Oracle PL/SQL (2)

John Ortiz

Package

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

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
```


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;
```

Package Body (cont.)

```
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.');
  else  update Accounts
        set balance = balance + amount
        where acct_no = account_no;
  end if;
end;
```


Package Body (cont.)

```
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.');
```

Package Body (cont.)

```
    else  update Accounts
          set balance = balance - amount
          where acct_no = account_no;
    end if;
  end;
end; /* end of the package body */
/
show errors
```


Public versus Private Constructs

- Any construct listed in package specification is public and accessible to anyone else.
- Constructs listed in package body but not in package specification are private and accessible only to other constructs in the same package.

Use of Package

- Objects defined in a package specification can be used in other PL/SQL packages, blocks and SQL queries.

  Package_Name.Object

- Compile the 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);

Use of a Stored Function

- Declare a variable

  SQL> var bal number
- Execute the function

  SQL> execute :bal :=

  banking.check_balance(200);

  ☛Must precede variable with a colon ":".
- Print the value

  SQL> print bal

Find About Packages

- What packages do we have?

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

- What code is in the package?

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

☛ Both specification and body will be shown.

Cursor

- A construct to access query result one tuple at a time.
- Define a cursor:

```
cursor c1 is
select cid, cname, city
from customers;
```

- Working with a cursor:
  - open cursor_name;
  - fetch cursor_name into record or variables;
  - close cursor_name;

An Example of Cursors

```
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;
/
```


Cursor Attributes

```
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 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;
/
```

- open, fetch and close are done implicitly.
- No need to declare c1_record.

Cursor for Update

- Declare a cursor for update

```
cursor c1 is select cid, cname, city
             from customers for update;
```

- Use the current tuple in a cursor.

```
update customers
  set city = 'Boston' where current of c1;

delete from customers
  where current of c1;
```

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

- 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');
```

Built-in Package: dbms_output

- dbms_output is primarily used to help debugging PL/SQL programs.
- Has following public 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)

Trigger

- A trigger is an event-condition-action rule coded in PL/SQL and is useful for enforcing various integrity constraints and business rules.
- An event is an update operation: insertion, deletion or update.
- The action can be a set of additional update operations or other PL/SQL statements.
- A trigger fires (executes the action) at a time before or after an event occurs and additional condition are satisfied.

A Sample Trigger

```
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;
```


Row Trigger

- Row Trigger
  - Fire once for each row that is affected by the event and satisfies the additional condition in the when clause.
  - Must specify *for each row*.
- Predefined references: new & old
  - new is applicable for insert and update
  - old is applicable for update and delete
  - use :new.salary & :old.salary in trigger body

Trigger Applications

- Add a log entry each time the price of a product is changed.
  - The log table:

    products_log (pid, username, update_date, old_price, new_price);

Trigger Application (cont.)

- Create 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;
```

Another Trigger Application

- If a student is removed, delete all enrollments by the student.

```
create or replace trigger stud_enroll
after delete on students
for each row
begin
 delete from enrollments where sid = :old.sid;
end;
```