

CS3743 Introduction to Database Systems Oracle PL/SQL (2)

Weining Zhang

Department of Computer Science
University of Texas at San Antonio

August 4, 2009

Outline

- 1 Oracle PL/SQL
 - Packages
 - Cursors
 - Triggers

Package

- A package is a group of related PL/SQL objects (variables, ...), procedures, and functions.
- Each package definition has two parts:
 - Package specification that provides an interface to the users of the package.
 - Package body that contains the actual code.

A Sample Package Specification

```
create or replace package banking as
function check_balance
(account_no in Accounts.acct_no%type)
return Accounts.balance%type;
procedure deposit
(account_no in Accounts.acct_no%type,
amount in Accounts.balance%type);
procedure withdraw
(account_no in Accounts.acct_no%type,
amount in Accounts.balance%type);
end;
/
show errors
```

A Sample Package Body

```
create or replace package body banking as
function check_balance
(account_no in Accounts.acct_no%type)
return Accounts.balance%type is
acct_balance Accounts.balance%type;
begin
select balance into acct_balance
from Accounts where acct_no = account_no;
return acct_balance;
end;
```

A Sample Package Body

```
procedure deposit
(account_no in Accounts.acct_no%type,
amount in Accounts.balance%type) is
begin
if (amount <= 0) then
dbms_output.put_line('Wrong amount.');
```

A Sample Package Body

```
procedure withdraw
(account_no in Accounts.acct_no%type,
amount in Accounts.balance%type) is
acct_balance Accounts.balance%type;
begin
acct_balance := check_balance(account_no);
if (amount > acct_balance) then
dbms_output.put_line('Insufficient fund.');
```

Use Packages

Example

Compile a package

```
SQL> start packspec1 (or @packspec1)
SQL> start packbody1
```

May use execute to invoke a package.

```
SQL> execute banking.deposit(100, 200);
SQL> execute banking.withdraw(200, 500);
```

Define a variable in SQL*Plus, call a function, then print the result

```
SQL> var bal number
SQL> execute :bal := banking.check_balance(200);
SQL> print bal
```

Built-in Packages

Example

Package `dbms_output` is primarily used for debugging PL/SQL programs. It contains several procedures.

- `dbms_output.disable`.
- `dbms_output.enable(20000)`. Set buffer size to 20,000 bytes (default is 2000)
- `put_line() & put()`.
- `new_line`.
- `Get_line(line, status)`. (`status=0` or `1`)

Cursors

- Cursor is a mechanism in SQL programs to access the result of a query one tuple at a time.
- A cursor is defined by a query and accessed using operators such as `open`, `fetch` and `close`

```
declare
  cursor c1 is select cid, cname, city from customers;
  c1_rec c1%rowtype;
begin
  open c1;
  fetch c1 into c1_rec;
  dbms_output.put_line(c1_rec.cid || ', ' || c1_rec.cname ||
    ', ' || c1_rec.city);
  close c1;
end;
/
```

Find Information About Packages

Information about packages is in the data dictionary

Example

What packages do we have?

```
select object_name, object_type, created
from user_objects
where object_type = 'PACKAGE';
```

How is a package programmed?

```
select text from user_source
where name = 'BANKING';
```

Cursor's Attributes

Each cursor has several predefined attributes that can be tested.

```
begin
  if (not c1%isopen) then /* attribute */
    open c1;
  end if;
  fetch c1 into c1_rec;
  while c1%found loop /* attribute */
    dbms_output.put_line(c1_rec.cid || ', ' || c1_rec.cname ||
      ', ' || c1_rec.city);
    fetch c1 into c1_rec;
  end loop;
  close c1;
end;
/
```

Cursor In a For-Loop

```
begin
  for c1_record in c1 loop
    if (c1_record.city = 'New York') then
      dbms_output.put_line(c1_rec.cid || ', ' ||
        c1_rec.cname || ', ' || c1_rec.city);
    end if;
  end loop;
end;
/
```

- Here, open, fetch and close of the cursor are made implicitly.
- There is no need to declare c1_record in the for loop.

Use Cursor For Update

```
declare
  cursor c1 is select * from customers for update;
begin
  for c1_rec in c1 loop
    if (c1_rec.city = 'New York') then
      delete from customers where current of c1;
    end if;
  end loop;
end;
/
```

Parameterized Cursor

Example

Declaration:

```
cursor c1(d_name in Students.dept_name%type) is
  select age, avg(GPA)
  from Students
  where dept_name = d_name
  group by age;
```

Usage:

```
open c1('Computer Science');
```

Triggers

- A trigger is an event-condition-action rule coded in PL/SQL and is useful for enforcing integrity constraints and business rules.
 - An event is an update operation: insertion, deletion or update.
 - An action is a set of additional update operations or other PL/SQL statements.
 - A trigger fires (i.e., executes the action) if the event occurs and the condition is satisfied.

Row Trigger

- Fire once for each row that is updated in the event and satisfies the specified condition
- It has two built-in references: new & old

```
create or replace trigger raise_sal
before update of salary on employees
for each row
when (new.salary > old.salary * 1.2)
begin
  dbms_output.put_line('Old salary is ' || :old.salary ||
    ' , ' || 'New salary is ' || :new.salary);
  dbms_output.put_line('The raise is too high!');
end;
```

A Use of Trigger

A record needs to be added to a log whenever someone changes the price of a product. There are two tables.

```
products(pid, description, price,...)
products_log (pid, name, date, old_price, new_price)
```

This can be automated by a trigger.

```
create or replace trigger update_p_price
after update of price on products
for each row
begin
  insert into products_log
  values (:old.pid, user, sysdate, :old.price, :new.price);
end;
```