Stored procedure

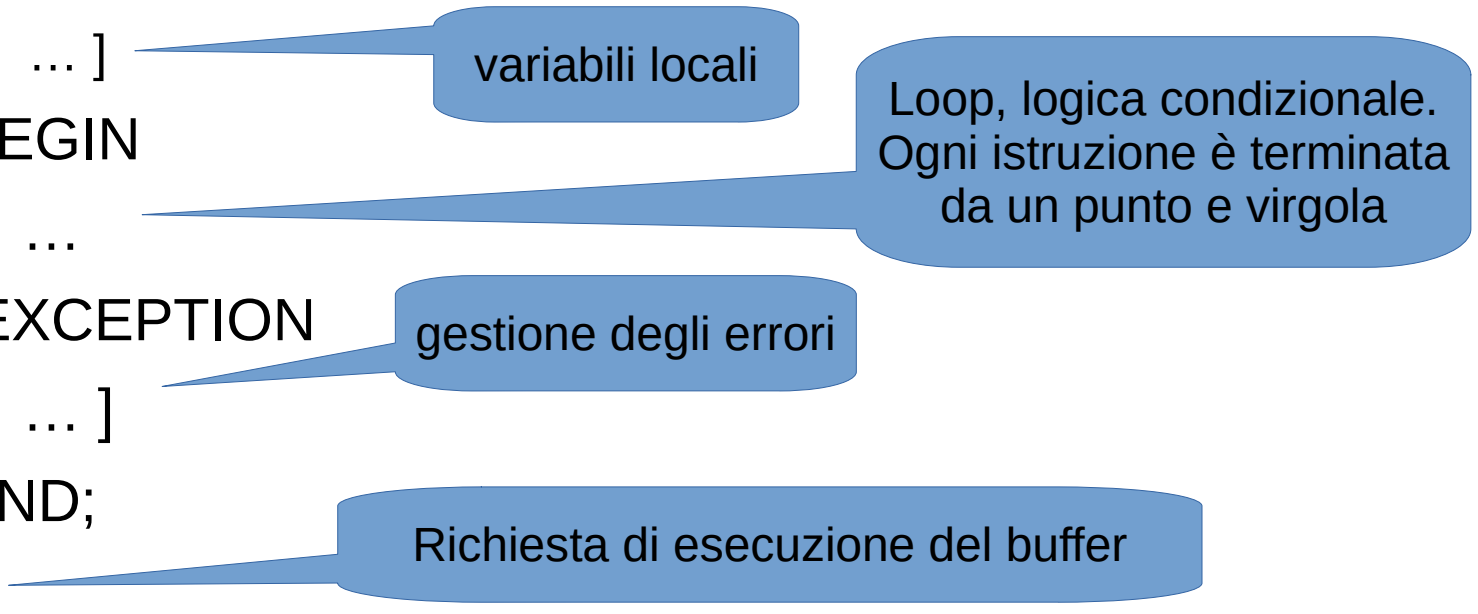
Supporto procedurale a SQL fornito da alcuni DBMS

- Non fa parte dello standard SQL
- Oracle: PL/SQL (linguaggio procedurale basato su SQL)

- procedura: accetta parametri (in/out)
- funzione: procedura che ritorna un valore
- trigger: procedura eseguita automaticamente
  - in seguito ad una operazione DML su una tabella
- Progetto di riferimento
  - https://github.com/egalli64/mjd – folder oracle *(modulo 5)*

Blocco

```
[DECLARE
    … ]
BEGIN
    …
[EXCEPTION
    … ]
END;
/
```



Hello PL/SQL

- Un blocco minimale
  - non sono necessarie DECLARE e EXCEPTION
- Se l'output su console non è attivo
  - set serveroutput on

```
begin
   dbms_output.put_line('Hello PL/SQL');
end;
/
```

Variabili

- Dichiarazione / Definizione
  - nel blocco DECLARE
- Assegnamento via :=
- Supporto per tutti i tipi Oracle SQL
  - più altri, come BOOLEAN
- Una convenzione sul nome
  - prefisso “v_”

```
declare
    v_width integer;
    v_height integer := 2;
    v_area integer := 6;
begin
    v_width := v_area / v_height;
    dbms_output.put_line(
        'v_width = ' || v_width);
end;
/
```

Eccezioni

- Gestione degli errori di esecuzione
- Simile al meccanismo try/catch di Java

```
begin
    dbms_output.put_line(6 / 0);
exception
    when zero_divide then
        dbms_output.put_line('Zero divide!');
end;
/
```

```
begin
    dbms_output.put_line(1 / 0);
exception
    when others then
        dbms_output.put_line('Exception!');
end;
/
```

IF – ELSIF – ELSE – END IF

```
declare
    v_a integer := 1;
begin
    if v_a > 0 then
        dbms_output.put_line('v_a is positive');
    elsif v_a = 0 then
        dbms_output.put_line('v_a is zero');
    else
        dbms_output.put_line('v_a is negative');
    end if;
end;
/
```

LOOP

- Loop semplice: EXIT (WHEN), CONTINUE (WHEN)

```
v_x := 0;
loop
    v_x := v_x + 1;
    if v_x = 3 then exit;
    end if;
end loop;
```

```
v_x := 0;
loop
    v_x := v_x + 1;
    exit when v_x = 5;
end loop;
```

```
v_x := 0;
loop
    v_x := v_x + 1;
    if v_x = 3 then
        -- something special
        continue;
    end if;
    exit when v_x = 5;
end loop;
```

```
v_x := 0;
loop
    v_x := v_x + 1;
    continue when v_x = 3;

    -- something normal
    exit when v_x = 5;
end loop;
```

WHILE e FOR

- WHILE LOOP, finché la condizione è vera
- FOR LOOP, per ogni valore indicato (REVERSE)

```
v_x := 0;
while v_x < 5 loop
    v_x := v_x + 1;
end loop;
```

```
for i in 1..5 loop
    dbms_output.put_line('for loop: ' || i);
end loop;
```

```
for i in reverse 1..5 loop
    dbms_output.put_line('for loop: ' || i);
end loop;
```

SELECT INTO

- Lettura di una singola riga

```
declare
    v_first_name coders.first_name%type;
    v_last_name coders.last_name%type;
begin
    select first_name, last_name
    into v_first_name, v_last_name
    from coders
    where coder_id = 103;

    dbms_output.put_line('[' || v_first_name || ' ' || v_last_name || ']');
end;
/
```

Tipo di una colonna

CURSOR

- Lettura di più righe

- Si definisce un CURSOR associato a SELECT
- OPEN CURSOR esegue la SELECT
- FETCH – INTO legge la riga corrente
- EXIT WHEN %NOTFOUND termina la lettura del cursore
- CLOSE CURSOR rilascia le risorse associate

```
declare
   v_last_name coders.last_name%type;
   v_hire_date coders.hire_date%type;
   cursor v_coder_cursor is
      select last_name, hire_date from coders;
begin
   open v_coder_cursor;
   loop
      fetch v_coder_cursor
      into v_last_name, v_hire_date;
      exit when v_coder_cursor%notfound;

      dbms_output.put_line(
         '[' || v_last_name || ', ' || v_hire_date || ']');
   end loop;
   close v_coder_cursor;
end;
/
```

CURSOR in FOR LOOP

- gestione implicita, codifica semplificata

OPEN cursor implicita

CLOSE cursor implicita

```
declare
    cursor v_coder_cursor is
        select last_name, hire_date from coders;
begin
    for v_cur in v_coder_cursor loop
        dbms_output.put_line(
            '[' || v_cur.last_name || ', ' || v_cur.hire_date || ']');
    end loop;
end;
/
```

CREATE PROCEDURE

Parametri IN / OUT

PROCEDURE body

IS / AS

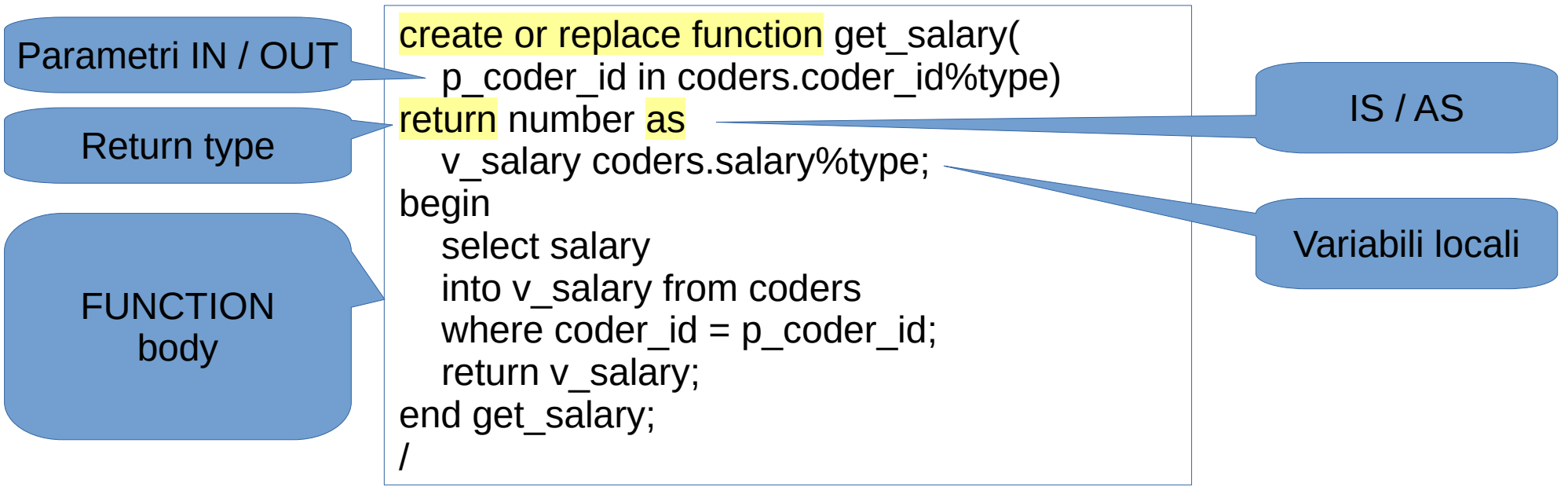
```
create or replace procedure get_coder_salary(
    p_coder_id in coders.coder_id%type,
    p_salary out coders.salary%type) is
begin
    select salary
    into p_salary
    from coders
    where coder_id = p_coder_id;
end get_coder_salary;
/
```

```
drop procedure get_coder_salary;
```

Esecuzione di una procedura

```
declare
    v_id coders.coder_id%type := 105;
    v_salary coders.salary%type;
begin
    get_coder_salary(v_id, v_salary);
    dbms_output.put_line('Salary is ' || v_salary);
exception
    when others then
        dbms_output.put_line('Can''t get salary for ' || v_id);
end;
/
```

CREATE FUNCTION




```
create or replace function get_salary(
    p_coder_id in coders.coder_id%type)
return number as
    v_salary coders.salary%type;
begin
    select salary
    into v_salary from coders
    where coder_id = p_coder_id;
    return v_salary;
end get_salary;
/
```

```
drop function get_salary;
```

Esecuzione di una funzione

```
declare
    v_id coders.coder_id%type := 105;
    v_salary coders.salary%type;
begin
    v_salary := get_salary(v_id);
    dbms_output.put_line('Salary is ' || v_salary);
exception
    when others then
        dbms_output.put_line('Can''t get salary for ' || v_id);
end;
/
```

TRIGGER

- Procedura eseguita automaticamente in relazione (prima, dopo, o invece) all'esecuzione di un comando DML
- Row-level
  - Eseguito per ogni riga coinvolta
  - In update, accesso a stato precedente e successivo
  - Esecuzione condizionale
- Statement-level
  - Eseguito una volta per tutte le righe

Un esempio di trigger

Tabella di output del trigger

```
create table coder_salaries (
    coder_id number(6, 0)
        references coders(coder_id),
    old_salary number(8, 2),
    new_salary number(8, 2)
);
```

Trigger

```
create or replace trigger salary_update
before update of salary on coders
for each row
begin
    insert into coder_salaries values(
        :old.coder_id, :old.salary, :new.salary);
end salary_update;
/
```

Generazione di eventi che scatenano il trigger

```
update coders
set salary = salary * 1.3
where coder_id > 103;
```

Esercizi

- Scrivere la procedura day_after() che ha parametri
  - Input: una data
  - Output: la data di domani
- Riscrivere la day_after() come funzione
- Scrivere la procedura get_coder() con parametri
  - Input: id di un coder
  - Output: nome e cognome associato
- Riscrivere la get_coder() come funzione