

## Chapter 19

# PROCEDURES, FUNCTIONS AND PACKAGES

- ❑ What is a stored procedure?
- ❑ Advantages of stored procedures
- ❑ Creating a stored procedure
- ❑ Creating a stored function
- ❑ Recompiling
- ❑ Types of parameters
- ❑ Parameter modes
- ❑ NCOPY compiler hints
- ❑ RAISE\_APPLICATION\_ERROR procedure
- ❑ Packages

## What is a stored procedure?

As we have seen in the introduction to PL/SQL that there are two types of PL/SQL blocks – anonymous and stored procedures.

A stored procedure is a PL/SQL block that is stored in the database with a name. It is invoked using the name. Each procedure is meant for a specific purpose.

A stored procedure is stored in the database as an object. It is also called as database procedure as it is stored in the database.

A procedure may take one or more parameters. If a procedure takes parameters then these parameters are to be supplied at the time of calling the procedure.

### **What is a function?**

A function is similar to procedure, except that it returns a value. The calling program should use the value returned by the function.

### **Advantages of stored procedures**

Stored procedures and functions offer the following benefits.

#### **Reduced network traffic**

Stored procedures are stored in the database on the server. In order to invoke a stored procedure a client has to use only the name of the stored procedure. This results in reduced network traffic because only the name is passed to server and not the entire PL/SQL block.

#### **Better Performance**

Stored procedures improve performance because the database accessed by stored procedures and stored procedures themselves are stored in the database. Furthermore, because a stored procedure's compiled form is available in the database, no compilation step is required to execute the code.

Apart from this, stored procedures make use of shared memory. That means if a stored procedure is called by user-1, then it is brought into server's memory. As a result there is a chance of finding the stored procedure in memory, when user-2 later wants to use the same procedure. In this case stored procedure need not be brought into memory since it is already in the memory. This saves time and increases performance.

#### **Easy Maintenance**

If there is any change in the application, only the concerned stored procedures are to be modified to incorporate the change. As all the clients access stored procedures, they automatically use the modified definition. You only have to change business logic once in the stored procedure. Therefore stored procedures improve the integrity and consistency of your applications.

#### **Security**

Stored procedures help enforcement of data security. You can allow users to access only the stored procedures and not the table that are manipulated by stored procedures.

Whenever a stored procedure runs, it runs with the privileges of the owner and not the user running it. Because users only have the privilege to execute the procedure and not the privileges to access the underlying tables, it increases security.

### Owner rights

When a stored procedure is run under the privileges of the owner of the stored procedure, it is called as owner rights.

### Invoker rights

When a stored procedure is executed with the privileges of the invoker and not under the privileges of the owner, it is called as invoker rights.

We will discuss about the difference between these two ways of running a stored procedure in detailed later in this chapter.

## Creating a stored procedures

A stored procedure is created using CREATE PROCEDURE command.

```
CREATE [OR REPLACE] PROCEDURE name [(parameter[,parameter, ...])] {IS
|AS}
[local declarations]
BEGIN
    executable statements
[EXCEPTION
    exception handlers]
END [name];
```

**OR REPLACE** is used to create a procedure even a procedure with the same name is already existing.

Oracle creates procedure even though the procedure is not valid but it displays the message saying that the procedure is created with errors. Then you have to rectify the errors and recreate the procedure. If OR REPLACE option is not given Oracle doesn't allow you to replace an existing stored procedure. So, it is better you use OR REPLACE option so that the existing invalid version of the procedure will be replaced with the valid version.

### SHOW ERRORS

During the creation of the procedure if there are any errors Oracle displays the message saying procedure is created but with errors with the following message:

```
Warning: Procedure created with compilation errors.
```

In order to displays errors in the most recent CREATE PROCEDURE statement, use SHOW ERRORS command.

The following stored procedure inserts a new record into PAYMENTS table with the given roll number and amount.

```
create or replace procedure  newpayment(rollno number, amt number)
is
begin
    insert into payments values(rollno, sysdate,amt);
    commit;
end;
/
```

Once a stored procedure is stored in the database, it can be invoked using EXECUTE command by using the name of the procedure.

### EXECUTE command

A procedure can be executed using EXECUTE command. To execute a procedure either you must be the owner of the procedure or you must have EXECUTE privilege on the procedure.

The following example shows how to invoke NEWPAYMENT procedure.

```
SQL> execute newpayment(10,2000);
```

```
PL/SQL procedure successfully completed.
```

In the above example, NEWPAYMENT is invoked by passing 10 and 2000. It inserts a new row into PAYMENTS table with values 10, sysdate, and 2000.

### Creating a stored function

A stored function is same as a procedure, except that it returns a value. CREATE FUNCTION command is used to create a stored function.

```
CREATE [OR REPLACE] FUNCTION name
    [(parameter[,parameter, ...])]
    RETURN datatype
{IS | AS}
[local declarations]
BEGIN
    executable statements

    RETURN value;
```

```
[EXCEPTION
      exception handlers]
END [name];
```

**OR REPLACE** is used to create a function even though a function with the same name already exists

**RETURN datatype** specifies the type of data to be returned by the function.

**RETURN** statement in the executable part returns the value. The value must be of the same type as the return type specified using RETURN option in the header.

The following stored function takes roll number of the student and return the amount yet to be paid by the student.

```
create or replace function  getdueamt(prollno number)
return number
is
  v_fee      number(5);
  v_amtpaid number(5);
begin
  -- get total amount paid by student
  select sum(amount) into v_amtpaid
  from payments
  where rollno = prollno;
  -- get course fee of the course into which student joined
  select fee into v_fee
  from courses
  where ccode = ( select ccode from batches
                  where bcode in
                    ( select bcode from students
                      where rollno = prollno)
                  );
  -- return the difference
  return v_fee - v_amtpaid;
end;
/
```

The above function can be called from a PL/SQL block as follows:

```
begin
  dbms_output.put_line(getdueamt(10));
end;
```

User-defined PL/SQL functions can be used in SQL in the same manner as the standard functions such as ROUND and SUBSTR..

For example, the function GETDUEAMT can be invoked from SELECT command as follows:

```
SQL> select rollno, getdueamt(rollno)
       2  from students;
```

| ROLLNO | GETDUEAMT (ROLLNO) |
|--------|--------------------|
| 1      | 0                  |
| 2      | 0                  |
| 3      | 0                  |
| 4      | 0                  |
| 5      | 0                  |
| 6      | 0                  |
| 7      | 0                  |
| 8      | 0                  |
| 9      | 1500               |
| 10     | 2000               |
| 11     | 0                  |
| 12     | 1500               |
| 13     | 0                  |

### Getting source code

It is possible to see the source code of stored procedures and function by using USER\_SOURCE data dictionary view.

The following SELECT command will display the source code of NEWPAYMENT stored procedure.

```
SQL> select text
       2  from user_source
       3  where name = 'NEWPAYMENT';
```

TEXT

```
-----
procedure newpayment(rollno number, amt number)
is
begin
  insert into payments values(rollno, sysdate,amt);
  commit;
end;
```

## Privileges required

To create a stored procedure, you must have CREATE PROCEDURE system privilege.

You must also have required object privileges on the objects that are referred in the procedure in order to successfully compile the procedure.

---

**Note:** The owner of the procedure CANNOT obtain required privileges on the stored procedure through ROLES.

---

## Recompiling

Stored procedures and functions are compiled at the time of creation and stored in the compiled form. If a procedure becomes invalid afterwards, it is to be recompiled before it is executed. Oracle implicitly compiles the procedure when an invalid procedure is referred. However, it is possible to explicitly recompile

In order to recompile a stored procedure use ALTER PROCEDURE and ALTER FUNCTION to recompile a procedure and function respectively.

```
ALTER PROCEDURE procedurename    COMPILE;  
ALTER FUNCTION  functionname     COMPILE;
```

The following sequence will illustrate the importance of recompilation.

- ❑ Assume user SCOTT has created NEWPAYMENT procedure as follows

```
create or replace procedure  newpayment(rollno number, amt number)  
is  
begin  
    insert into book.payments values(rollno, sysdate,amt);  
    commit;  
end;  
/
```

- ❑ Since SCOTT doesn't have INSERT privilege on PAYMENTS table of BOOK, the procedure is created but marked as invalid. You can see the status of the procedure using the following command.

```
select status from user_objects where object_name = 'NEWPAYMENT';  
  
STATUS  
-----  
INVALID
```

- ❑ Now, user BOOK has granted INSERT privilege on PAYMENTS table to SCOTT as follows:

```
GRANT INSERT ON PAYMENTS to SCOTT;
```

- ❑ Then any subsequent reference to NEAPAYMENT procedure in SCOTT will implicitly recompile the procedure. But in order to avoid extra time taken to recompile the procedure at runtime, it can be recompiled using ALTER PROCEDURE command as follows:

```
ALTER PROCEDURE newpayment COMPILE;
```

- ❑ After recompilation, the procedure will have status VALID as provide by the following query.

```
select status from user_objects where object_name = 'NEWPAYMENT';
```

```
STATUS  
-----  
VALID
```

## Types of Parameters

Parameters of a procedure are of two types.

- ❑ Formal parameters
- ❑ Actual Parameters

### Formal Parameters

The parameters declared in the definition of procedure are called as **formal parameters**. They receive the values sent while calling the procedure.

```
procedure increase_fee (pccode varchar2, pamt number)
```

In the above procedure, PCCODE, PAMT parameters are called as formal parameters.

## Actual Parameters

The values given within parentheses while calling the procedure are called as **actual parameters**.

```
increase_feepaid ( v_ccode, 2000);
```

v\_ccode and 2000 are actual parameters. These values are copied to the corresponding formal parameters - pccode and *pamt*.

## Parameter Modes

Parameter mode is used to specify what can be done with formal parameter. The following are the available modes.

- ❑ IN
- ❑ OUT
- ❑ IN OUT

### IN mode

IN parameters lets you pass a value to the subprogram being called. The value cannot be changed inside the subprogram. It is like a constant in the subprogram. Therefore it cannot be assigned a value.

```
procedure increase_fee (pccode in varchar2, pamt number) is
begin
    . . .
end;
```

The actual parameter corresponding to IN parameter can be a variable, constant, or expression.

The default parameter mode is IN. In the above example though we didn't specify the parameter mode for PAMT is it taken as IN parameter.

### OUT Mode

An OUT parameter lets the subprogram pass a value to caller. Inside the subprogram OUT parameter is an uninitialized variable.

Subprogram has to place a value in the OUT parameters, which will be used by caller program. Whatever changes are made to formal parameter, the changes will be made available to actual parameter.

---

The actual parameter corresponding to OUT parameter must be a variable.

### IN OUT Mode

It is same as IN and OUT modes put together. It can get a value from the calling program and return value to calling program. The value of this type of parameter can be used in the subprogram and

The actual parameter corresponding to IN OUT parameter must be a variable.

The following procedure takes course code and returns the dates on which the first and last batches of that course have started.

```
create or replace procedure get_dates( pccode in varchar2,
                                     first_date out date,
                                     last_date out date) is
begin
    select min(stdate) into first_date
    from   batches
    where  ccode = pccode;

    select max(stdate) into last_date
    from   batches
    where  ccode = pccode;

end;
```

Once the procedure is created, it can be called as follows:

```
declare
    min_date date;
    max_date date;
begin
    get_dates( 'ora', min_date, max_date);
    dbms_output.put_line( min_date || ':' || max_date);
end;
```

The output of the above program will be:

```
12-JAN-01:15-AUG-01
```

## NOCOPY Compiler Hint

By default, OUT and IN OUT parameters are passed by value. That means, the value of an IN OUT actual parameter is copied into the corresponding formal parameter. Then, if the procedure exits normally, the values assigned to OUT and IN OUT formal parameters are copied into the corresponding actual parameters.

When the parameters hold large data structures such as records, and instances of object types (which we will in later chapters), copying slows down execution and uses up more memory. To prevent that, you can specify the NOCOPY hint, which allows the PL/SQL to pass OUT and IN OUT parameters by reference.

```
PROCEDURE change(pstudent IN OUT NOCOPY student_type) IS ...
```

In the above example parameter PSTUDENT is of object type – STUDENT\_TYPE. It is now passed by reference as we used NOCOPY option with it.

## Invoker Rights vs. definer rights

By default, when a user executes a procedure, the procedure is executed with the privileges of the owner. That means, the privileges of invoking user (invoker) are not taken into account only the privileges of definer (owner) of the procedure will be considered.

If the procedure is to be called by user other than the owner then the use must be granted EXECUTE privilege on the procedure.

When the procedure is executed using the privileges of definer then, it is called as definer rights.

Definer rights is the default option. All object references in the procedure will be referring to objects of definer and not invoker. Let us see an example to understand this point further.

Assume we have procedure ADDMESSAGE created by user SRIKANTH as follows:

```
create or replace procedure addmessage(msg varchar2)
is
begin
    insert into messages values (msg, sysdate);
    commit;
end;
```

Then user SRIKANTH has granted EXECUTE privilege to user PRANEETH as follows:

```
grant execute on addmessage to praneeth;
```

Now user PRANEETH can execute the procedure as follows:

```
execute Srikanth.addmessage('First message');
```

The message "First message" is inserted into MESSAGES table of SRIKANTH – the definer of the procedure.

What if user PRANEETH also has MESSAGE table with the same structure as MESSAGES table of SRIKANTH? The answer is; even now the message goes to MESSAGES table of SRIKANTH, since all references to objects in the procedure are resolved to definer of the procedure.

## Invoke Rights

Oracle8i has introduced invoker rights. In case of invoker rights, procedure is executed with the privileges of invoker and not the definer of the procedure.

If you want the procedure to be executed with invoker rights and not with definer right, then the procedure is to be created with AUTHID CURRENT\_USER option as follows.

```
create or replace procedure addmessage(msg varchar2)
authid current_user as
is
begin
    insert into messages values (msg, sysdate);
    commit;
end;
```

AUTHID CURRENT\_USER option specifies that the procedure to be executed under the privileges of invoking user. Also remember that all object references are resolved to invoker and not the definer. However, if any object is qualified with schema (username) then it will refer to object in that schema. In other words, all unqualified object are taken from invoker's schema.

Now if you execute ADDMESSAGE procedure from user PRANEETH, it will fail, as there is no MESSAGES table in that schema.

```
SQL> execute srikanth.addmessage('Second message');
BEGIN srikanth.addmessage('fine'); END;
```

\*

```
ERROR at line 1:
ORA-00942: table or view does not exist
ORA-06512: at "SRIKANTH.ADDMESSAGE", line 4
ORA-06512: at line 1
```

As you can understand, the above message is complaining about missing MESSAGES table. If you create MESSAGES table in PRANEETH schema, then the procedure will succeed and the message is inserted into MESSAGES table of PRANEETH account.

The advantage of invoker rights is that it allows you to centralized code whereas the data is stored in individual schema. That means though users use the same common code the data is stored in objects of their schema.

## RAISE\_APPLICATION\_ERROR Procedure

This procedure is used to create your own application error numbers and messages.

When you have to terminate a PL/SQL program, you can display an error message and send error code to host environment using this procedure.

```
RAISE_APPLICATION_ERROR (errornumber,  errormessage);
```

**errornumber** is a number in the range -20001 and -20999.

**errormessage** is the message to be displayed.

The following PL/SQL block displays the error message along with error number (-20100) when NO\_DATA\_FOUND exception is raised.

```
declare
    v_fee courses.fee%type;
begin

    select fee into v_fee
    from   courses
    where  ccode = 'ora';

    -- remaining statements

exception
    when no_data_found then
        raise_application_error(-20100, 'Invalid course code');
end;
/
```

## Packages

A package is a collection of related procedures, functions, variables and data types. A package typically contains two parts – specification and body.

## Specification

Package specification contains declarations for items, procedure and functions that are to be made public. All public objects of package are visible outside the package. In other words, public objects are callable from outside of the package.

Private items of the package can be used only in the package and not outside the package.

The following is the syntax to create package specification.

```
CREATE PACKAGE package_name AS
    /* declare public objects of package */
END;
```

## Body

Body of the package defines all the objects of the package. It includes public objects that are declared in package specification and objects that are to be used only within the package – private members.

```
CREATE PACKAGE BODY package_name AS
    /* define objects of package */
END;
```

- procedures declared in the package specification
- functions declared in the package specification
- definition of cursors declared in the package specification
- local procedures and functions, not declared in the package specification
- local variables

## Calling a procedure of package

In order to access a public object of a package use the following syntax:

```
package_name.object_name
```

package\_name is the name of the package whose object you want to access.

*object\_name* is the name of a public object in the package.

Let us now create a package called COURSE\_PKG that contains two subprograms – CHANGEFEE and GETBATCHCOURSE.

We have to first create package specification and then body as follows:

```
create or replace package course_pkg as
    procedure changefee (pccode varchar2, newfee number);
    function getbatchcourse(pbcode varchar2) return varchar2;
end;
/
```

The following CREATE PACKAGE BODY statement creates the body of the package.

```
create or replace package body course_pkg as
    procedure changefee (pccode varchar2, newfee number)
    is
    begin
        update courses set fee = newfee
        where ccode = pccode;
        if sql%found then
            commit;
        else
            raise_application_error(-20010,'Invalid course code');
        end if;
    end;

    function getbatchcourse(pbcode varchar2) return varchar2
    is
        v_ccode courses.ccode%type;
    begin
        select ccode into v_ccode
        from batches
        where bcode = pbcode;

        return v_ccode;

    exception
        when no_data_found then
            raise_application_error( -20011,'Invalid batch code');

    end;

end;
```

In order to call procedure CHANGEFEE of the package, use the package name followed by procedure name as follows:

```
Execute course_pkg.changefee('ora',5000);
```

### Initializing package

It is possible to execute some code when a user refers the package for the first time. Any code that you feel is to be executed at the time of a user referring to the package for the first time can be put in initialization part of the package.

The following is the structure of initialization part.

```
create package body name as

    /* definition of package body */

begin
    /* code to be executed when package is referred for the first time */
end;
```

### Summary

Stored procedure and functions are sub programs stored in database. Functions return a single value whereas procedure doesn't return any value. Stored procedures have important advantages such as improving performance, making maintenance and security implementation easy.

Oracle allows parameters of three types – in, out, and in out. OUT and IN OUT parameters are used to return values back to calling program. They can be passed by either value (default) or by reference using NOCOPY hint.

Procedures are executed under privileges of owner of the procedure. However, it is possible to execute procedure with the privileges of invoker using AUTHID CURRENT\_USER option of CREATE PROCEDURE command.

Standard procedure RAISE\_APPLICATION\_ERROR is used to raise an application error with the given error number and message.

Package allows a group of related procedures and function to be identified by a single name. It is used to avoid name conflicts between names of different procedures.

## Exercises

1. \_\_\_\_\_ command is used to display errors that occurred during compilation of a stored procedure.
2. \_\_\_\_\_ view provides information about stored procedures.
3. \_\_\_\_\_ option is used to specify that a parameter is both input and output parameter.
4. What is the command used to compile a procedure explicitly?
5. Create a procedure to insert a new row into payments with the given roll number. DP is system date and AMOUNT is the amount to be paid by the student.
6. Create a function to take batch code and return the number of students in the batch.
7. Create a function to return the first missing roll number. If no roll number is missing then return the highest roll number + 1.
8. Create a function to take faculty code and return the number of batches the faculty can handle.
9. Create a procedure to take course code and return minimum and maximum duration of batches of that course.
10. Create a package to contain the following procedures and functions.  
Function BATCHSTATUS – takes batch code and returns S - if batch is yet to start, C – if batch is completed, R – if batch is currently running.  
Function BATCHAMOUNT – return the total amount collected from the given batch code.