Introduzione al linguaggio
PL/SQL (Procedural Language for SQL)

Disclaimer

Commenti, suggerimenti e critiche sono benvenuti: mail a rossati@libero.it

Indice generale

1 Premessa

1.1 Caratteristica della documentazione

Il presente è un manuale introduttivo di PL/SQL per ORACLE 8, non è, quindi, un manuale di riferimento del linguaggio.

1.2 Convenzioni usate

Nel manuale sono indicate in maiuscolo le parole di PL/SQL; qui di seguito un breve elenco di parole che indicano elementi dei comandi di PL/SQL.

EspressioneBooleana	condizione che può valere vero o falso
istruzione(i)	una o più istruzioni PL/SQL
Tipo	Qualsiasi tipo di dato PL/SQL

Nelle espressioni della sintassi dei comandi, le parti opzionali saranno racchiuse fra parentesi quadre. Parti in alternativa saranno racchiuse fra parentesi graffe, separate dal carattere |.

2 Generalità sul linguaggio

PL/SQL è il linguaggio di programmazione di ORACLE che estende le capacità di elaborazione di SQL[1].

I programmi PL/SQL sono divisi in blocchi composti di tre parti:

- Dichiarativa — opzionale, contiene la dichiarazione delle variabili, costanti e funzioni
- Eseguibile — obbligatoria, contiene le istruzioni
- gestione delle eccezioni — opzionale

La parte dichiarativa inizia con la parola chiave **DECLARE**
La parte eseguibile inizia con la parola chiave **BEGIN** e termina con la parola chiave **END;**
La parte di gestione delle eccezioni inizia con la parola chiave **EXCEPTION** e termina con la parola chiave **END;**

Qui di seguito vi è un "canovaccio" di programma PLS/QL

```
DECLARE
-- qui si dichiarano oggetti, variabili e costanti utilizzate nel programma
BEGIN
-- qui si scrive il programma PLSQL
END;
EXCEPTION
-- qui si scrive l'eventuale programma di gestione di errori ed eccezioni
END;
```

3 Definizione delle variabili e delle costanti

Gli identificatori delle variabili sono costituite da una lettera eventualmente seguita da altre lettere, numeri, segni di dollaro, trattoni (underscore). Altri caratteri quali meno, barre e spazi sono illegali.

PL/SQL non fa distinzione fra maiuscolo e minuscolo, eccetto se inserite in stringhe di testo fra apici.

3.1 Variabili

Sintassi: NomeVariabile Tipo [:= *valore di default*];

3.1.1 Tipi BINARY_INTEGER

Memorizza numeri compresi fra -2147483647 e 2147483647. I seguenti tipi sono sottotipi di BINARY_INTEGER.

- NATURAL, NATURALN (0 .. 2147483647)

[1] Sono accettati tutti i comandi di trattamento dei dati (eccetto EXPLAIN PLAN), i comandi per il controllo delle transazioni, le funzioni, le pseudocolonne, e gli operatori. PL/SQL non consente i comandi di definizione di dati come CREATE, i comandi di controllo della sessione come SET ROLE, o i comandi di controllo del sistema come ALTER SYSTEM.

- POSITIVE, POSITIVEN (1 .. 2147483647)

3.1.2 Tipi NUMBER

Sintassi: NomeVariabile NUMBER(precisione, scala);

Esempio `part_no NUMBER(4);`

dove precisione è il numero di cifre esatte, scala se positivo i decimali, se negativo è il troncamento di –scala cifre.

I seguenti sono sinonimi del formato dati NUMBER.

DEC, DECIMAL, DOUBLE PRECISION, INTEGER, INT, NUMERIC, REAL, SMALLINT, FLOAT
Nota Nel Tipo FLOAT non si deve indicare precisione e scala.

3.1.3 Tipo PLS_INTEGER

Come BINARY INTEGER. Occupano meno memoria dei NUMBER, e le operazioni su di essi utilizzano le istruzioni di macchina piuttosto che librerie matematiche, risultando quindi più veloci.

3.1.4 Tipi CARATTERI

Per memorizzare stringhe di caratteri. Esistono i seguenti sottotipi:

- CHAR o CHARACTER ***sintassi***: NomeVariabile CHAR(n) Dove n è un numero fra 1 e 32767, il default è 1.
- LONG per stringhe di lunghezza variabile fino ad una lunghezza di 32760
- ROW ***sintassi***: NomeVariabile ROW(n) Dove n è un numero fra 1 e 32767 che indica la lunghezza massima
- LONG ROW come LONG
- ROWID per memorizzare il ROWID di una riga di tabella, cioè il suo indirizzo fisico
- VARCHAR2 ***sintassi***: NomeVariabile VARCHAR2(n) Dove n è un numero fra 1 e 32767 che indica la lunghezza massima
- STRING, VARCHAR sinonimi di VARCHAR2
- Per gli alfabeti che richiedono due byte per codificare un carattere i seguenti tipi: NCHAR, NVARCHAR2, sono i corrispondenti di CHAR e VARCHAR2.

3.1.5 Tipi Large Object (LOB)

BFILE, BLOB, CLOB, and NCLOB

3.1.6 Altri Tipi di dato

- BOOLEAN
- DATE
- MLSLABEL

Esempio `in_stock BOOLEAN;`

3.2 Attributi

3.2.1 %TYPE

sintassi: NomeVariabile%TYPE
esempio:
```
credito REAL(7,2);
debito credit%TYPE;
```

la variabile *debito* è stata dichiarata dello stesso tipo della variabile *credito*.

3.2.2 %ROWTYPE

sintassi: NomeRecord NomeTabella%ROWTYPE
esempio:
```
emp_rec emp%ROWTYPE;
……..
IF emp_rec.deptno = 20 THEN ...
emp_rec.ename := 'JOHNSON';
emp_rec.sal := emp_rec.sal * 1.15;
```

la prima riga dell'esempio associa ad una struttura record la stessa struttura (tipi e nomi) delle righe della tabella *emp*. Le righe successive elaborano in PL/SQL i campi di detto record.

3.2.3 Sottotipi definiti dall'utilizzatore

E' possibile definire propri sottotipi

Sintassi: SUBTYPE NomeMioTipo IS Tipo;

Esempi
```
DECLARE
temp VARCHAR2(15);
SUBTYPE Word IS temp%TYPE; -- maximum size of Word is 15
```

3.3 Costanti

Sintassi: NomeVariabile CONSTANT Tipo := Valore;

Esempio: `credito_limite CONSTANT REAL := 5000.00;`

3.4 Funzioni predefinite

Errori SQLCODE, SQLERRM

Numeriche ABS, ACOS, ASIN, ATAN, ATAN2, CEIL, COS, COSH, EXP, FLOOR, LN, LOG, MOD, POWER, ROUND, SIGN, SIN, SINH, SQRT, TAN, TANH, TRUNC

Carattere ASCII, CHR, CONCAT, INITCAP, INSTR, INSTRB, LENGTH, LENGTHB, LOWER, LPAD, LTRIM, NLS_INITCAP, NLS_LOWER, NLS_UPPER, NLSSORT, REPLACE, RPAD, RTRIM, SOUNDEX, SUBSTR, SUBSTRB

Conversione CHARTOROWID, CONVERT, HEXTORAW, NLS_CHARSET_ID, NLS_CHARSET_NAME, RAWTOHEX, ROWIDTOCHAR, TO_CHAR, TO_DATE, TO_LABEL, TO_MULTI_BYTE, TO_NUMBER, TO_SINGLE_BYTE

Date ADD_MONTHS, LAST_DAY, MONTHS_BETWEEN, NEW_TIME, NEXT_DAY, ROUND, SYSDATE, TRUNC

Miscellanea DECODE, DUMP, GREATEST, GREATEST_LB, LEAST, LEAST_UB, NVL, UID, USER, USERENV, VSIZE

Esempi: TO_DATE, TO_NUMBER, CONCAT, SUBSTR (v.7.1.2).

3.5 Strutture Composte

3.5.1 Records

Sintassi: TYPE NomeDiTipo IS RECORD (DichiarazioneDiCampo [,DichiarazioneDiCampo]…);

Note DichiarazioneDiCampo: NomeDiCampo Tipo [[NOT NULL] {:= | DEFAULT} espressione]

Esempio
```
TYPE Record IS RECORD (
Rempno NUMBER(4),
REname VARCHAR2(10),
Rjob   VARCHAR2(9),
RSal   NUMBER(7,2));
My_record Record; -- My_record è una struttura del tipo Record
```

3.5.2 Tabelle (Nested Tables)

Sintassi: TYPE NomeDiTipo IS TABLE OF Tipo [NOT NULL]
INDEX BY BINARY_INTEGER;

Attributi: COUNT, EXIST, FIRST, LAST, PRIOR, NEXT, DELETE (v. Par. 3.6.2)

Note Le *Table* sono trattate come matrici, infatti ogni elemento ha un unico indice numerico che lo individua; tuttavia questo non è limitato, e può non essere formato da numeri consecutivi: ciò permette l'inserzione di nuovi elementi e la cancellazione, di fatto è una vera e propria chiave di accesso.
Le *Table* di PL/SQL non corrispondono alle TABLE di ORACLE, in quanto sono formate da una sola colonna, tuttavia questa può contenere strutture anche complesse, ad esempio *Records*.

3.5.3 Cursori

I Cursori sono aree di memoria in cui Oracle memorizza i risultati di una query SQL. Se il risultato è una sola riga, il cursore è definito implicitamente, altrimenti va dichiarato esplicitamente.

Sintassi: CURSOR NomeDiCursore IS SELECT… [FOR UPDATE [NomeColonna] [NOWAIT]];

Esempio
```
CURSOR c1 IS SELECT empno, sal FROM emp
WHERE job = 'SALESMAN' AND comm > sal FOR UPDATE NOWAIT;
```

3.5.4 Esempio di utilizzo CURSOR e TABLES

```
DECLARE
   TYPE TipoTabella IS Table OF emp%ROWTYPE;    -- tabella che conterra'
   Tabella TipoTabella := TipoTabella();        -- righe della TABLE emp
   CURSOR Cursore IS SELECT * from emp;         -- cursore per scorrere TABLE emp
   I INTEGER := 0;
BEGIN
-- porto nella Tabella le righe della TABLE emp
```

```
  FOR Indx IN Cursore LOOP
      I := I + 1;
      Tabella.EXTEND(1);
      Tabella(I) := Indx;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('Numero records: ' || Tabella.COUNT);
  FOR Indx IN 1..Tabella.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(Tabella(Indx).Ename);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('Elimino record di: ' || Tabella(5).ename);
  DBMS_OUTPUT.PUT_LINE('Elimino record di: ' || Tabella(6).ename);
  Tabella.DELETE(5,6);
  I := Tabella.FIRST;  -- vado all'inizio
  DBMS_OUTPUT.PUT_LINE('Numero records dopo delete: ' || Tabella.COUNT);
  WHILE I IS NOT NULL LOOP
     DBMS_OUTPUT.PUT_LINE(Tabella(I).Ename);
     I := Tabella.NEXT(I);
  END LOOP;
END;
/
```

3.6 Collezioni e Oggetti[2]

Le collezioni sono gruppi ordinati di elementi dello stesso tipo. Ogni elemento è individuato da un numero univoco. ORACLE mette a disposizione due tipi di variabili Collezione: le Matrici (*Varrays*) e le Tabelle(*Tables*). Gli Oggetti sono simili alle Collezioni, ma con la caratteristica di poter definire metodi e proprietà .
L'utilizzo degli oggetti è possibile in ORACLE8 *Enterprise Edition* con la *Objects Option*.
Collezione (e Oggetti) devono essere definiti, successivamente possono essere usati per definire variabili di detto tipo. A questo punto la Collezione è vuota, per assegnare dei valori ad essa occorre utilizzare appositi *metodi*.

3.6.1 Matrici unidimensionali (Varrays)

Sintassi: TYPE NomeDiTipo {IS VARRAY | VARYING ARRAY} (Limite) OF Tipo [NOT NULL];
Note Il Tipo non può essere: BINARY_INTEGER, BOOLEAN, LONG, LONG RAW, NATURAL, NATURALN, NCHAR, NCLOB, NVARCHAR2, tipi object con TABLE o attributi VARRAY, PLS_INTEGER, POSITIVE, POSITIVEN.
Esempio
```
TYPE Calendar IS VARRAY(366) OF DATE;
TYPE Array IS VARRAY (10) OF Number;
```

3.6.2 Metodi delle Collezioni

- **`Costruttore`**
  Il costruttore permette di inizializzare una collezione e deve essere sempre richiamato o nel blocco DECLARE o nel Blocco programma. Il nome del metodo *Costruttore* corrisponde al nome assegnato al TYPE.
  Sintassi: NomeVariabile NomeDiTipo := NomeDiTipo(ValoriIniziale$_1$, ValoriIniziale$_2$,…);
  NomeVariabile NomeDiTipo := NomeDiTipo();
  Note: La seconda forma crea la variabile vuota
  Esempio:
  ```
  Matrix Array :=Array(0);      -- in blocco DECLARE
         Matrix :=Array(0);           -- in blocco programma
  ```
- **`EXTEND`**
  Questo metodo aumenta l'ampiezza della collezione.
  Sintassi: NomeVariabile.EXTEND(N); -- Aggiunge alla Collezione N elementi nulli
  NomeVariabile.EXTEND(N,M); -- Aggiunge alla Collezione N elementi = all' M-esimo elemento
  Esempio:
  ```
  Matrix.EXTEND(9,1);           -- in blocco programma
  ```
- **`DELETE E TRIM`**
  Sono due metodi simili, entrambi eliminano uno o più elementi da una Collezione.
  Sintassi: NomeVariabile.TRIM; -- Elimina l'ultimo elemento dalla Collezione
  NomeVariabile.TRIM(N); -- Elimina gli ultimi N elementi dalla Collezione
  NomeVariabile.DELETE; -- Elimina tutti gli elementi della Collezione
  NomeVariabile.DELETE(N); -- Elimina l'elemento N-esimo della Collezione
  NomeVariabile.DELETE(N,M); -- Elimina gli elementi da N ad M della Collezione
- **`EXIST`**
  Sintassi: NomeVariabile.EXIST(N); -- risponde TRUE se esiste l'N-esimo elemento, altrimenti FALSE

[2] Collezione e Oggetti, tranne le Table, sono disponibili da ORACLE 8

Esempio: `IF Matrix.EXSIST(9) THEN`
`Matrix(9) := 3.14159;`

- **COUNT E LIMIT**
  Sintassi: NomeVariabile.COUNT; -- fornisce il numero di elementi della Collezione
  NomeVariabile.LIMIT; -- fornisce il numero massimo possibile di elementi della Collezione
- **FIRST E LAST**
  Sintassi: NomeVariabile.FIRST; -- fornisce l'indice del primo elemento della Collezione
  NomeVariabile.LAST; -- fornisce l'indice dell'ultimo elemento della Collezione
- **PRIOR E NEXT**
  Sintassi: NomeVariabile.PRIOR(N); -- fornisce l'indice dell'elemento che precede l'elemento N-esimo
  NomeVariabile.NEXT(N); -- fornisce l'indice dell'elemento che segue l'elemento N-esimo

4 Elementi del linguaggio

4.1 Commenti

Sintassi: --[Commento]
/*[commento]*/

La prima forma è solo a livello di riga, la forma /* */ genera un commento anche su più righe.

Esempi:

```
-- compute the area of a circle
area := pi * radius**2; /* pi equals 3.14159 */
```

4.2 Assegnazione

Sintassi: Variabile := espressione;[3]

SELECT .... INTO Variabile FROM.... ;

Esempi:
tax := prezzo * tax_rate;
premio := salario * 0.10;
valore := TO_NUMBER(SUBSTR('750 dollars', 1, 3));
valido := FALSE;

Nell'esempio seguente la variabile *premio* conterrà il 10% del salario di ROSSI CARLO:

SELECT salario * 0.10 INTO premio FROM anagrafica WHERE nome = 'ROSSI CARLO';

4.3 Goto e etichette (labels)

Le etichette indicano punti del programma. Sono utilizzabili, tramite l'istruzione GOTO per far proseguire il programma da quel punto.

Sintassi: <<NomeEtichetta>>
GOTO NomeEtichetta;

Non è possibile eseguire un GOTO dall'esterno all'interno di istruzioni IF, LOOP e sottoblocchi del programma.

4.4 If

Sintassi:IF EspressioneBooleana THEN istruzione(i)
[ELSIF EspressioneBooleana THEN istruzione(i)
ELSIF EspressioneBooleana THEN istruzione(i)
……..]
[ELSE istruzione(i)]
END IF;

Esempio:

```
IF shoe_count < 20 THEN
     order_quantity := 50;
ELSIF shoe_count < 30 THEN
     order_quantity := 20;
ELSE
  order_quantity := 10;
END IF;
```

[3] Si tenga presente che in SQL l'assegnazione, ad esempio nella clausola SET del comando UPDATE, è ottenuta con il simbolo =.

4.5 Iterazioni

Sintassi: LOOP istruzione(i) END LOOP;
WHILE EspressioneBooleana LOOP istruzione(i) END LOOP;
FOR VariabileIndice IN (REVERSE) ValoreIniziale..ValoreFinale LOOP istruzione(i) END LOOP;
FOR NomeRecord IN {Cursore | ComandoSelect} LOOP istruzione(i) END LOOP;

Note: Ad ognuna delle quattro forme di LOOP si può premettere una *Etichetta di programma* che può anche apparire dopo END LOOP.
VariabileIndice **non deve essere dichiarata** ne segue che non può essere utilizzata al di fuori del ciclo.

Esempi:

```
FOR i IN REVERSE 1..10 LOOP -- i starts at 10, ends at 1
-- statements here execute 10 times
END LOOP;

DECLARE
bonus REAL;
CURSOR c1 IS SELECT empno, sal, comm FROM emp;
BEGIN
FOR c1rec IN c1 LOOP
   bonus := (c1rec.sal * 0.05) + (c1rec.comm * 0.25);
   INSERT INTO bonuses VALUES (c1rec.empno, bonus);
END LOOP;
COMMIT;
END;
```

4.6 Uscita da Iterazione

Sintassi: EXIT | EXIT WHEN EspressioneBooleana;

4.7 Procedure e funzioni

Procedure e funzioni sono sottoprogrammi. Le funzioni restituiscono un valore al programma chiamante, tramite l'istruzione RETURN.

Sintassi

```
FUNCTION NomeFunzione [(parameter[, parameter, ...])] RETURN Tipo IS
[dichiarazioni locali]
BEGIN
-- istruzioni
RETURN Espressione corrispondente a Tipo;
[EXCEPTION gestione eventuali errori]
END [NomeFunzione];

PROCEDURE NomeProcedura [(parameter[, parameter, ...])] IS
[dichiarazioni locali]
BEGIN
-- istruzioni
[EXCEPTION gestione eventuali errori]
END [NomeProcedura];
```

dove parameter è parameter_name [IN | OUT | IN OUT] Tipo [{:= | DEFAULT} espressione]
IN indica che il parametro non è modificabile dal sottoprogramma,
OUT indica un parametro modificabile dal programma

Esempio

```
FUNCTION sal_ok (salary REAL, title REAL) RETURN BOOLEAN IS
min_sal REAL;
max_sal REAL;
BEGIN
SELECT losal, hisal INTO min_sal, max_sal FROM sals WHERE job =
title;
RETURN (salary >= min_sal) AND (salary <= max_sal);
END sal_ok;

DECLARE
  Moltip NUMBER;
```

```
            CURSOR c1 IS SELECT empno, ename, job, sal FROM emp WHERE sal >
             2000;
            FUNCTION Prodotto (Molt1 Real, Molt2 Real) RETURN Number IS
            BEGIN
               Return(Molt1 * Molt2);
            END;
          BEGIN
          FOR emp_record IN c1 LOOP
          Moltip := Prodotto(emp_record.sal,1.20);
          INSERT INTO TABALVOLO (Errore,Nome,Danumero) VALUES
          (emp_record.Ename,emp_record.sal,Moltip);
          END LOOP;
          END;
/
```

Nell'esempio seguente si utilizza il Package *DBMS_OUTPUT*, (v. 7.1.4)utile per eseguire il debug.

```
          CREATE PROCEDURE calc_payroll (payroll IN OUT REAL) AS
          CURSOR c1 IS SELECT sal,comm FROM emp;
          BEGIN
          payroll := 0;
          FOR c1rec IN c1 LOOP
          c1rec.comm := NVL(c1rec.comm, 0);
          payroll := payroll + c1rec.sal + c1rec.comm;
          END LOOP;
          /* Display debug info. */
          DBMS_OUTPUT.PUT_LINE('payroll: ' || TO_CHAR(payroll));
          END calc_payroll;
```

4.7.1 Ritorno da sottoprogrammi

E' il comando che permette di uscire da un sottoprogramma o da una funzione.

Sintassi RETURN [(ValoreDi Ritorno)

Note ValoreDiRitorno è il valore calcolato dalla Funzione.

4.7.2 Passaggio di Collezione a Funzione o procedura

```
CREATE OR REPLACE PACKAGE MATRIX AS
TYPE Staff IS VARRAY (20) OF Number;
staffer Staff;
FUNCTION Elemento (Staffer  Staff) RETURN NUMBER;
PRAGMA RESTRICT_REFERENCES (Elemento, WNDS);
END MATRIX;
/
CREATE OR REPLACE PACKAGE BODY MATRIX AS
FUNCTION Elemento (Staffer Staff) RETURN Number IS
BEGIN
  return(3.14159);
END ELEMENTO;
END MATRIX;
/
DECLARE
staffer Matrix.Staff := Matrix.Staff(0,1,2,3);
Indx number;
BEGIN
staffer.EXTEND(10,1);
Indx := MATRIX.Elemento(Staffer);
Staffer(1) := 12345;
Indx := Staffer(1);
Indx := GEN_RAND.RAND(Indx);
dbms_output.put_line(TO_CHAR(Indx));
END;
/
```

4.7.3 Prova del codice nell'ambiente SQL*PLUS

- Dichiarazione delle variabili

```
SQL> Variable m1 Number
SQL> Variable m2 Number
SQL> Variable m3 Number
```

- Esecuzione del programma

```
SQL> EXECUTE :m1:=33; -- assegno alla variabile m1 il valore 33
SQL> EXECUTE :m2:=3; -- assegno alla variabile m2 il valore 3
```

- Visualizzazione dei risultati

```
SQL> Print m1 m2
SQL> Execute :m3  := Prodotto(:m1,:m2)
SQL> select :m1,:m2,:m3 from dual;
SQL> select Prodotto(15,12) from dual;
```

4.7.3.1 Rendere pubblico il codice

```
SQL> grant execute on prodotto to public;
SQL> Execute :m3  := scott.Prodotto(:m1,:m2)

SQL> create public synonym prodotto for scott.prodotto;
SQL> Execute :m3  := Prodotto(:m1,:m2)
```

4.8 Gestione degli errori

Gli errori sono gestiti nel blocco EXCEPTION.

Sintassi WHEN TipoErrore [OR TipoErrore] … THEN Comando(i);
WHEN OTHERS THEN Comando(i);
TipoErrore: Tipo errore dell'utente, Tipo errore di sistema (es. ZERO_DIVIDE), OTHERS

Note Il tipo errore dell'utente deve essere dichiarato nel blocco DECLARE; per richiamarlo si usa il comando RAISE TipoErrore;

Esempio

```
create table TABALVOLO (c1 varchar2(10), c2 varchar2(70));
DECLARE
Mio_Errore EXCEPTION;
BEGIN
  DECLARE
  Alfa VARCHAR2(4) := 'Alfa';
  BEGIN
    INSERT INTO TABALVOLO (C1,C2) values (Alfa,'Prima di generare
    Exception');
    RAISE Mio_Errore;
    EXCEPTION
      WHEN Mio_Errore THEN
      INSERT INTO TABALVOLO (C2) values ('Generata Exception');
  END;
INSERT INTO TABALVOLO (C2) values ('Dopo la ripresa da Exception');
END;
/
Select * from TABALVOLO;
DROP TABLE TabalVolo;
```

5 Integrazione con altri linguaggi di programmazione

In ambiente WINDOWS è sostanzialmente il richiamo di DLL. I passi da seguire sono:

- Identificare la DLL
- Registrare la/le procedure o funzioni della DLL

Sintassi dell'identificazione della DLL: `CREATE LIBRARY library_name {IS | AS} 'file_path';`

Sintassi della registrazione:

```
CREATE FUNCTION Nome ( Parametro(i)) RETURN Tipo
AS EXTERNAL LIBRARY library_name NAME  "NomeFunzionein DLL"  -- quotes
preserve lower case
LANGUAGE C;
```

Esempio:

```
CREATE OR REPLACE LIBRARY spool32 IS 'C:\ROSS\SPOOL32.DLL'
/
CREATE OR REPLACE FUNCTION Token (
Parola IN OUT VARCHAR2,
Stringa IN OUT VARCHAR2) RETURN BINARY_INTEGER
AS EXTERNAL LIBRARY spool32
```

```
NAME "SPOOLTOKEN";  -- quotes preserve lower case
--LANGUAGE C; -- C per ora è il solo linguaggio accettato
/
DECLARE
 Miastringa VARCHAR2(50) := 'Parola1 parola2 parola3';
 Parola VARCHAR2(15) := '';
 Response INTEGER;
BEGIN
WHILE LENGTH(Miastringa) > 0 LOOP
  Response := Token(Parola, Miastringa);
  DBMS_OUTPUT.PUT_LINE('Parola estratta: ' || Parola);
END LOOP;
END;
/
```

6 Interazione con SQL

6.1 SELECT

Sintassi SELECT …. INTO NomeRecord FROM …;
SELECT …. INTO NomeVariabile[NomeVariabile]…. FROM …,

Esempio
```
SELECT STDDEV(comm) INTO comm_sigma FROM emp;
```

Note SELECT INTO da errore se le righe ritrovate sono diverse da 1. Se possibile conviene utilizzare i cursori.

6.2 Lavorare con i Cursori

6.2.1.1 Istruzioni

Sintassi
```
OPEN NomeDiCursore;                          -- apre un cursore
OPEN NomeDiCursore FOR SELECT …;             -- apre un cursore
FETCH NomeDiCursore INTO NomeStruttura;      -- porta in memoria una riga
CLOSE NomeDiCursore;                         -- chiude un cursore
```

Esempio
```
DECLARE
CURSOR c1 IS SELECT ename, sal, hiredate, job FROM emp;
emp_rec c1%ROWTYPE; -- declare record variable that represents a row in
the emp table
FETCH c1 INTO emp_rec;
```

6.2.1.2 Attributi del cursore

I Cursori hanno quattro attributi:

- %FOUND Vale NULL prima della prima FETCH, TRUE se il comando FETCH ha reso disponibile una riga, altrimenti vale FALSE
- %ISOPEN Vale TRUE o FALSE
- %NOTFOUND E' il contrario di %FOUND
- %ROWCOUNT E' il numero di righe –1 che FETCH ha reso disponibili

Esempi
```
DECLARE
   CURSOR C1 IS SELECT empno, ename, job, sal FROM emp WHERE sal > 2000;
   CURSOR c2 RETURN dept%ROWTYPE IS SELECT * FROM dept;
   dip dept%ROWTYPE; -- il tipo dip è dello stesso della riga della tabella dept
   my_ename emp.ename%type;
   my_ deptno emp. deptno%type;
BEGIN
OPEN c2;
LOOP
   FETCH c2 into Dip;
   EXIT WHEN c2%NOTFOUND;
   INSERT INTO TABALVOLO (Errore,Nome) VALUES ('Da tab deptno',Dip.deptno);
END LOOP;
CLOSE c2;
OPEN c1; -- apertura cursore
-- esempio con dichiarazione implicita di record (deptno_record)
```

```
-- in questo caso non sono necessarie OPEN e CLOSE
FOR deptno_record IN c2 LOOP
   INSERT INTO TABALVOLO (Errore,Nome) VALUES ('Da deptno via LOOP',
deptno_record.deptno);
END LOOP;
END;
/
LOOP
FETCH c1 INTO my_ename, my_deptno;
IF c1%ROWCOUNT > 10 THEN
...
END IF;
...
END LOOP;
```

6.3 Cursori impliciti

Si hanno *cursori impliciti* in presenza di comandi SQL non associati espressamente ad un cursore.
PL/SQL permette di riferirsi al più recente *cursore implicito* tramite i quattro attributi: %FOUND, %ISOPEN, %NOTFOUND e %ROWCOUNT.

Esempio
```
DELETE FROM parts WHERE status = 'OBSOLETE';
IF SQL%ROWCOUNT > 100 THEN -- more than 100 rows were deleted
RAISE large_deletion;
END IF;
```

7 I Packages

Il Package è un oggetto che raggruppa tipi di dato, variabili e sottoprogrammi di PL/SQL generalmente in relazione fra di essi. Il package è formato da specifiche, l'interfaccia verso le applicazioni che lo usano, ed un corpo in cui è effettuata la particolare elaborazione.

Sintassi
```
CREATE PACKAGE name AS -- specification (visible part)
     -- public type and item declarations
     -- subprogram specifications
END [name];

CREATE PACKAGE BODY name AS -- body (hidden part)
     -- private type and item declarations
     -- subprogram bodies
[BEGIN
     -- initialization statements]
END [name];
```

Note Occorre creare il Package e successivamente il corpo

Esempio
```
Drop PACKAGE GEN_RAND;
CREATE PACKAGE GEN_RAND AS

FUNCTION Rand (Seme REAL) RETURN REAL;
PRAGMA RESTRICT_REFERENCES (Rand, WNDS);
END GEN_RAND;
/
CREATE PACKAGE BODY GEN_RAND  AS
FUNCTION RAND (Seme REAL) RETURN REAL IS
Comodo VARCHAR2(12);
BEGIN
Comodo := to_Char(Seme * Seme);
--dbms_output.put_line('Comodo: ' || Comodo);
RETURN (TO_NUMBER(Comodo));
END RAND;
END GEN_RAND;
/
```

7.1 Package di sistema

Sono Package forniti con ORACLE.

7.1.1 Package STANDARD

Il package STANDARD definisce l'ambiente PL/SQL . Il package definisce tipi dichiarati a livello globale, le eccezioni, i sottoprogrammi e le funzioni disponibili automaticamente a tutti i programmi PL/SQL.
E' possibile ridefinire nel programma una funzione, ad esempio ABS, che sostituisce quella del Package. Per richiamare la versione originale la sintassi è STANDARD.NomeFunzione.

7.1.2 UTL_FILE

Il package UTL_FILE permette ad un programma PL/SQL di leggere e scrivere file di testo.
Esempio

```
CREATE TABLE TAGLIAERBA
      (Data  Date,
       Ore   Number(5,1),
       Costo Number(7),
       Localita  varchar2(15),
       Descrizione varchar2(40));

DECLARE
-- handle dei files
hFlin UTL_FILE.FILE_TYPE;
hFlout UTL_FILE.FILE_TYPE;
Riga Varchar2(200);
Campo VARCHAR2(100); -- campo estratto
my_sqlerrm  CHAR(70);
CkIn PLS_INTEGER;
CkCampi PLS_INTEGER;
Separator char := chr(9);
-- tabella interna per memorizzare i campi estratti dai record
TYPE ArrayType IS TABLE OF Varchar2(100)
      INDEX BY BINARY_INTEGER;
Array ArrayType;  -- declare PL/SQL table

BEGIN
  BEGIN
    CKIn := 0;
    -- apro file di log
    hFlout := UTL_FILE.FOPEN('C:\CONDOR\ORACLE','TAGLIAERBA.log','w');
    -- apro file di input
    hFlin := UTL_FILE.FOPEN('C:\CONDOR\ORACLE','TAGLIAERBA.txt','r');
    LOOP
      UTL_FILE.GET_LINE(hflin,Riga);
      CKIn := CKIn + 1;
      UTL_FILE.PUT_LINE (hflout, Riga);
      -- estrazione campi
      CAMPO := '';
      Riga := CONCAT(Riga, Separator);
      CkCampi := 0;
      FOR I in 1..LENGTH(Riga) LOOP
        IF SUBSTR(Riga,I,1) = Separator THEN
        CkCampi := CkCampi + 1;
        Array(CkCampi) := Campo;
          CAMPO := '';
        ELSE
          Campo := CONCAT(Campo,SUBSTR(Riga,I,1));
        END IF;
      END LOOP;
      INSERT INTO TAGLIAERBA VALUES(TO_DATE(Array(1),'dd/mm/yy'),

TO_NUMBER(Array(2),'9999D9','NLS_NUMERIC_CHARACTERS =',.'''),
                                  TO_NUMBER(Array(3)),
                                  Array(4),
                                  Array(5));
```

```
    END LOOP;

  EXCEPTION
    WHEN UTL_FILE.INVALID_PATH THEN
      UTL_FILE.PUT_LINE (hflout, 'DIRECTORY/FILE INVALIDO');
    WHEN UTL_FILE.INVALID_FILEHANDLE THEN
      UTL_FILE.PUT_LINE (hflout, 'HANDLE INVALIDO');
    WHEN UTL_FILE.INVALID_OPERATION THEN
      UTL_FILE.PUT_LINE (hflout, 'OPERAZIONE INVALIDA');
    WHEN UTL_FILE.READ_ERROR then
      UTL_FILE.PUT_LINE (hflout, 'ERRORE LETTURA');
    -- gestione della fine del file
    WHEN NO_DATA_FOUND THEN
      UTL_FILE.PUTF(hflout, 'letti %s Records\n',CKIN);
    WHEN OTHERS THEN
      my_sqlerrm := SUBSTR(SQLERRM, 1, 70);
      UTL_FILE.PUTF(hflout, 'ERRORE %s\n', my_sqlerrm);
  END;
  UTL_FILE.FCLOSE(hFlin);
  UTL_FILE.FCLOSE(hFlout);
END;
/
```

7.1.3 DBMS_SQL

Il Package DBMS_SQL permette a PL/SQL di eseguire comandi SQL sia di definizione di dati sia comandi di trattamento di dati in modo dinamico.

Esempio

```
DECLARE
   MioNumero INTEGER(7,3);
   cid INTEGER;
   Comando VARCHAR(200);
   ritorno INTEGER;
BEGIN
   cid := DBMS_SQL.OPEN_CURSOR; -- Open new cursor and return cursor ID.
   MioNumero := 73.13;
   DBMS_SQL.PARSE(cid, 'CREATE TABLE TABALVOLO (CAMPO VARCHAR2(20), CAMPO2
   NUMBER(5))', dbms_sql.v7);
   Comando := 'INSERT INTO TABALVOLO (CAMPO, CAMPO2) VALUES (' || MioNumero ||
   ', 40)';
   DBMS_SQL.PARSE(cid, Comando,dbms_sql.v7);    /* esegue l'aggiornamento */
   ritorno := DBMS_SQL.EXECUTE(cid);

   DBMS_SQL.CLOSE_CURSOR(cid);    /* Close cursor. */
EXCEPTION
   /* If an exception is raised, close cursor before exiting. */
   WHEN OTHERS THEN
      DBMS_SQL.CLOSE_CURSOR(cid);
      RAISE;  -- reraise the exception
END;
/
Select * from TABALVOLO;
DROP TABLE TabalVolo;
```

7.1.4 DBMS_OUTPUT

Il package DBMS_OUTPUT permette di visualizzare un output da un blocco PL/SQL e da sottoprogrammi, facilitandone la messa a punto. In SQL*Plus il comando SET SERVEROUTPUT ON permette la visualizzazione delle informazioni.

L'esempio utilizza la procedura dell'esempio relativo al paragrafo 4.7.

```
SQL> SET SERVEROUTPUT ON
SQL> VARIABLE num NUMBER
```

```
SQL> EXECUTE calc_payroll(:num)
```

8 Interazione SQL*PLUS e PL/SQL

8.1 Comandi VARIABLE, EXECUTE, PRINT

Dichiara una *bind* variabile:

Sintassi VARIABLE NomeVariabile Tipo
EXECUTE (NomeProcedura | assegnazione PL/SQL)
PRINT NomeVariabile

Esempio:
```
VARIABLE num NUMBER
EXECUTE :num := 3.14159;
EXECUTE :num2 := GEN_RAND.RAND(:num);
SQL> print num
      NUM
---------
9.8695877
```

8.2 Comando DEFINE

Il comando serve per l'interazione con procedure SQL in cui sono previsti parametri da sostituire. (Quelli individuati da &).

Sintassi DEFINE NomeVariabile = Valore

Esempio:
```
DEFINE EMPLOYEE = SMITH
select ename from scott.emp where ename = '&Employee';
```

Nota: Il comando `DEFINE` da solo, elenca tutte le variabili definite.