TOO_MANY_ROWS	ORA-01422	Single row SELECT returned more than one row
NO_DATA_FOUND	ORA-01403	Single row SELECT returned no data.
DUP_VAL_ON_INDEX	ORA-00001	Attempt to insert a duplicate value.
PROGRAM_ERROR	ORA-06501	PL/SQL experienced an internal error.
ZERO_DIVIDE	ORA-01476	Attempt to divide by zero

```
DECLARE
        v_lname VARCHAR2(20)
BEGIN
        SELECT last_name INTO v_lname
        FROM  emp WHERE first_name = 'Chris';
EXCEPTION
        WHEN TOO_MANY_ROWS THEN
DBMS_OUTPUT.PUT_LINE ('Your query returned
more than a single row into a scalar variable');
        WHEN OTHERS THEN…
END;
```

Note the WHEN OTHERS THEN… clause. This enables you to trap unexpected errors and thus avoid abnormal program termination.

Non-predefined Server Error:

You name the standard ORA-xxxxx errors and then refer to them in the exception section as such. Uses a PRAGMA clause to pass the directive to the compiler. Note the SQLERRM reference in the EXCEPTION SECTION – This prints the standard ORA-error message. (This demonstration is for showing sysntac and basic usage. Oracle already provide's a NULL insert error.)

```
DECLARE
        null_insert EXCEPTION;
        PRAGMA EXCEPTION_INIT
          null_insert, -01400);
BEGIN
        INSERT INTO emp VALUES (NULL, 'Plum');
EXCEPTION
        WHEN null_insert THEN
DBMS_OUTPUT.PUT_LINE ('NULL insert not allowed.);
DBMS_OUTPUT.PUT_LINE (SQLERRM);
END
```

RAISE_APPLICATION_ERROR:

Allows you to determine the error condition and RAISE the exception based on logic inside your program. The user defined errors are returned to the user/app in a manner consistent with ORA-errors. You assignan ORA-error between -20000 and -20999. Note use of SQL%NOTFOUND. This is an implicit cursor attribute that returns TRUE/FALSE for the immediately preceding SQL statement. (Oracle already provide's a standard NO_DATA_FOUND error.)

```
RAISE_APPLICATION_ERROR (err-num, message)
```

```
BEGIN
        DELETE FROM emp WHERE sal > 1000000
EXCEPTION
        WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR (-20201, 'No one in
that salary range.);
END;
```

PTI
Perpetual
Technologies, Inc.

Perpetual Technologies, Inc. (PTI) provides mission-critical database and information systems support to commercial and government enterprises worldwide. Focused on improving performance and lowering costs, our subject-matter experts plan, design, develop, deploy, and manage Oracle database environments running on UNIX and Windows platforms. Perpetual Technologies strives to create tailored, flexible IT solutions in the areas of Oracle database 8i, 9i, 10g, Oracle RAC, capacity planning, disaster recovery planning, performance tuning, Oracle Application Server , Oracle content manager, Oracle database design, complete or supplemental remote Oracle database administration, afterhours DBA coverage, and Oracle database vacation support.

PTI
Perpetual
Technologies, Inc.

9155 Harrison Park Court
Indianapolis, IN 46216
800-538-0453
www.perptech.com

PTI
Perpetual
Technologies, Inc.

ORACLE PL/SQL

Quick Reference Guide
Beginning User

*Service Excellence
for the Data Driven Enterprise.*

Oracle PL/SQL QUICK GUIDE

FOR THE BEGINNING USER

PL/SQL is a 3GL programming language proprietary to Oracle that offers procedural extensions to SQL. It uses embedded SQL to access and manipulate data in the database. The following provides some PL/SQL fundamentals to the beginning user.

PL/SQL MODULARIZED BLOCK STRUCTURE

```
DECLARE          --declarative section

        (Declarations of scalar and composite variables, constants,
        cursors, types, exceptions)

BEGIN            --executable section

        (Embedded SQL statements, PL/SQL conditional IF/THEN logic,
        LOOPs, variable assignments, nested blocks. May be nested to
        many levels.)

EXCEPTION        --exception section

        (Error handling routines to "trap" errors)

END;    --ends the executable section
```

BASIC BUILDING BLOCKS

Common Delimiters and their purpose:

= != < >	Comparison operators: equal to, not-equal to, greater than, less than
:=	Variable or constant assignment operator
:	Colon - precedes a bind variable (x =:*var*)
;	Semi-colon - Termminates SQL and PL/SQL statements
||	Concatenation operator
+ - * /	Arithmetic operators
--	2 hyphens: Denotes a comment to the end of the line
/* */	Denotes a multi-line comment

Scalar Variables: (Composite variables are not discussed here)

Named with an identifier and datatype, and optionally initialized, in the declarative section. Assigned and re-assigned values in the executable section. Visible to immediate and nested blocks only.

Constants:

Named with an identifier and MUST be initialized in the declarative section. Assigned value will not change during execution.

```
identifier [CONSTANT] DATATYPE [NOT NULL]
[:= | DEFAULT expression];
```

```
DECLARE
        v_fname VARCHAR2(20) :=  'Smith';
        c_tax NUMBER := .08;
        v_saledate DATE;
        …
```

Embedded SQL:

In order to work with data from the database, it must be retrieved into local variables, and any changes are then sent back to the database. Note the SELECT that retrieves a single row, single column value INTO the local scalar variable. This allows us to work with database data inside of our PL/SQL.

```
DECLARE
        v_ph NUMBER;
BEGIN
        SELECT phone_num INTO v_ph FROM emp
        WHERE last_name='Smith';
        v_ph := 9999999999;
        UPDATE emp SET phone_num=v_ph
        WHERE last_name='Smith';
END;
```

CONDITIONAL CONTROL

IF/THEN Logic and CASE expressions:

Enables conditional control of statement execution. The first statement that evaluates to true is executed and the remaining statements are skipped. Place the most likely true statements first to avoid unnecessary evaluation.

```
DECLARE
        x NUMBER :=10;
BEGIN
        IF x = 5 THEN…;
          ELSIF x = 20 THEN…;
          ELSE…;
        END IF;
END;
```

CASE Expressions:

```
DECLARE
        X NUMBER := 10;
BEGIN
  y := CASE x
        WHEN 5 THEN…;
        WHEN 20 THEN…;
        END;
END;
```

ITERATIVE CONTROL WITH LOOPS

LOOPs allow us to repeat a statement or set of statements over and over. The number of iterations can be either pre-determined or based upon some condition.

Basic LOOP:

Loop executes at least once before the EXIT WHEN condition is evaluated.

```
DECLARE
        x NUMBER := 1;
BEGIN
        LOOP
          statement1;
          statement2;
          x := x+1;
        EXIT WHEN x > 10;
        END LOOP;
END;
```

WHILE LOOP :

Loop will only execute until condition evaluates to true. If the condition immediately evaluates to true, the loop will not execute even once.

```
DECLARE
        x NUMBER := 1;
BEGIN
        WHILE x <= 10 LOOP
          statement1;
          statement2;
        x := x+1;
        END LOOP;
END;
```

FOR LOOP:

Loop will execute a pre-defined number of times. Lower and upper bounds must evaluate to an integer. (lower_bound..upper_bound) Counter identifier is implicitly declared and has is visible only within the context of the loop.

```
DECLARE
        x NUMBER := 10;
BEGIN
        FOR i IN 1..x LOOP
          statement1;
          statement2;
          i = i+1;
        END LOOP;
END;
```

EXCEPTION HANDLING

Exception handling is a powerful PL/SQL feature that lets the developer decide what to do when an error occurs in the program. The error might be an ORA-xxxxx server error or one that the programmer defines based on certain business rules. When an exception is handled it is "trapped" and execution continues in the enclosing block or calling procedure. If an exception is not trapped, the program will terminate abruptly and the error will propagate to the calling environment.

Predefined Oracle server errors (Partial list, est 20 available):