

# Oracle PL/SQL FAQ

## Topics

- [What is PL/SQL and what is it used for?](#)
- [Should one use PL/SQL or Java to code procedures and triggers?](#)
- [How can one see if somebody modified any code?](#)
- [How can one search PL/SQL code for a string/key value?](#)
- [How can one keep a history of PL/SQL code changes?](#)
- [How can I protect my PL/SQL source code?](#)
- [Can one print to the screen from PL/SQL?](#)
- [Can one read/write files from PL/SQL?](#)
- [Can one call DDL statements from PL/SQL?](#)
- [Can one use dynamic SQL statements from PL/SQL?](#)
- [What is the difference between %TYPE and %ROWTYPE?](#)
- [What is the result of comparing NULL with NULL?](#)
- [How does one get the value of a sequence into a PL/SQL variable?](#)
- [Can one execute an operating system command from PL/SQL?](#)
- [How does one loop through tables in PL/SQL?](#)
- [How often should one COMMIT in a PL/SQL loop? / What is the best commit strategy?](#)
- [I can SELECT from SQL\\*Plus but not from PL/SQL. What is wrong?](#)
- [What is a mutating and constraining table?](#)
- [Can one pass an object/table as an argument to a remote procedure?](#)
- [Is it better to put code in triggers or procedures? What is the difference?](#)
- [Is there a PL/SQL Engine in SQL\\*Plus?](#)
- [Is there a limit on the size of a PL/SQL block?](#)
- [Where can one find more info about PL/SQL?](#)

---

[Back to Oracle FAQ Index](#)

---

## What is PL/SQL and what is it used for?

PL/SQL is Oracle's **P**rocedural **L**anguage extension to **SQL**. PL/SQL's language syntax, structure and data types are similar to that of [ADA](#). The PL/SQL language includes object oriented programming techniques such as encapsulation, function overloading, information hiding (all but inheritance). PL/SQL is commonly used to write data-centric programs to manipulate data in an Oracle database.

- [Back to top of file](#)
-

## Should one use PL/SQL or Java to code procedures and triggers?

Internally the Oracle database supports two procedural languages, namely PL/SQL and Java. This leads to questions like "Which of the two is the best?" and "Will Oracle ever desupport PL/SQL in favour of Java?".

Many Oracle applications are based on PL/SQL and it would be difficult of Oracle to ever desupport PL/SQL. In fact, all indications are that PL/SQL still has a bright future ahead of it. Many enhancements are still being made to PL/SQL. For example, Oracle 9iDB supports native compilation of PL/SQL code to binaries.

PL/SQL and Java appeal to different people in different job roles. The following table briefly describes the difference between these two language environments:

### PL/SQL:

- Data centric and tightly integrated into the database
- Proprietary to Oracle and difficult to port to other database systems
- Data manipulation is slightly faster in PL/SQL than in Java
- Easier to use than Java (depending on your background)

### Java:

- Open standard, not proprietary to Oracle
  - Incurs some data conversion overhead between the Database and Java type systems
  - Java is more difficult to use (depending on your background)
  - [Back to top of file](#)
- 

## How can one see if somebody modified any code?

Code for stored procedures, functions and packages is stored in the Oracle Data Dictionary. One can detect code changes by looking at the LAST\_DDL\_TIME column in the USER\_OBJECTS dictionary view.

Example:

```
SELECT OBJECT_NAME,
       TO_CHAR(CREATED,          'DD-Mon-RR HH24:MI') CREATE_TIME,
       TO_CHAR(LAST_DDL_TIME,    'DD-Mon-RR HH24:MI') MOD_TIME,
       STATUS
FROM   USER_OBJECTS
WHERE  LAST_DDL_TIME > '&CHECK_FROM_DATE';
```

- [Back to top of file](#)
- 

## How can one search PL/SQL code for a string/ key value?

The following query is handy if you want to know where a certain table, field or expression is referenced in your PL/SQL source code.

```
SELECT TYPE, NAME, LINE
FROM   USER_SOURCE
WHERE  UPPER(TEXT) LIKE '%&KEYWORD%';
```

- [Back to top of file](#)
-

## How can one keep a history of PL/SQL code changes?

One can build a history of PL/SQL code changes by setting up an AFTER CREATE schema (or database) level trigger (available from Oracle 8.1.7). This way one can easily revert to previous code should someone make any catastrophic changes. Look at this example:

```
CREATE TABLE SOURCE_HIST                                -- Create history
table                                                    table
    AS SELECT SYSDATE CHANGE_DATE, USER_SOURCE.*
    FROM    USER_SOURCE WHERE 1=2;

CREATE OR REPLACE TRIGGER change_hist                    -- Store code in
hist table                                              hist table
    AFTER CREATE ON SCOTT.SCHEMA                        -- Change SCOTT to
your schema name
DECLARE
BEGIN
    if DICTIONARY_OBJ_TYPE in ('PROCEDURE', 'FUNCTION',
                              'PACKAGE', 'PACKAGE BODY', 'TYPE') then
        -- Store old code in SOURCE_HIST table
        INSERT INTO SOURCE_HIST
            SELECT sysdate, user_source.* FROM USER_SOURCE
            WHERE  TYPE = DICTIONARY_OBJ_TYPE
            AND    NAME = DICTIONARY_OBJ_NAME;
    end if;
EXCEPTION
    WHEN OTHERS THEN
        raise_application_error(-20000, SQLERRM);
END;
/
show errors
```

- [Back to top of file](#)
- 

## How can I protect my PL/SQL source code?

PL/SQL V2.2, available with Oracle7.2, implements a binary wrapper for PL/SQL programs to protect the source code.

This is done via a standalone utility that transforms the PL/SQL source code into portable binary object code (somewhat larger than the original). This way you can distribute software without having to worry about exposing your proprietary algorithms and methods. SQL\*Plus and SQL\*DBA will still understand and know how to execute such scripts. Just be careful, there is no "decode" command available.

The syntax is:

```
wrap iname=myscript.sql oname=xxxx.plb
```

- [Back to top of file](#)
- 

## Can one print to the screen from PL/SQL?

One can use the DBMS\_OUTPUT package to write information to an output buffer. This buffer can be displayed on the screen from SQL\*Plus if you issue the *SET SERVEROUTPUT ON*; command. For example:

```
set serveroutput on
begin
    dbms_output.put_line('Look Ma, I can print from PL/SQL!!!');
end;
/
```

DBMS\_OUTPUT is useful for debugging PL/SQL programs. However, if you print too much, the output buffer will overflow. In that case, set the buffer size to a larger value, eg.: set serveroutput on size 200000

If you forget to set serveroutput on type *SET SERVEROUTPUT ON* once you remember, and then *EXEC NULL*; . If you haven't cleared the DBMS\_OUTPUT buffer with the disable or enable procedure, SQL\*Plus will display the entire contents of the buffer when it executes this dummy PL/SQL block.

- [Back to top of file](#)
- 

## Can one read/write files from PL/SQL?

Included in Oracle 7.3 is an UTL\_FILE package that can read and write operating system files. The directory you intend writing to has to be in your INIT.ORA file (see UTL\_FILE\_DIR=... parameter). Before Oracle 7.3 the only means of writing a file was to use DBMS\_OUTPUT with the SQL\*Plus SPOOL command.

Copy this example to get started:

```
DECLARE
    fileHandler UTL_FILE.FILE_TYPE;
BEGIN
    fileHandler := UTL_FILE.FOPEN('/tmp', 'myfile', 'w');
    UTL_FILE.PUTF(fileHandler, 'Look ma, I'm writing to a
file!!!\n');
    UTL_FILE.FCLOSE(fileHandler);
EXCEPTION
    WHEN utl_file.invalid_path THEN
        raise_application_error(-20000, 'ERROR: Invalid path for
file or path not in INIT.ORA.');
```

- [Back to top of file](#)
- 

## Can one call DDL statements from PL/SQL?

One can call DDL statements like CREATE, DROP, TRUNCATE, etc. from PL/SQL by using the "EXECUTE IMMEDIATE" statement. Users running Oracle versions below 8i can look at the DBMS\_SQL package (see FAQ about Dynamic SQL).

```
begin
    EXECUTE IMMEDIATE 'CREATE TABLE X(A DATE)';
end;
```

NOTE: The DDL statement in quotes should not be terminated with a semicolon.

- [Back to top of file](#)
- 

## Can one use dynamic SQL statements from PL/SQL?

Starting from Oracle8i one can use the "EXECUTE IMMEDIATE" statement to execute dynamic SQL and PL/SQL statements (statements created at run-time). Look at these examples. Note that statements are NOT terminated by semicolons:

```
EXECUTE IMMEDIATE 'CREATE TABLE x (a NUMBER)';

-- Using bind variables...
sql_stmt := 'INSERT INTO dept VALUES (:1, :2, :3)';
EXECUTE IMMEDIATE sql_stmt USING dept_id, dept_name, location;

-- Returning a cursor...
sql_stmt := 'SELECT * FROM emp WHERE empno = :id';
EXECUTE IMMEDIATE sql_stmt INTO emp_rec USING emp_id;
```

One can also use the older DBMS\_SQL package (V2.1 and above) to execute dynamic statements. Look at these examples:

```
CREATE OR REPLACE PROCEDURE DYNSQL AS
  cur integer;
  rc integer;
BEGIN
  cur := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(cur, 'CREATE TABLE X (Y DATE)',
DBMS_SQL.NATIVE);
  rc := DBMS_SQL.EXECUTE(cur);
  DBMS_SQL.CLOSE_CURSOR(cur);
END;
/
```

More complex DBMS\_SQL example using bind variables:

```
CREATE OR REPLACE PROCEDURE DEPARTMENTS(NO IN DEPT.DEPTNO%TYPE)
AS
  v_cursor integer;
  v_dname char(20);
  v_rows integer;
BEGIN
  v_cursor := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(v_cursor, 'select dname from dept where deptno
> :x', DBMS_SQL.V7);
  DBMS_SQL.BIND_VARIABLE(v_cursor, ':x', no);
  DBMS_SQL.DEFINE_COLUMN_CHAR(v_cursor, 1, v_dname, 20);
  v_rows := DBMS_SQL.EXECUTE(v_cursor);
  loop
    if DBMS_SQL.FETCH_ROWS(v_cursor) = 0 then
      exit;
    end if;
    DBMS_SQL.COLUMN_VALUE_CHAR(v_cursor, 1, v_dname);
    DBMS_OUTPUT.PUT_LINE('Department name: '||v_dname);
  end loop;
  DBMS_SQL.CLOSE_CURSOR(v_cursor);
EXCEPTION
  when others then
    DBMS_SQL.CLOSE_CURSOR(v_cursor);
    raise_application_error(-20000, 'Unknown Exception
Raised: '||sqlcode||' '||sqlerrm);
```

```
END;  
/
```

- [Back to top of file](#)
- 

## What is the difference between %TYPE and %ROWTYPE?

The %TYPE and %ROWTYPE constructs provide data independence, reduces maintenance costs, and allows programs to adapt as the database changes to meet new business needs.

%ROWTYPE is used to declare a *record* with the same types as found in the specified database table, view or cursor. Example:

```
DECLARE  
    v_EmpRecord emp%ROWTYPE;
```

%TYPE is used to declare a *field* with the same type as that of a specified table's column. Example:

```
DECLARE  
    v_EmpNo emp.empno%TYPE;
```

- [Back to top of file](#)
- 

## What is the result of comparing NULL with NULL?

NULL is neither equal to NULL, nor it is not equal to NULL. Any comparison to NULL is evaluated to NULL. Look at this code example to convince yourself.

```
declare  
    a number := NULL;  
    b number := NULL;  
begin  
    if a=b then  
        dbms_output.put_line('True, NULL = NULL');  
    elsif a<>b then  
        dbms_output.put_line('False, NULL <> NULL');  
    else  
        dbms_output.put_line('Undefined NULL is neither = nor <> to  
NULL');  
    end if;  
end;
```

- [Back to top of file](#)
- 

## How does one get the value of a sequence into a PL/SQL variable?

As you might know, one cannot use sequences directly from PL/SQL. Oracle (for some silly reason) prohibits this:

```
i := sq_sequence.NEXTVAL;
```

However, one can use embedded SQL statements to obtain sequence values:

```
select sq_sequence.NEXTVAL into :i from dual;
```

- [Back to top of file](#)
- 

## Can one execute an operating system command from PL/SQL?

There is no direct way to execute operating system commands from PL/SQL in **Oracle7**. However, one can write an external program (using one of the precompiler languages, OCI or Perl with Oracle access modules) to act as a listener on a database pipe (SYS.DBMS\_PIPE). Your PL/SQL program then put requests to run commands in the pipe, the listener picks it up and run the requests. Results are passed back on a different database pipe. For an Pro\*C example, see chapter 8 of the Oracle Application Developers Guide.

In **Oracle8** one can call external 3GL code in a dynamically linked library (DLL or shared object). One just write a library in C/ C++ to do whatever is required. Defining this C/C++ function to PL/SQL makes it executable. Look at this [External Procedure](#) example.

- [Back to top of file](#)
- 

## How does one loop through tables in PL/SQL?

Look at the following **nested loop** code example.

```
DECLARE
    CURSOR dept_cur IS
        SELECT deptno
          FROM dept
         ORDER BY deptno;
    -- Employee cursor all employees for a dept number
    CURSOR emp_cur (v_dept_no DEPT.DEPTNO%TYPE) IS
        SELECT ename
          FROM emp
         WHERE deptno = v_dept_no;
BEGIN
    FOR dept_rec IN dept_cur LOOP
        dbms_output.put_line('Employees in Department
'||TO_CHAR(dept_rec.deptno));
        FOR emp_rec IN emp_cur(dept_rec.deptno) LOOP
            dbms_output.put_line('...Employee is '||emp_rec.ename);
        END LOOP;
    END LOOP;
END;
/
```

- [Back to top of file](#)
- 

## How often should one COMMIT in a PL/SQL loop? / What is the best commit strategy?

Contrary to popular believe, one should **COMMIT less frequently** within a PL/SQL loop to prevent ORA-1555 (Snapshot too old) errors. The higher the frequency of commit, the sooner the extents in the rollback segments will be cleared for new transactions, causing ORA-1555 errors.

To fix this problem one can easily rewrite code like this:

```

        FOR records IN my_cursor LOOP
            ...do some stuff...
            COMMIT;
        END LOOP;
... to ...
        FOR records IN my_cursor LOOP
            ...do some stuff...
            i := i+1;
            IF mod(i, 10000) THEN      -- Commit every 10000 records
                COMMIT;
            END IF;
        END LOOP;

```

If you still get ORA-1555 errors, contact your DBA to increase the rollback segments.

**NOTE:** Although fetching across COMMITs work with Oracle, is not supported by the ANSI standard.

- [Back to top of file](#)
- 

## I can SELECT from SQL\*Plus but not from PL/SQL. What is wrong?

PL/SQL respect object privileges given directly to the user, but does not observe privileges given through roles. The consequence is that a SQL statement can work in SQL\*Plus, but will give an error in PL/SQL. Choose one of the following solutions:

- Grant **direct** access on the tables to your user. Do not use roles!
  - `GRANT select ON scott.emp TO my_user;`
  - Define your procedures with invoker rights (Oracle 8i and higher);
  - Move all the tables to one user/schema.
- [Back to top of file](#)
- 

## What is a mutating and constraining table?

"Mutating" means "changing". A mutating table is a table that is currently being modified by an update, delete, or insert statement. When a trigger tries to reference a table that is in state of flux (being changed), it is considered "mutating" and raises an error since Oracle should not return data that has not yet reached its final state.

Another way this error can occur is if the trigger has statements to change the primary, foreign or unique key columns of the table off which it fires. If you must have triggers on tables that have referential constraints, the workaround is to enforce the referential integrity through triggers as well.

There are several restrictions in Oracle regarding triggers:

- A row-level trigger cannot query or modify a mutating table. (Of course, NEW and OLD still can be accessed by the trigger) .
- A statement-level trigger cannot query or modify a mutating table if the trigger is fired as the result of a CASCADE delete.



- Etc.
  - [Back to top of file](#)
- 

## Can one pass an object/table as an argument to a remote procedure?

The only way the same object type can be referenced between two databases is via a database link. Note that it is not enough to just use the same type definitions. Look at this example:

```
-- Database A: receives a PL/SQL table from database B
CREATE OR REPLACE PROCEDURE pcalled(TabX DBMS_SQL.VARCHAR2S) IS
BEGIN
    -- do something with TabX from database B
    null;
END;
/

-- Database B: sends a PL/SQL table to database A
CREATE OR REPLACE PROCEDURE pcalling IS
    TabX DBMS_SQL.VARCHAR2S@DBLINK2;
BEGIN
    pcalled@DBLINK2 (TabX) ;
END;
/
```

- [Back to top of file](#)
- 

## Is it better to put code in triggers or procedures? What is the difference?

In earlier releases of Oracle it was better to put as much code as possible in procedures rather than triggers. At that stage procedures executed faster than triggers as triggers had to be re-compiled every time before executed (unless cached). In more recent releases both triggers and procedures are compiled when created (stored p-code) and one can add as much code as one likes in either procedures or triggers.

- [Back to top of file](#)
- 

## Is there a PL/SQL Engine in SQL\*Plus?

No. Unlike Oracle Forms, SQL\*Plus does not have an embedded PL/SQL engine. Thus, all your PL/SQL code is sent directly to the database engine for execution. This makes it much more efficient as SQL statements are not stripped off and sent to the database individually.

- [Back to top of file](#)
- 

## Is there a limit on the size of a PL/SQL block?

Yes, the max size is not an explicit byte limit, but related to the parse tree that is created when you compile the code. You can run the following select statement to query the size of an existing package or procedure:

```
SQL> select * from dba_object_size where name =
'procedure_name';
```

- [Back to top of file](#)
-

## Where can one find more info about PL/SQL?

- [Oracle FAQ: PL/SQL code examples](#)
  - [Oracle FAQ: PL/SQL Books](#)
  - [PLNet.org](#) - An open source repository for PL/SQL developers
  - [RevealNet PL/SQL Pipeline](#) - A free community for Oracle developers worldwide
  - [The PL/SQL Cellar](#) - Free Oracle PL/SQL scripts including a bitwise operations package and message digest algorithms
  - [PLSolutions.com](#) - PL/Solutions provides consulting and training services for the Oracle PL/SQL language and PL/Vision
  - [The DMOZ PL/SQL Directory](#)
- 
- [Back to top of file](#)