B

PL/SQL Functions in SQL Statements

PL/SQL Functions in SQL Statements

Using your own PL/SQL Function in a SQL Statement

SQL expressions can reference PL/SQL user-defined functions. Anywhere a built-in SQL function can be placed, a PL/SQL function can be placed as well.

Advantages

- Permit calculations that are too complex, awkward, or unavailable with SQL.
- Complex data analysis can be done within the Oracle Server, rather than retrieving the data into an application. This increases data independence.
- Increase efficiency of queries by performing functions in the query rather than in the application.
- Provide parallel query execution; if the query is parallelized, it can be executed in parallel (using the Parallel Query Option).
- Manipulate new types of data (for example, latitude and longitude) by encoding character strings and using functions to operate on the strings.

PL/SQL Functions in SQL Statements

Locations to call a PL/SQL Function

PL/SQL functions can be called from any SQL expression anywhere a built-in function can be called.

- Select list of a SELECT command
- Condition of the WHERE and HAVING clauses
- CONNECT BY, START WITH, ORDER BY, and GROUP BY clauses
- VALUES clause of the INSERT command
- SET clause of the UPDATE command

Note:

- Only stored functions are callable from SQL statements. PL/SQL procedures cannot be called.
- This feature is only available with PL/SQL 2.1. Tools using earlier versions of PL/SQL do not support this functionality.
- Arguments to a PL/SQL function called from a SQL statement must use the positional notations for arguments. The named notation is not supported.
- PL/SQL functions cannot be called from CHECK clause of a view.
- You must own or have EXECUTE privilege on the function in order to call the functions from a SQL statement.

PL/SQL Functions in SQL Statements

Meeting Basic Requirements

To be callable from SQL expressions, a user-defined PL/SQL function must meet certain basic requirements.

Requirements

- It must be a stored function, not a function defined within a PL/SQL block or subprogram.
- It must be a row function, not a column (group) function; that is, it cannot take an entire column of data as its argument.
- All of its formal arguments must be IN arguments; none can be an OUT or IN OUT argument.
- The datatypes of its formal arguments must be Oracle Server internal types such as CHAR, DATE or NUMBER, not PL/SQL types such as BOOLEAN, RECORD or TABLE.
- Its return type (the datatype of its result value) must be an Oracle Server internal type.

PL/SQL Functions in SQL Statements

Controlling Side Effects

To execute a SQL statement that calls a stored function, the Oracle Server must know if the functions is free of side effects. Side effects would be changes to database table or public packaged variables (those declared in a package specification.) Side effects could delay the executions of a query, yield order-dependent (therefore interminate) results, or require that package state be maintained across user sessions (which is not allowed). Therefore, the following restrictions apply to stored functions called from SQL expressions.

Restrictions

- The function cannot modify database tables; it cannot execute an INSERT, UPDATE or DELETE statement.
- Only local functions that are referenced in a SELECT list, VALUES clause of an INSERT statement, or SET clause of an UPDATE statement can update a variable defined in a package.
- Functions referenced in CONNECT BY, START WITH, ORDER BY or GROUP BY clauses cannot update package variables.
- Remote functions cannot read or write package state in that remote package. Every time the database link is accessed, a new session is established to the database.
- Functions that read or write package state cannot take advantage of the parallel query option.
- The function cannot call another subprogram that breaks one of the above restrictions.

Note:

- For stand-alone functions, Oracle can enforce these rules by checking the function body. However, the body of a packaged function is hidden; only the specification is visible. So, for packaged functions, you must use the ***pragma*** (compiler directive) **RESTRICT_REFERENCES** to enforce the rules.

Purity Level of a Packaged Function

Specify the purity level of a packaged function when creating the package. The purity level asserts the extent to which the function permits database operations. Specify the purity level with the **PRAGMA RESTRICT_REFERENCES** directive.

```
PRAGMA RESTRICT_REFERENCES (function_name
                            WNDS
                            [,WNPS]
                            [,RNDS]
                            [,RNPS] );
```

where:	WNDS	means 'writes no database state'. That is, the function cannot modify database tables.
	WNPS	means 'write no package state'. That is, the function cannot change the values of packaged variables.
	RNDS	means 'reads no database state'. That is, the function cannot query database tables.
	RNPS	means 'reads no package state'. That is, the function cannot reference the values of public packaged variables.

Note:

- Use RESTRICT_REFERENCES compiler directive to alert application developers of procedural code that violates restrictions of PL/SQL functions in SQL statements.
- The pragma tells the PL/SQL compiler to deny the packaged function read/write access to database tables, public packages or both. SQL statements that cause a function to violate the pragma result in a compilation error.
- The WNDS argument must be passed to the RESTRICT_REFERENCES pragma.

Example

Create a function, COMPOUND, in a package FINANCE. The function will be called from SQL statements on remote databases. Therefore you will need to assert the RNPS and WNPS purity levels because remote functions can neither read nor write package state when called from a SQL statement.

```
CREATE PACKAGE finance AS -- package specification
     interest REAL;        -- public packaged variable
. . .
     FUNCTION compound
          (years    IN NUMBER,
          amount    IN NUMBER,
          rate      IN NUMBER) RETURN NUMBER;
. . .
     PRAGMA RESTRICT_REFERENCES (compound, WNDS, RNPS,
WNPS);
END finance;

CREATE PACKAGE BODY finance AS -- package body
     FUNCTION compound
           (years   IN NUMBER,
           amount   IN NUMBER,
           rate     IN NUMBER) RETURN NUMBER IS
     BEGIN
       RETURN amount * POWER ((rate/100) + 1, years);
     END compound;
. . .
END finance;
```

Note:

- Assert the purity level of a function in the package specification.
- If a package contains multiple functions with the same name, the pragma applies to the last one

PL/SQL Functions in SQL Statements

Calling Packaged Functions

Call PL/SQL functions the same way you would call built-in SQL functions.

Example

Call the COMPOUND function (in the FINANCE package) from a SELECT statement within a PL/SQL block. The COMPOUND functions resides on a database in New York.

```
DECLARE
     interest  NUMBER;
BEGIN
. . .
     SELECT finance.compound@ny(yrs,amt,rte)
                    --function call
     INTO interest
     FROM accounts
     WHERE acctno = acct_id;
. . .
END;
```