

FUNZIONI

- Dichiarazione:
 `function nome [(parametri)] return tipo;`
- Definizione:
 `function nome [(parametri)] return tipo is`
 dichiarazioni;
 `begin` comandi;
 `exception` gestori;
 `end [nome];`
- Il corpo contiene uno o più comandi `return expr` (meglio uno solo e meglio in coda)
- Se la funzione esce senza ritornare un valore vi è un errore a run-time
- Chiamata: `nome [(attuali)]`

PARAMETRI

- Sintassi:
 - parametri := parametro {, parametro }*
 - parametro
 ::= nome [in | out | in out] tipo [default expr]
- Tipo:
 - uno scalare (boolean, number, varchar2...) senza dimensioni
 - nome%type, nome%rowtype
- possono essere in (caso default), out, in out
- i parametri in possono avere un valore default (default expr oppure := expr)

PROCEDURE

- Dichiarazione:
 - `procedure` nome [(parametri)];
- Definizione:
 - `procedure` nome [(parametri)] is
 - dichiarazioni;
 - `begin` comandi;
 - `exception` gestori;
 - `end` [nome];

DEFINIZIONI DI PROCEDURE

- In un corpo di modulo (globali)
- In coda in una qualunque sezione declare
- Dichiarazione forward: ha la sintassi di una dichiarazione
- Nella base di dati:
 - `create procedure / function [(..)] is / as`
- È ammesso in una certa misura l'overloading di funzioni i cui parametri hanno un tipo diverso

PARAMETRI POSIZIONALI O NOMINATI

- Notazione posizionale:
 - `dividi(10,5)`
- Nominale:
 - `dividi(dividendo => 10, divisore => 5)`
 - `dividi(divisore => 5, dividendo => 10)`
- Mista:
 - `dividi(10, divisore => 5)`: sì
 - `dividi(dividendo => 10, 5)`: no
 - `dividi(10, dividendo =>5)`: no

IN, OUT, IN OUT

- **in**: caso default; è una costante; una funzione dovrebbe avere solo parametri in
- **out**: possono solo subire assegnamenti nel corpo della procedura
- **in out**: sono come variabili
- Non è definito se i parametri sono passati per valore o per riferimento; evitare quindi l'aliasing

IN, OUT, IN OUT

- Dichiarazione:
 - create procedure f(
 f1 in varchar2,
 f2 in out varchar2,
 f3 out varchar2) as body;
- Chiamata
 - f(3, y, x.a)
- Vuole dire:
 - f1 := 3; f2 := y; (f1 in, f2 in out)
 - body;
 - y := f2; x.a := f3; (f2 in out; f3 out)

MODULI (PACKAGES)

- Un modulo ha un'interfaccia ed un'implementazione; entrambe sono memorizzate in uno schema
- Vantaggi: modularità, information hiding, compilazione separata
- Interfaccia:
 - Definisce:
 - Tipi, variabili, costanti, eccezioni
 - Dichiarazione: cursori, procedure/funzioni
- Implementazione:
 - Definisce:
 - Cursori e procedure pubbliche
 - Tipi, variabili, costanti, eccezioni, cursori, procedure private
 - Non è indispensabile

SINTASSI

create package nome as

definizioni pubbliche;

dichiarazioni pubbliche di cursori;

dichiarazioni di procedure pubbliche;

end [nome];

create package body nome as

definizioni private e definizioni dei cursori pubblici;

definizioni di procedure private e pubbliche;

[**begin** codice di inizializzazione rieseguito per ogni sessione]

end [nome];

- Lo stato di un package è legato alla sessione (connessione/disconnessione); due sessioni concorrenti vedono due stati indipendenti

DICHIARAZIONI E DEFINIZIONI DI CURSORI E PROCEDURE

- Procedure:
 - `procedure` nome[(parametri)];
 - `procedure` nome[(parametri)]
`is ...;`
- Funzioni:
 - `function` nome[(parametri)] `return` tipo;
 - `function` nome[(parametri)] `return` tipo
`is ... ;`
- Cursori:
 - `cursor` nome[(parametri)] [`return` tipo];
 - `cursor` nome[(parametri)] [`return` tipo]
`is query;`
 - Nei cursori la parte *return tipo* serve solo per poter mettere la dichiarazione nel package e la definizione nel corpo di un modulo

MODULI PREDEFINITI

- `standard`: contiene le funzioni standard (che quindi restano accessibili dopo un'eventuale ridefinizione)
- `dbms_standard`: le funzioni SQL standard
- `dbms_sql`: esecuzione dinamica di SQL
- `dbms_output`: output verso `sqlplus`
- `utl_file`: I/O su files del sistema operativo
- altri 103 ...

ECCEZIONI

- Le eccezioni si dichiarano:
 - In un modulo (visibilità globale)
 - In un blocco (visibilità locale)
- Sintassi
 - nomeecc **exception**;
- Si sollevano con un **raise** nomeecc
- Si propagano ai blocchi esterni ed alle funzioni chiamanti fino a che non trovano una sezione **exception** in grado di gestirle
- Terminata l'esecuzione del gestore, il blocco termina
- Due eccezioni diverse con lo stesso nome sono diverse
- Un fallimento nella sezione **declare** o **exception** può essere solo gestito da un blocco più esterno.

CONDIVISIONE DI ECCEZIONI

```
create or replace package eccp as
  f exception;
  g exception;
  procedure uu;
end eccp;
```

```
create or replace package pack2 as ...;
```

```
create or replace package body pack2 as
  procedure ss is
  begin
    raise eccp.f ;
  exception
    when eccp.g then stampaP.stampa('eccezione g');
  end ss;
end pack2;
```

ECCEZIONI INTERNE

- Molte hanno un numero, non un nome. Per darvi un nome:

```
declare pochi_privilegi exception;
```

```
pragma exception_init(pochi_privilegi, -1031);
```

Eccezioni interne: esempio

- Errore:
 - begin select * from acquisti into aa; end;
 - ERRORE alla riga 1:
 - ORA-01422: exact fetch returns more than requested number of rows
 - ORA-06512: at ...
- Definisco l'eccezione:

```
create or replace package p is
  troppe_righe exception;
  pragma exception_init(troppe_righe,-1422);
end p;
```

Eccezioni interne: esempio

- Gestisco l'eccezione:

```
begin
```

```
  select * from acquisti into aa; end;
```

```
exception
```

```
  when troppe_righe then ...
```

```
end;
```

I GESTORI

- Sintassi:

exception

when ecc1 **or** ... **or** ecc5 **then**

seq di comandi;

when ecc6 **then**

seq di comandi;

when others **then**

seq di comandi;

end

- **raise** (solo nella sezione **exception**) risolve l'eccezione corrente

SQLCODE ED SQLERRM

- **sqlcode**: il numero dell'eccezione:
 - utente: +1
 - sql: negativo, o +100 (no data found)
- **sqlerrm**: il messaggio dell'eccezione:
 - utente: 'User-Defined Exception'
 - sql: un qualche messaggio
- **sqlerrm(i)**: il messaggio associato ad i
- **raise_application_error(code,message)**: application error che si comporta come se fosse predefinito

RIESEGUIRE UNA TRANSAZIONE

```
suffisso:=0;
nome:=ilNome;
while suffisso < 100 loop
  begin
    savepoint riparti;
    ....
    insert into persone(nome, ...)
    commit;
    exit;
  exception
    when dup_val_on_index then
      rollback to riparti;
      nome := ilNome || tochar(suffisso);
      suffisso := suffisso + 1;
  end;
end loop;
```