PL/SQL: Functions, Procedures, Packages

User-defined Functions and Procedures in Anonymous Blocks

User-defined functions and procedures are to be located in the *declaration* part of PL/SQL's anonymous blocks.

Procedure declaration

Procedures are callable blocks of PL/SQL code which produce side-effects (e.g., changes in the database, printed output), but do not return any values.

The syntax of the procedure declaration is

```
procedure <Name> [(<parameters>)] is
  [<declarations>]
begin
  <statements>;
  [exception  <statements>;]
end;
```

`<Name>` : name of the procedure
`<parameters>` : comma-separated list of parameters (see below)
`<declarations>` sequence of type, variable and constant declarations

The syntax for defining a parameter of the procedure is

```
<VarName> [in] [out] <type>
```

- **in parameters:** input parameters: must contain value when the procedure is called, do not change value in the procedure (analagous to pass-by-value).
- **out parameters:** output parameters: need not contain value when the procedure is called, should contain value when the procedure is over (analagous to pass-by-name/pass-by-reference).
- **in out parameters:** input-output parameters: must contain value when the procedure is called, the value may change during the procedure, new value is returned (analogous to pass-by-name/pass-by-reference).

Function declaration

Functions are callable blocks of PL/SQL code which may produce side-effects, but also return values.

The syntax of the procedure declaration is

```
function <Name> [(<parameters>)] return <type> is
  [<declarations>]
begin
  <statements>;
  [exception  <statements>;]
end;
```

`<Name>`	: name of the function
`<type>`	: return type of the function
`<parameters>`	: comma-separated list of parameters (specified in the same way as for procedures)
`<declarations>`	sequence of type, variable and constant declarations

Forward Declarations

Due to Pascal-like (or Ada-like) semantics of PL/SQL, all functions and procedures must be declared before they are used in any statements. Thus, dealing with *mutually recursive* subprograms requires an extra bit of syntax: **forward declarations**.

A forward declaration for a procedure or a function declares the procedure/function name and its parameters (and the return type for the functions):

For procedures:

```
procedure <Name> [(<parameters>)];
```

For functions:

```
function <Name> [(<parameters>)] return <type>;
```

Note: Full function/procedure declaration must appear at some point below the forward declaration in the declaration section of an anonymous block.

Stored Functions and Procedures

Functions and procedures defined in anonymous blocks have as their scope only the anonymous block itself.

Stored Functions and **Stored Procedures** are **first-class database objects**, i.e., *just like database tables they are part of the* ***persistent database state***.

Stored functions can be used in:

- SQL statements;
- other stored functions/procedures;
- PL/SQL anonymous blocks and/or local functions/procedures;

- embedded SQL programs;
- database triggers (to be discussed in CPE 366.

Syntax. Similar to that of function and procedure declarations in anonymous blocks.

For procedures:

```
create  [or replace] procedure <Name> [(<parameters>)] as
   [<declarations>]
 begin
   <statements>;
   [exception  <statements>;]
 end;
```

For functions:

```
create [or replace]  function <Name> [(<parameters>)] return <type> as
   [<declarations>]
 begin
   <statements>;
   [exception  <statements>;]
 end;
```

Rules for calling stored functions from SQL statements. SQL semantics requires the following rules when stored functions are called from SQL statements:

- function has no `out` parameters;
- function is applicable to a row in the table;
- function returns a data type compatible with some SQL data type.

Return statement

`Retrun` statement can be found in the body of any subprogram (function or procedure).

For procedures, use `Return;`.

For functions, use `Return <expression>;`.

User-defined Packages

A **package** groups a number of declarations (types, constants, variables, functions, procedures) together and stores them persistently in on the Oracle's server.

A user-defined package consists of:

- Package header declarations: declarations of all constants, variables and types, as well as specifications (only the heads) for all functions and procedures.

- Package body declarations: code for all functions and procedures defined in the package. Also may include package initialization code.

Package header declarations

The syntax is

```
CREATE [OR REPLACE] package <Name> as

  <declarations>;

end;
```

Here,
- `<Name>` : name of the package. All objects defined in the package will be accessible through it.
- `<declarations>` : PL/SQL constant, variable, type declarations and heads of function and procedure declarations.

Note: If `OR REPLACE` key phrase is present, then the defined package replaces a package with the same name, is one exists in the database. If `OR REPLACE` is absent, and the package with the given name is already defined, the new package is not stored.

Package body declaraions

The syntax is

```
CREATE [OR REPLACE] package  body <Name> as

  <declarations>;

 [begin
    <statements>;

  end;]

end;
```

Here,
- `<Name>` : name of the package.
- `<declarations>` : full PL/SQL declarations of functions and procedures.
- `<statements>` : initialization code for the package.

Access to package components.

Access to components of the package is provided via the `<PackageName>.<Component>` or `<PackageName>.<Component>(<parameters>)` syntax.

Example Consider the following simple package header declaration:

```
create or replace package mineCSU as
  constant numCampuses binary_integer :=23;
  ourCampus binary_integer := 20;

  procedure rollUp_Enrollment();

  function largestEnrollment(campus in binary_integer)
           returns binary_integer;

  function campusName_to_Id(campusName in varchar)
            returns binary_integer;
```

We can now access components of this package as `mineCSU.numCampuses`, `mineCSU.rollup_Enrollment()`, `mineCSU.campusName_to_Id('Sonoma State University')`.