9.

Nó có một phiên bản không đòi hỏi các tham số hình thức. Khi được sử dụng mà không có các tham số hình thức, TRIM hủy cấp phát không gian cho một phần tử trong tập hợp. Tuy nhiên, nếu bạn cố TRIM không gian bên dưới zero phần tử, nó sẽ đưa ra một ngoại lệ.

Phiên bản còn lại đòi hỏi một tham số hình thức. Tham số phải là một giá trị số nguyên hợp lệ. TRIM với một tham số thật sự sẽ hủy cấp phát không gian cho số phần tử được xác định bởi tham số thật sự. Như với phiên bản không có tham số, việc cố TRIM không gian dưới zero phần tử sẽ đưa ra một ngoại lệ.

Chương trình mẫu sau đây minh họa phương thức TRIM:

```

DECLARE
    TYPE number_varray IS VARRAY(5) OF INTEGER;
    number_list NUMBER_VARRAY := number_varray(1,2,3,4,5);

    -- Define a local procedure to check and print elements.
    PROCEDURE print_list(list_in NUMBER_VARRAY) IS
    BEGIN

        -- Print all subscripted elements.
        DBMS_OUTPUT.PUT_LINE('-----');
        FOR i IN list_in.FIRST..list_in.COUNT LOOP
            DBMS_OUTPUT.PUT_LINE('List Index [ | | | | ] ' ||
                                'List Value [ | | list_in(i) | | ]');
        END LOOP;
    END print_list;
BEGIN
    -- Print initialized collection.
    DBMS_OUTPUT.PUT_LINE('Varray after initialization');
    print_list(number_list);

    -- Trim one element from the end of the collection.
    number_list.TRIM;

    -- Print collection minus last element.
    DBMS_OUTPUT.PUT(CHR(10));
    DBMS_OUTPUT.PUT_LINE('Varray after a trimming one element');
    print_list(number_list);

```

```

-- Trim three elements from the end of the collection.
   number_list.TRIM(3);
-- Print collection minus another three elements.
   DBMS_OUTPUT.PUT(CHR(10));
   DBMS_OUTPUT.PUT_LINE('Varray after a trimming three elements');
   print_list(number_list);
END;

```

Chương trình mẫu định nghĩa một biến vô hướng cục bộ, khai báo một biến tập hợp được khởi tạo, in nội dung, xén phần tử cuối cùng, in nội dung nhỏ hơn, xén ba phần tử cuối cùng và in những gì còn lại. Chương trình này in kết quả sau đây:

```

Varray after initialization
-----
List Index [1] List Value [1]
List Index [2] List Value [2]
List Index [3] List Value [3]
List Index [4] List Value [4]
List Index [5] List Value [5]
Varray after a trimming one element
-----
List Index [1] List Value [1]
List Index [2] List Value [2]
List Index [3] List Value [3]
List Index [4] List Value [4]
Varray after a trimming three elements
-----
List Index [1] List Value [1]

```

Bây giờ bạn đã xem qua Oracle 11g Collection API đầy đủ. Đến lúc tóm tắt những gì bạn đã học trong chương này.

Tóm tắt

Chương này đã đề cập đến định nghĩa và công dụng của các varray, nested table và các associative array vốn là các loại tập hợp Oracle 11g. Bạn đã làm việc qua những ví dụ trong SQL DML và PL/SQL sử dụng những tập hợp Oracle 11g. Cuối cùng, bạn cũng đã xem qua các chi tiết của Collections API.

Mục lục

Phần I : Các điểm cơ bản về Oracle PL/SQL 7

Chương 1: Tổng quan về Oracle PL/SQL 8

Lịch sử và thông tin cơ bản	8
Kiến trúc	10
Các cấu trúc khối cơ bản	12
Các tính năng mới của Oracle 10g	17
<i>Các gói cài sẵn</i>	17
<i>Các cảnh báo thời gian biên dịch</i>	18
<i>Biên dịch có điều kiện</i>	18
<i>Hành vi kiểu dữ liệu số</i>	19
<i>Trình biên dịch PL/SQL được tối ưu hoá</i>	20
<i>Các biểu thức thông thường</i>	20
<i>Lựa chọn trích dẫn</i>	21
<i>Các toán tử tập hợp</i>	21
<i>Các lỗi truy vết ngăn xếp</i>	22
<i>Các chương trình lưu trữ PL/SQL bao bọc</i>	23
Những tính năng mới của Oracle 11g	24
<i>Inlining chương trình con tự động</i>	25
<i>Câu lệnh Continue</i>	26
<i>Cache kết quả hàm PL/SQL session chéo</i>	26
<i>Các cải tiến SQL động</i>	27
<i>Các lệnh gọi ký hiệu tên hỗn hợp và vị trí</i>	28
<i>Các lệnh gọi PL/SQL</i>	28
<i>Ký hiệu vị trí</i>	28
<i>Ký hiệu định danh</i>	29
<i>Ký hiệu hỗn hợp</i>	29
<i>Ký hiệu loại trừ</i>	29
<i>Ký hiệu lệnh gọi SQL</i>	30
<i>Bộ chứa nối kết đa tiến trình</i>	30

PL/SQL Hierarchical Profile	33
PL/SQL Native Compiler tạo mã riêng	33
PL/Scope	34
Kiểu dữ liệu SIMPLE_INTEGER	34
Các lệnh gọi dãy trực tiếp trong các câu lệnh SQL	34
Tóm tắt	35
Chương 2: Các điểm cơ bản về PL/SQL	36
Cấu trúc khối Oracle PL/SQL	37
Các biến, phép gán và toán tử	40
Các cấu trúc điều khiển	43
<i>Các cấu trúc có điều kiện</i>	<i>43</i>
<i>Các cấu trúc lặp lại</i>	<i>47</i>
<i>Các vòng lặp WHILE</i>	<i>50</i>
Các hàm lưu trữ, thủ tục và gói (package)	51
<i>Các hàm lưu trữ</i>	<i>51</i>
<i>Các thủ tục</i>	<i>53</i>
<i>Các package</i>	<i>55</i>
Phạm vi giao tác	56
<i>Phạm vi giao tác đơn</i>	<i>56</i>
<i>Nhiều phạm vi giao tác</i>	<i>57</i>
Các trigger cơ sở dữ liệu	58
Tóm tắt	59
Chương 3: Các điểm cơ bản về ngôn ngữ	60
Các đơn vị ký tự và đơn vị từ vựng	60
<i>Các dấu tách</i>	<i>61</i>
<i>Các định danh</i>	<i>69</i>
<i>Các trực kiện</i>	<i>70</i>
<i>Các trực kiện số</i>	<i>71</i>
<i>Các chú giải</i>	<i>73</i>
Các cấu trúc khối	74
Các kiểu biến	79
<i>Các kiểu dữ liệu vô hướng</i>	<i>83</i>
Các đối tượng lớn (LOB)	103
Các kiểu dữ liệu tổng hợp	107

<i>Các record</i>	107
<i>Các cursor tham chiếu hệ thống</i>	115
Phạm vi biến	118
Tóm tắt	119
Chương 4: Các cấu trúc điều khiển	120
Các câu lệnh có điều kiện	121
<i>Các câu lệnh IF</i>	130
<i>Các câu lệnh CASE</i>	134
<i>Các câu lệnh biên dịch có điều kiện</i>	137
Các câu lệnh lặp lại	139
<i>Các câu lệnh vòng lặp đơn giản</i>	140
<i>Các câu lệnh vòng lặp FOR</i>	145
<i>Các câu lệnh vòng lặp WHILE</i>	148
Các cấu trúc Cursor	149
<i>Các cursor ngầm định</i>	150
<i>Các cursor tường minh</i>	154
Các câu lệnh Bulk	162
Các câu lệnh BULK COLLECT INTO	163
<i>Các đích tập hợp được ràng buộc bởi LIMIT</i>	166
Các câu lệnh FORALL	169
Tóm tắt	173
Chương 5: Quản lý lỗi	174
Các loại ngoại lệ và phạm vi	175
<i>Các lỗi biên dịch</i>	176
<i>Các lỗi run-time</i>	179
<i>Các lỗi khối khai báo</i>	183
Các hàm cài sẵn quản lý ngoại lệ	185
<i>Các ngoại lệ do người dùng định nghĩa</i>	188
<i>Khai báo các ngoại lệ do người dùng định nghĩa</i>	189
Các ngoại lệ động do người dùng định nghĩa	190
Các hàm ngăn xếp ngoại lệ	192
<i>Quản lý ngăn xếp ngoại lệ</i>	192
<i>Định dạng ngăn xếp lỗi</i>	197
Quản lý ngoại lệ Trigger cơ sở dữ liệu	201

Các trigger cơ sở dữ liệu lỗi quan trọng 201
Các trigger cơ sở dữ liệu lỗi không quan trọng 208
Tóm tắt 211

Phần II: Lập trình PL/SQL212

Chương 6: Các hàm và thủ tục 213

Cấu trúc hàm và thủ tục 214
Phạm vi giao tác 223
Gọi các thường trình con 223
Ký hiệu vị trí 224
Ký hiệu định danh 224
Ký hiệu hỗn hợp 225
Ký hiệu loại trừ 225
Ký hiệu lệnh gọi SQL 226
Các hàm 226
Các tùy chọn tạo 228
Các hàm chuyển theo giá trị 242
Các hàm chuyển theo tham chiếu 250
Các thủ tục 254
Các thủ tục chuyển theo giá trị 255
Các thủ tục chuyển theo giá trị 262
Tóm tắt 270

Chương 7: Các tập hợp 271

Các loại tập hợp 274
Varrays 277
Các Nested Table 293
Các Associative Array 310
Các toán tử tập hợp 321
Toán tử CARDINALITY 324
Toán tử EMPTY 325
Toán tử MEMBER OF 325
Toán tử MULTISSET INTERSECT 325
Toán tử MULTISSET INTERSECT 326
Toán tử MULTISSET UNION 326

<i>Toán tử SET</i>	328
<i>Toán tử SUBMULTISET</i>	329
Collection API	330
<i>Phương thức COUNT</i>	334
<i>Phương thức DELETE</i>	334
<i>Phương thức EXISTS</i>	336
<i>Phương thức EXTEND</i>	338
<i>Phương thức FIRST</i>	340
<i>Phương thức LAST</i>	341
<i>Phương thức LIMIT</i>	342
<i>Phương thức NEXT</i>	343
<i>Phương thức PRIOR</i>	343
<i>Phương thức TRIM</i>	343
Tóm tắt	345

**Giáo trình Hướng dẫn Lý thuyết
và kèm theo bài tập thực hành
ORACLE 11g (Tập 1)**

Th.S: NGUYỄN QUẢNG NINH - NGUYỄN NAM THUẬN

Chịu trách nhiệm xuất bản

HOÀNG CHÍ DŨNG

Biên tập : NAM THUẬN

Trình bày : CÔNG SƠN

Bìa : LÊ THÀNH

TỔNG PHÁT HÀNH

Nhà Sách Nhân Văn

- Số 01 Trường Chinh – P. 11– Q.Tân Bình – TP.HCM
- ĐT: 9712285 – 9710306 – 8490048 – Fax: 9712286
- Số 486 Nguyễn Thị Minh Khai – P. 2– Q.3 ĐT: 8396733
- Số 875 CMT8, P.15, Q.10 ĐT: 9708161

NHÀ XUẤT BẢN HỒNG ĐỨC

111 LÊ THÁNH TÔN, Q.1, TP.HCM TEL: 08.38244534

In 1000 bản khổ 16x24cm tại Công ty IN VẠN XUÂN

Giấy ĐKKHXB số : 323-2008/CXB/T1-42-24/HĐ

Cấp ngày 03-11-2008

In xong và nộp lưu chiểu tháng quý I năm 2009.

Th.S NGUYỄN QUẢNG NINH - NGUYỄN NAM THUAN
và nhóm tin học thực dụng



GIÁO TRÌNH
HƯỚNG DẪN LÝ THUYẾT
KÈM THEO BÀI TẬP THỰC HÀNH

Oracle 11g

DÀNH CHO HỌC SINH, SINH VIÊN

ẤN BẢN 2009

TẬP 2



NHÀ XUẤT BẢN HỒNG ĐỨC

6387 263



NVC
NHAN VAN CORP.

HỆ THỐNG NHÀ SÁCH NHÂN VĂN



NhanVan.vn
Nhà sách trực tuyến

TRUNG TÂM PHÁT HÀNH SÁCH:

* 875 CMT8 - P. 15 - Q.10,
TP. Hồ Chí Minh
Tel: 9770096 Fax: 9708161

* 486 Nguyễn thị Minh Khai,
P.2, Q. 3, TP. Hồ Chí Minh
Tel/Fax: 8396733

* 01 Trương Chinh, P.11, Q. Tân Bình,
TP. Hồ Chí Minh
Tel: 9712285 Fax: 9712286

ORACLE 11g 1



8 93 5072 840303

GIÁ: 63.000Đ