Tehnici de programare cu baze de date

#11 PL/SQL
Funcții în PL/SQL (partea II-a)

https://www.runceanu.ro/adrian/activitate-didactica/

Curs 11

Funcţii în PL/SQL (partea II)

Cuprins

Funcţii în PL/SQL(partea II)

1. Recursivitate
2. Utilizarea în expresii SQL a funcţiilor definite de utilizator
3. Informaţii referitoare la subprograme
4. Dependenţa subprogramelor

1. Recursivitate

- *Un subprogram recursiv presupune că acesta se apelează pe el însuşi.*
- În *Oracle* o problemă delicată este legată de locul unde se plasează un apel recursiv.
- De exemplu, dacă apelul este în interiorul unui cursor *FOR* sau între comenzile *OPEN* şi *CLOSE*, atunci la fiecare apel este deschis alt cursor.
- În felul acesta, programul poate depăşi limita pentru *OPEN_CURSORS* setată în parametrul de iniţializare *Oracle*.

1. Recursivitate

Exemplu:

Să se calculeze recursiv al *m*-lea termen din şirul lui Fibonacci.

```
FUNCTION fibonacci(m INTEGER)
RETURN INTEGER IS
BEGIN
  IF (m = 1) OR (m = 2) THEN
    RETURN 1;
  ELSE
    RETURN fibonacci(m-1) + fibonacci(m-2);
  END IF;
END fibonacci;
```

1. Recursivitate

Declaraţii *forward*

- *Subprogramele sunt reciproc recursive dacă ele se apelează unul pe altul direct sau indirect.*
- Declaraţiile *forward* permit definirea subprogramelor reciproc recursive.
- În *PL/SQL*, un identificator trebuie declarat înainte de a-l folosi.
- De asemenea, un subprogram trebuie declarat înainte de a-l apela.

1. Recursivitate

```
PROCEDURE alfa ( ... ) IS
BEGIN
  beta( ... );        -- apel incorect
  ...
END;
PROCEDURE beta ( ... ) IS
BEGIN
  ...
END;
```

- Procedura *beta* nu poate fi apelată deoarece nu este încă declarată.
- Problema se poate rezolva simplu, inversând ordinea celor două proceduri.
- Această soluţie nu este eficientă întotdeauna.

1. Recursivitate

- *PL/SQL* permite un tip special de declarare a unui subprogram numit *forward*.
- El constă dintr-o specificare de subprogram terminată prin ";".

```
PROCEDURE beta ( ... );   -- declaraţie forward
..
PROCEDURE alfa ( ... ) IS
BEGIN
  beta( ... );
  ...
END;
PROCEDURE beta ( ... ) IS
BEGIN
  ...
END;
```

1. Recursivitate

Se pot folosi declaraţii *forward:*

- pentru a defini subprograme într-o anumită ordine logică
- pentru a defini subprograme reciproc recursive
- pentru a grupa subprograme într-un pachet

1. Recursivitate

- Lista parametrilor formali din declaraţia *forward* trebuie să fie identică cu cea corespunzătoare corpului subprogramului.

- Corpul subprogramului poate apărea oriunde după declaraţia sa *forward*, dar trebuie să rămână în aceeaşi unitate de program.

Cuprins

Funcţii în PL/SQL(partea II)

1. Recursivitate
2. Utilizarea în expresii SQL a funcţiilor definite de utilizator
3. Informaţii referitoare la subprograme
4. Dependenţa subprogramelor

2. Utilizarea în expresii SQL a funcţiilor definite de utilizator

- Începând cuvarianta Oracle Release 7.1, o funcţie stocată poate fi referită într-o comandă SQL la fel ca orice funcţie standard furnizată de sistem (built-in function), dar cu anumite restricţii.

- Funcţiile PL/SQL definite de utilizator pot fi apelate din orice expresie SQL în care pot fi folosite funcţii SQL standard.

2. Utilizarea în expresii SQL a funcţiilor definite de utilizator

- Funcţiile *PL/SQL* pot să apară în:

1. lista de selecţie a comenzii *SELECT*
2. condiţia clauzelor *WHERE* şi *HAVING*
3. clauzele *CONNECT BY*, *START WITH*, *ORDER BY* şi *GROUP BY*
4. clauza *VALUES* a comenzii *INSERT*
5. clauza *SET* a comenzii *UPDATE*

2. Utilizarea în expresii SQL a funcţiilor definite de utilizator

Exemplu:

Să se afişeze salariile (nume angajat, salariu, job) a căror valoare este mai mare decât valoarea medie a tuturor salariilor din firma.

```
create or replace function "VALOARE_MEDIE2“
    return  NUMBER is
   v_val_mediu emp.sal%TYPE;
begin
  SELECT AVG(sal)
  INTO   v_val_mediu
  FROM   emp;
  RETURN v_val_mediu;
end;
```

2. Utilizarea în expresii SQL a funcţiilor definite de utilizator

Referirea acestei funcţii într-o comanda *SQL* se poate face prin:

```
SELECT ename, sal, job
FROM   emp
WHERE  sal >= valoare_medie2;
```

2. Utilizarea în expresii SQL a funcţiilor definite de utilizator

Există restricţii referitoare la folosirea funcţiilor definite de utilizator într-o comandă SQL.

Câteva dintre acestea, care s-au păstrat şi pentru versiunile ulterioare Oracle9i:

1. funcţia definită de utilizator trebuie să fie o funcţie stocată (procedurile stocate nu pot fi apelate în expresii SQL), nu poate fi locală unui alt bloc
2. funcţia definită de utilizator trebuie să fie o funcţie linie şi nu una grup (restricţia dispare în Oracle9i)
3. funcţia apelată dintr-o comandă SELECT, sau din comenzi DML: INSERT, UPDATE şi DELETE nu poate modifica tabelele bazei de date

2. Utilizarea în expresii SQL a funcţiilor definite de utilizator

4. funcţia apelată dintr-o comandă UPDATE sau DELETE nu poate interoga sau modifica tabele ale bazei reactualizate chiar de aceste comenzi (table mutating)
5. funcţia apelată din comenzile SELECT, INSERT, UPDATE sau DELETE nu poate executa comenzi LCD (COMMIT), ALTER SYSTEM, SET ROLE sau comenzi LDD (CREATE)
6. funcţia nu poate să apară în clauza CHECK a unei comenzi CREATE/ALTER TABLE
7. funcţia nu poate fi folosită pentru a specifica o valoare implicită pentru o coloană în cadrul unei comenzi CREATE/ALTER TABLE

2. Utilizarea în expresii SQL a funcţiilor definite de utilizator

8. funcţia poate fi utilizată într-o comandă SQL numai de către proprietarul funcţiei sau de utilizatorul care are privilegiul EXECUTE asupra acesteia

9. funcţia definită de utilizator, apelabilă dintr-o comandă SQL, trebuie să aibă doar parametri de tip IN, cei de tip OUT şi IN OUT nefiind acceptaţi

10. parametrii unei funcţii PL/SQL apelate dintr-o comandă SQL trebuie să fie specificaţi prin poziţie (specificarea prin nume nefiind permisă)

2. Utilizarea în expresii SQL a funcţiilor definite de utilizator

11. parametrii formali ai unui subprogram funcţie trebuie să fie de tip specific bazei de date (NUMBER, CHAR, VARCHAR2, ROWID, LONG, LONGROW, DATE), nu tipuri PL/SQL (BOOLEAN sau RECORD)

12. tipul returnat de un subprogram funcţie trebuie să fie un tip intern pentru server, nu un tip PL/SQL

13. funcţia nu poate apela un subprogram care nu respectă restricţiile anterioare

2. Utilizarea în expresii SQL a funcţiilor definite de utilizator

Exemplu de restrictie a utilizarii functiilor in comenzi SQL:

Restrictia
4. funcţia apelată dintr-o comandă UPDATE sau DELETE nu poate interoga sau modifica tabele ale bazei reactualizate chiar de aceste comenzi (table mutating)

- Comanda *UPDATE* va returna o eroare deoarece tabelul *emp* este *mutating*.

2. Utilizarea în expresii SQL a funcţiilor definite de utilizator

CALCUL

Code Dependencies Errors Grants

Save & Compile | Find & Replace | Undo | Redo | Download Source | Drop

```
1  create or replace function "CALCUL"
2  (p_sal in NUMBER)
3  return NUMBER
4  is
5  begin
6    INSERT INTO emp(empno, ename, job, hiredate, sal);
7    VALUES (1358, 'popa', SYSDATE, 700000);
8    RETURN (p_sal*7);
9  end;
```

Rows 10 Save Run

```
UPDATE  emp
SET     sal = calcul (550000)
WHERE   empno = 7531;
```

Results Explain Describe Saved SQL History

ORA-06575: Package or function CALCUL is in an invalid state

0.04 seconds

Cuprins

Funcţii în PL/SQL(partea II)

1. Recursivitate
2. Utilizarea în expresii SQL a funcţiilor definite de utilizator
3. Informaţii referitoare la subprograme
4. Dependenţa subprogramelor

3. Informaţii referitoare la subprograme

Informaţiile referitoare la subprogramele PL/SQL şi modul de acces la aceste informaţii sunt următoarele:

- codul sursă, utilizând vizualizarea USER_SOURCE din dicţionarul datelor (DD)
- informaţii generale, utilizând vizualizarea USER_OBJECTS din dicţionarul datelor
- tipul parametrilor (IN, OUT, IN OUT), utilizând comanda DESCRIBE
- erorile la compilare, utilizând vizualizarea USER_ERRORS din dicţionarul datelor sau comanda SHOW ERRORS
- informaţii de depanare, utilizând pachetul DBMS_OUTPUT.

3. Informaţii referitoare la subprograme

Vizualizarea USER_OBJECTS conţine informaţii generale despre toate obiectele manipulate în BD, în particular şi despre subprogramele stocate.

Vizualizarea USER_OBJECTS are următoarele câmpuri:

1. OBJECT_NAME – numele obiectului
2. OBJECT_TYPE – tipul obiectului (PROCEDURE, FUNCTION etc.)
3. OBJECT_ID – identificator intern al obiectului
4. CREATED – data când obiectul a fost creat
5. LAST_DDL_TIME – data ultimei modificări a obiectului
6. TIMESTAMP – data şi momentul ultimei recompilări
7. STATUS – starea obiectului (VALID sau INVALID)

3. Informaţii referitoare la subprograme

Pentru a verifica dacă recompilarea explicită (ALTER) sau implicită a avut succes se poate verifica starea subprogramelor utilizând USER_OBJECTS.

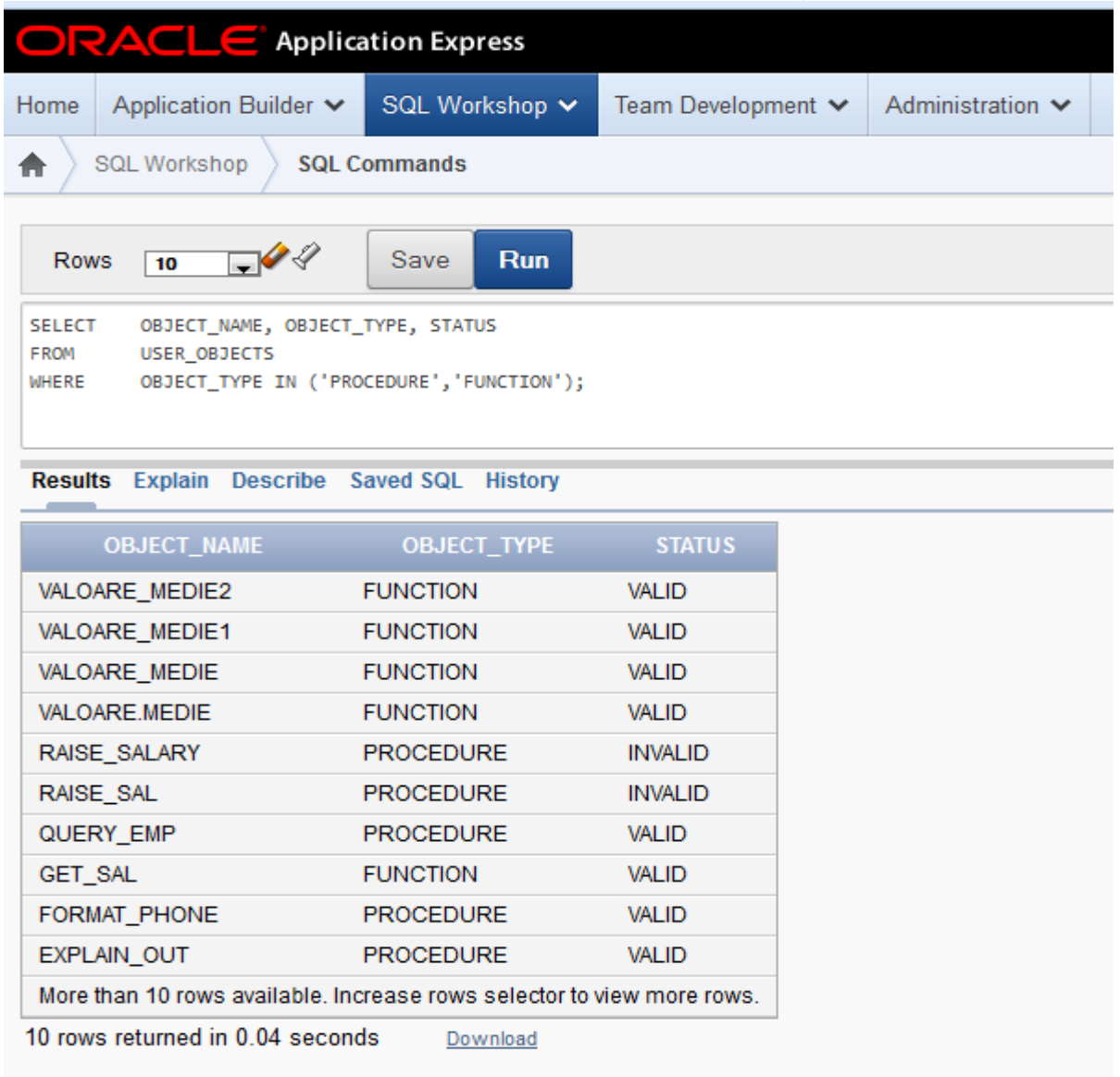
Orice obiect are o stare (status) sesizată în DD, care poate fi:

- VALID (obiectul a fost compilat şi poate fi folosit când este referit)
- INVALID (obiectul trebuie compilat înainte de a fi folosit)

3. Informaţii referitoare la subprograme

Exemplu:

Să se listeze procedurile şi funcţiile deţinute de utilizatorul curent, precum şi starea acestora.



3. Informaţii referitoare la subprograme

După ce subprogramul a fost creat, codul sursă al acestuia poate fi obţinut consultând vizualizarea USER_SOURCE din DD, care are următoarele câmpuri:

1. NAME - numele obiectului
2. TYPE - tipul obiectului
3. LINE - numărul liniei din codul sursă
4. TEXT - textul liniilor codului sursă

3. Informaţii referitoare la subprograme

Exemplu:
Să se afişeze codul complet pentru funcţia valoare_medie.

```
SELECT    TEXT
FROM      USER_SOURCE
WHERE     NAME = ‘valoare_medie’
ORDER BY  LINE;
```

3. Informaţii referitoare la subprograme

Exemplu:

Să se scrie o procedură care recompilează toate obiectele invalide din schema personală.

```
CREATE OR REPLACE PROCEDURE sterge IS
 CURSOR obj_curs IS
   SELECT  OBJECT_TYPE, OBJECT_NAME
   FROM    USER_OBJECTS
   WHERE   STATUS = 'INVALID'
   AND     OBJECT_TYPE IN ('PROCEDURE', 'FUNCTION',
  PACKAGE', 'PACKAGE BODY', 'VIEW');
BEGIN
 FOR obj_rec IN obj_curs LOOP
   DBMS_DDL.ALTER_COMPILE(obj_rec.OBJECT_TYPE,
                 USER, obj_rec.OBJECT_NAME);
 END LOOP;
END sterge;
```

3. Informaţii referitoare la subprograme

- Dacă se recompilează un obiect PL/SQL, atunci server-ul va recompila orice obiect invalid de care depinde.
- Dacă recompilarea automată implicită a procedurilor locale dependente are probleme, atunci starea obiectului va rămâne INVALID şi server-ul Oracle semnalează eroare.

3. Informaţii referitoare la subprograme

Deci:

- este preferabil ca recompilarea să fie manuală (recompilare explicită utilizând comanda ALTER: PROCEDURE, FUNCTION, TRIGGER, PACKAGE) cu opţiunea COMPILE
- este necesar ca recompilarea să se facă cât mai repede, după definirea unei schimbări referitoare la obiectele bazei

Cuprins

Funcţii în PL/SQL(partea II)

1. Recursivitate
2. Utilizarea în expresii SQL a funcţiilor definite de utilizator
3. Informaţii referitoare la subprograme
4. Dependenţa subprogramelor

4. Dependenţa subprogramelor

- Când este compilat un subprogram, toate obiectele Oracle care sunt referite vor fi înregistrate în dicţionarul datelor (DD).
- Subprogramul este dependent de aceste obiecte.
- *Un subprogram care are erori la compilare este marcat ca "invalid" în dicţionarul datelor.*
- Un subprogram stocat poate deveni, de asemenea, *invalid dacă o operaţie LDD este executată asupra unui obiect de care depinde*.

4. Dependenţa subprogramelor

Obiecte dependente	Obiecte referite
View	*Table*
Procedure	*View*
Function	*Procedure*
Package Specification	*Function*
Package Body	*Synonym*
Database Trigger	*Package Specification*

4. Dependenţa subprogramelor

Dacă se modifică definiţia unui obiect referit, obiectul dependent poate (sau nu) să continue să funcţioneze normal.

Există două tipuri de dependenţe:

1. dependenţă directă, în care obiectul dependent (procedure sau function) face referinţă direct la un table, view, sequence, procedure, function.
2. dependenţă indirectă, în care obiectul dependent (procedure sau function) face referinţă indirect la un table, view, sequence, procedure, function prin intermediul unui view, procedure sau function.

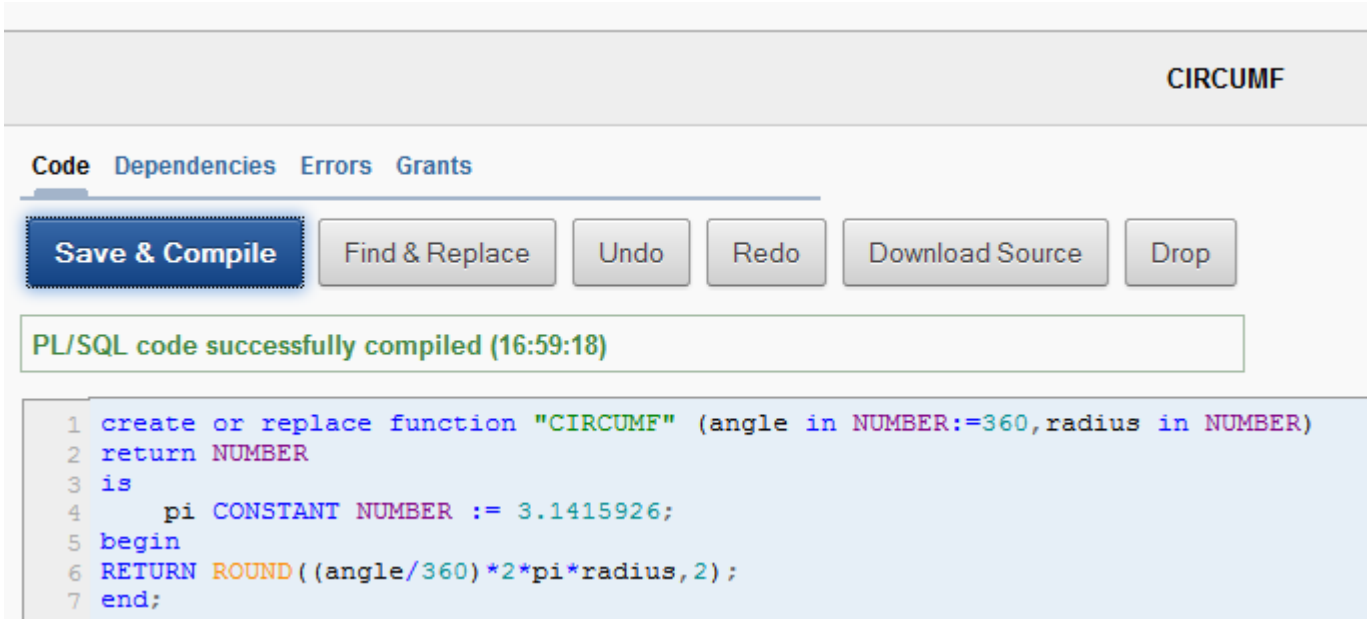
Testare

1. Exemple de funcţii
2. Probleme propuse spre rezolvare

1. Exemple de funcţii

Functie care calculeaza circumferinta unui cerc:

```
FUNCTION circumf (angle NUMBER:=360,radius
   NUMBER)
RETURN NUMBER IS
   pi CONSTANT NUMBER := 3.1415926;
BEGIN
   RETURN ROUND((angle/360)*2*pi*radius,2);
END;
```



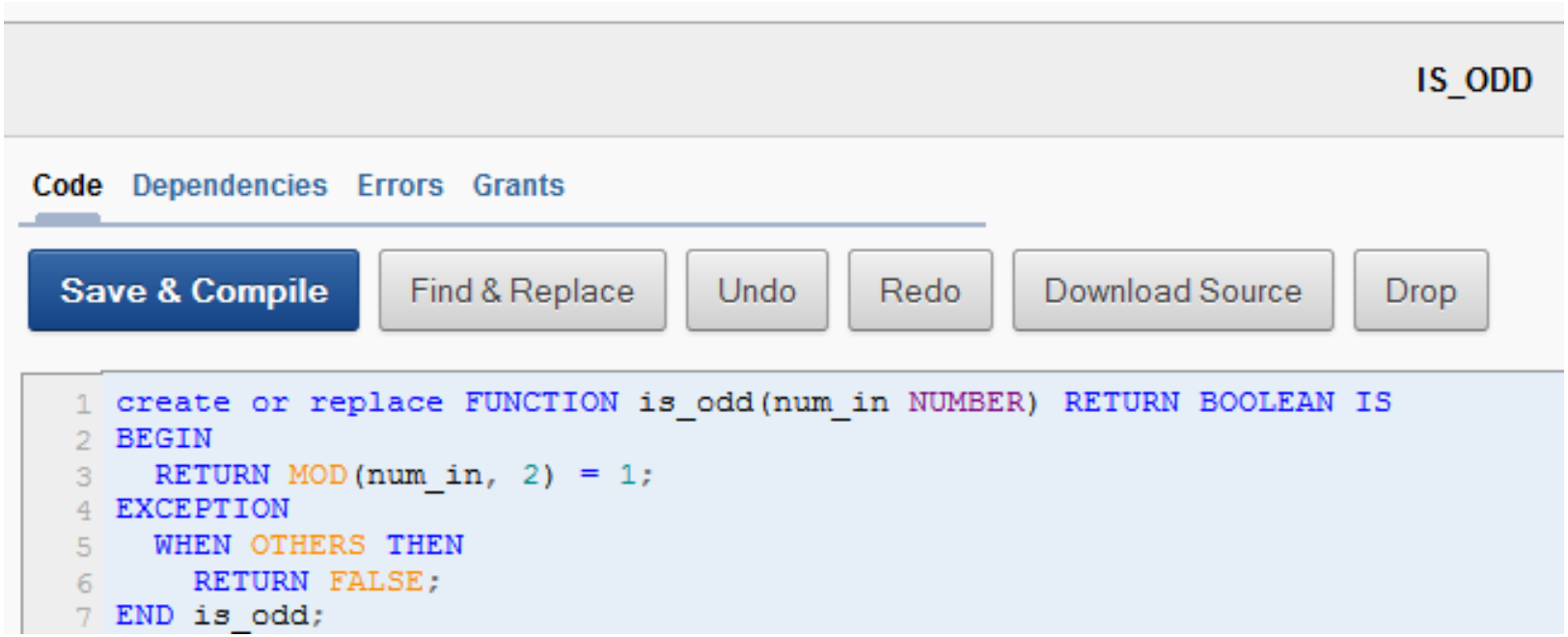
1. Exemple de funcţii

Functie care verifica daca un numar este par

```
CREATE OR REPLACE FUNCTION is_even(num_in NUMBER)
   RETURN BOOLEAN IS
BEGIN
  IF MOD(num_in, 2) = 0 THEN
    RETURN TRUE;
  END IF;
  EXCEPTION
  WHEN OTHERS THEN
    RETURN FALSE;
END is_even;
```

1. Exemple de funcţii

- Functie care verifica daca un numar este impar



Testare

1. Exemple de funcţii
2. Probleme propuse spre rezolvare

2. Probleme propuse spre rezolvare

Să se creeze tabelele ***Angajati_pnu***, ***Departamente_pnu*** (în șirul de caractere "**pnu**", ***p*** reprezintă prima literă a prenumelui, iar ***nu*** reprezintă primele două litere ale numelui dumneavoastră).

DEPARTAMENTE_PNU
department_id number(3) cheie primara (PK)
department_id varchar2(20)
manager_id varchar2(3)
location_id varchar2(100)

ANGAJATI_PNU
employee_id number(3) cheie primara (PK)
department_id number(3) referinta (FK) la tabela **DEPARTAMENTE_PNU**
last_name varchar2(40)
first_name varchar2(40)
function varchar2(25)
salary number(7)
manager_id varchar2(3)
hiredate date
commission_pct number(5)

2. Probleme propuse spre rezolvare

1. Să se declare o procedură locală într-un bloc PL/SQL anonim prin care să se introducă în tabelul DEPARTAMENTE_PNU o nouă înregistrare precizând, prin intermediul parametrilor, valori pentru toate câmpurile.

Invocaţi procedura în cadrul blocului.

Interogaţi tabelul DEPARTAMENTE_PNU şi apoi anulaţi modificările (ROLLBACK).

2. Probleme propuse spre rezolvare

2. Să se declare o procedură locală care are parametrii următori:
- p_rezultat (parametru de tip OUT) de tipul coloanei last_name din tabelul ANGAJATI_PNU;
- p_comision (parametru de tip IN) de tipul coloanei commission_pct din ANGAJATI_PNU, iniţializat cu NULL;
- p_cod (parametru de tip IN) de tipul coloanei employee_id din ANGAJATI_PNU, iniţializat cu NULL.

Dacă p_comision nu este NULL atunci în p_rezultat se va memora numele salariatului care are salariul maxim printre salariaţii având comisionul respectiv.
În caz contrar, în p_rezultat se va memora numele salariatului al cărui cod are valoarea dată la apelarea procedurii.

2. Probleme propuse spre rezolvare

3. Să se creeze o procedură stocată fără parametri care afişează un mesaj “Programare PL/SQL”, ziua de astăzi în formatul DD-MONTH-YYYY şi ora curentă, precum şi ziua de ieri în formatul DD-MON-YYYY.

2. Probleme propuse spre rezolvare

4. Să se creeze o procedură stocată care pentru un anumit cod de departament (dat ca parametru) calculează prin intermediul unor funcţii locale numărul de salariaţi care lucrează în el, suma salariilor şi numărul managerilor salariaţilor care lucrează în departamentul respectiv.

2. Probleme propuse spre rezolvare

5. Să se creeze o funcţie stocată care determină numărul de salariaţi din problema4_pnu angajaţi după 1995, într-un departament dat ca parametru.

Să se apeleze această funcţie într-un bloc PL/SQL.

2. Probleme propuse spre rezolvare

6. Să se calculeze recursiv numărul de permutări ale unei mulţimi cu n elemente, unde n va fi transmis ca parametru.

2. Probleme propuse spre rezolvare

7. Să se afişeze numele, job-ul şi salariul angajaţilor al căror salariu este mai mare decât media salariilor din tabelul angajati_pnu.

Întrebări?