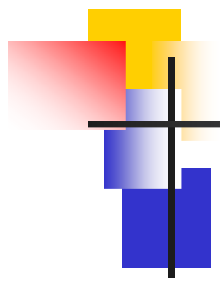




# CURS 8

---

PL/SQL



# Introducere

---

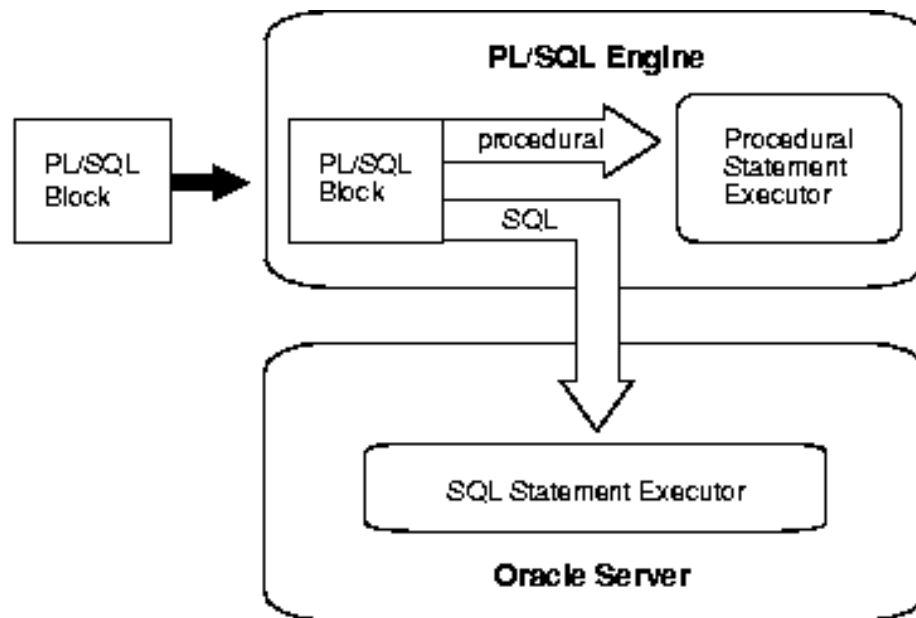
- PL/SQL reprezintă extensia procedurală pentru SQL în Oracle
- Utilizarea PL/SQL permite:
  - Folosirea SQL pentru a manipula date Oracle
  - Declararea de constante și variabile
  - Definirea de proceduri și funcții
  - Manipularea erorilor
- Combină puterea de manipulare a datelor din SQL cu puterea de prelucrare a limbajelor procedurale



# Arhitectura PL/SQL

- PL/SQL este o tehnologie si nu un produs independent
  - Poate fi gândit ca un motor care compilează și execută blocuri și subprograme PL/SQL
- Poate fi instalat:
  - într-un server Oracle sau
  - într-un instrument de dezvoltare a aplicațiilor (Oracle Forms sau Oracle Reports).
- Cele doua medii în care poate exista PL/SQL sunt indepenente

- In orice situație motorul PL/SQL acceptă ca intrări orice bloc sau subprogram valid
- Motorul PL/SQL execută instrucțiunile procedurale dar trimite către serverul Oracle instrucțiunile SQL





# Avantaje

---

- PL/SQL este un limbaj de procesare a tranzacțiilor de înaltă performanță, complet portabil
- Avantaje:
  - Suport pentru SQL
  - Suport pentru programarea orientată obiect
  - Performanțe îmbunătățite
  - Productivitate crescută
  - Portabilitate completă
  - Integrare completă cu Oracle
  - Securitate sporită



# Suport pentru SQL

- PL/SQL permite utilizarea tuturor instrucțiunilor SQL de manipulare a datelor, comenzi de control al cursorilor și control al tranzacțiilor, funcții SQL operatori și pseudocoloane
  - pot fi manipulate flexibil și sigur date Oracle
  - PL/SQL suporta setul complet de tipuri de date SQL
    - Se elimină necesitatea de a converti datele transmise între aplicație și baza de date
- PL/SQL suportă SQL dinamic –programele pot construi și prelucra instrucțiuni de definire a datelor, de control al datelor și de control al sesiunilor în timpul execuției ("on the fly" )

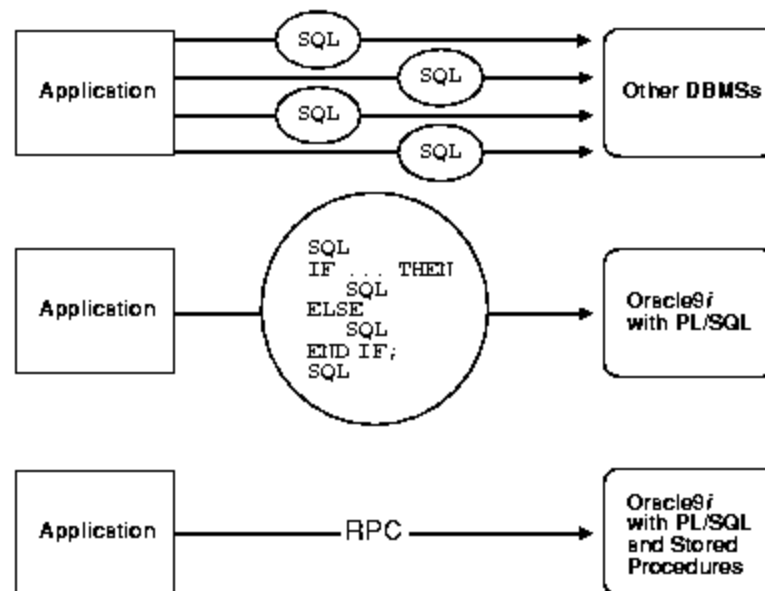


# Suport pentru programarea orientată pe obiecte

- Tipurile obiect sunt un instrument ideal de modelare orientată-obiect utile pentru a reduce costul și timpul necesar pentru a construi aplicații complexe
  - Permit crearea componentelor software modulare, care pot fi întreținute și reutilizabile
  - Permit echipelor de programatori să dezvolte concurrent componente software
- Prin încapsularea operațiilor cu datele, tipurile obiect permit transferarea codului de întreținere a datelor în afara scripturilor SQL sau a blocurilor PL/SQL către metode.
- Tipurile obiect ascund detaliile de implementare → se pot schimba aceste detalii fără a afecta programele client.
- Tipurile obiect permit o abstractizare mai apropiată de realitate și programele reflectă mai corect lumea ce se dorește simulată

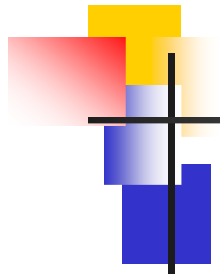
# Performanțe crescute 1

- fără PL/SQL, Oracle trebuie să prelucreze instrucțiunile SQL pe rând.
  - Fiecare instrucțiune SQL apelează Oracle ceea ce conduce la performanțe reduse
    - mai ales în condițiile în care se lucrează în rețea traficul este foarte ridicat



- cu PL/SQL, la un moment dat va fi trimis către Oracle un bloc întreg – ceea ce poate reduce drastic comunicația dintre aplicație și Oracle
- pentru aplicațiile care accesează intens baza de date se pot utiliza blocuri SQL și subprograme pentru a grupa instrucțiunile SQL înainte de a le trimite către Oracle pentru a fi executate.





## Performanțe crescute 2

---

- Procedurile stocate PL/SQL sunt compilate o singură dată și sunt memorate în format executabil → apelurile sunt rapide și eficiente.
- PL/SQL îmbunătățește performanțele prin adăugarea posibilităților de prelucrare procedurală instrumentelor Oracle

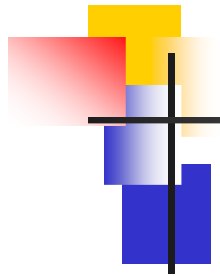


# *Productivitate ridicată*

---

PL/SQL adaugă funcționalitate instrumentelor non-procedurale precum Oracle Forms și Oracle Reports.

- În aceste instrumente pot fi folosite construcțiile procedurale din PL/SQL pentru a construi aplicații.
- Ex. se poate utiliza un bloc PL/SQL complet într-un trigger în Oracle Forms
  - PL/SQL crește productivitatea prin furnizarea unor instrumente mai eficiente
- PL/SQL este același în toate mediile
  - Scripturile scrise cu un instrument pot fi utilizate și de altele



## *Portabilitate completă*

---

- Aplicațiile scrise în PL/SQL sunt portabile pe orice sistem de operare și platformă pe care rulează Oracle.
  - Se pot scrie biblioteci de programe portabile care pot fi reutilizate în diverse medii.



# *Integrare strânsă cu Oracle*

- PL/SQL si SQL sunt strâns integrate
- PL/SQL suporta toate tipurile de date SQL plus NULL.
  - permite manipularea datelor Oracle simplu și eficient si ajută la scrierea de cod de mare performanță
- attributele %TYPE și %ROWTYPE integrează PL/SQL cu SQL
  
- Exemplu: Se poate utiliza atributul %TYPE pentru a declara variabile pe baza definiția coloanei din baza de date.
  - dacă aceasta definiție se schimbă declararea variabilei se schimbă corespunzător la următoarea rulare a programului.
  - aceasta este o formă de manifestare a independenței datelor
  - se reduc costurile de întreținere
  - permit programelor adaptarea la schimbările din baza de date



# *Securitate sporită*

---

- Procedurile stocare PL/SQL permit partiționarea logicii aplicației între client și server
  - Se poate evita ca aplicațiile client să manipuleze datele Oracle sensibile
  - Trigger-ele scrise în PL/SQL pot restricționa, selectiv, posibilitățile de actualizare ale aplicațiilor și pot face un audit bazat pe conținut pentru inserările de date.
- Se poate restricționa accesul la date dacă se permite ca utilizatorii să aibă le manipuleze numai prin intermediul procedurilor stocate pe care le pot executa doar conform privilegiilor pe care le dețin.
  - Se poate permite utilizatorilor accesul la o procedură care actualizează un tabel, dar nu și accesul la tabelul însuși



# Structura programelor PL/SQL

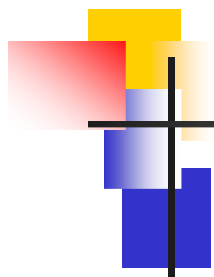
- Limbaj structurat pe blocuri
- Unitatea de bază – blocul logic, care poate conține orice număr de blocuri înglobate
  - în mod normal fiecare bloc corespunde unei probleme de rezolvat
  - PL/SQL folosește pentru o problemă o abordare de tip divide –și-cucerește -> rafinare pas cu pas
- Blocul (sub blocul) permite gruparea logică a declarațiilor și instrucțiunilor înrudite
  - se pot plasa declarațiile cât mai aproape de locul în care sunt folosite
  - declarațiile sunt locale
- Un bloc are trei părți componente:
  - O secțiune declarativă
  - O secțiune executabilă
  - O secțiune de manipulare a excepțiilor
- Un bloc poate îngloba alte sub blocuri în secțiunea executabilă sau de manipulare a excepțiilor
- Nu poate îngloba sub-blocuri în secțiunea declarativă
- În secțiunea declarativă se pot defini subprograme locale, însă acestea se pot apela numai în blocul în care au fost definite



# Structura programelor PL/SQL

---

- DECLARE (opțional)
  - Variabile, cursoare, excepții definite de utilizator
- BEGIN (obligatoriu)
  - Instrucțiuni SQL
  - Instrucțiuni PL/SQL
- EXCEPTION (opțional)
  - Acțiuni ce trebuie efectuate când apar erori
- END; (obligatoriu)



# Tipuri de blocuri

## Anonim

```
[DECLARE]

BEGIN
...instrucțiuni
[EXCEPTION]

END;
```

## Procedură

```
PROCEDURE
nume_proc IS

BEGIN
...instrucțiuni
[EXCEPTION]

END;
```

## Funcție

```
FUNCTION
nume_func RETURN
tip_data

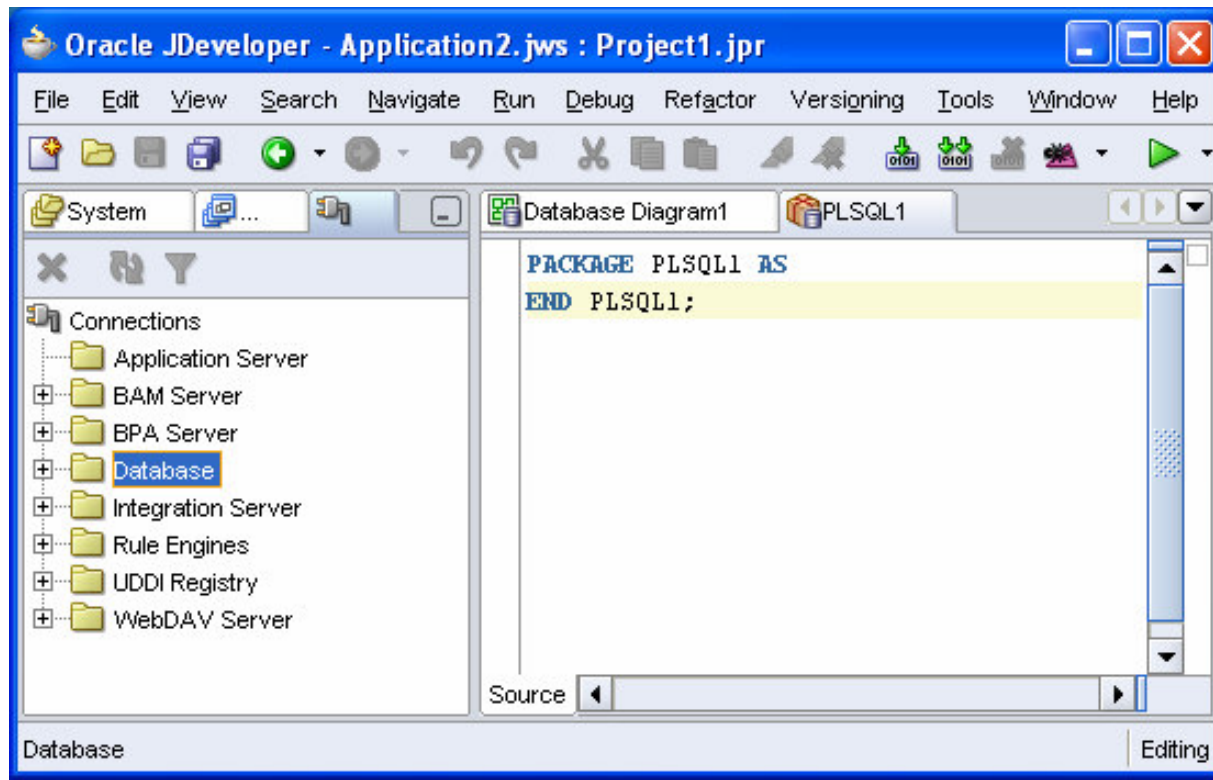
BEGIN
...instrucțiuni
[EXCEPTION]

END;
```



# Medii de programare PL/SQL

- *iSQL\*Plus*
- *jDeveloper*





# Unități lexicale în blocul PL/SQL

- Identificatori – nume atribuite obiectelor PL/SQL
- Delimitatori – simboluri cu o semnificație aparte
  - Ex. simbolul ‘;’
- Literalii – orice valoare care este asignată unei variabile
  - Pot fi:
    - caractere
    - date numerice
    - date de tip boolean
- Comentarii –
  - pe o singura linie, comentariul este precedat de simbolul ‘- -’
  - pe mai multe linii – comentariul este lăsat între perechile de simboluri ‘/\*’ și ‘\*/’.

# Declararea și inițializarea variabilelor în PL/SQL

- Trebuie declarate după care pot fi utilizate în SQL sau în secvențele procedurale
- Este necesar ca declarația să fie făcută înaintea folosirii lor în alte instrucțiuni (inclusiv în alte declarații)
- Pot avea orice tip de dată SQL (CHAR, DATE, NUMBER... ) sau PL/SQL (BOOLEAN sau BINARY-INTEGER)
- Se pot declara și ca tipuri de date compozite - nested tables, varray sau records prin utilizarea cuvintelor cheie TABLE, VARRAY și RECORD

Sintaxa:

*identificator* [CONSTANT] *tip\_data* [NOT NULL]  
[:= | DEFAULT *expresie*];



# Variabile - Exemple

```
DECLARE
v_data_ang  DATE;
v_dep_id    NUMBER (2) NOT NULL :=10;
v_sal       REAL(7,2);
V_loc       CONSTANT VARCHAR2(20) :='Suceava';
```

```
SET SERVEROUTPUT ON
DECLARE
    v_num     VARCHAR2 (20);
BEGIN
    DBMS_OUTPUT.PUT_LINE ('Numele meu este: ' || v_num);
    v_num := 'Adriana';
    DBMS_OUTPUT.PUT_LINE ('Numele meu este: ' || v_num);
END;
/
```



# Atribuirea de valori variabilelor

- Atribuirea valorilor se poate face în trei moduri:
  - Utilizarea operatorului ‘:=’  
val:= cant \* pret\_unit
  - Extragerea valorilor din baza de date în variabilă  
SELECT salary\*0.1 INTO bonus  
FROM employees  
WHERE employee\_id=100
  - Considerarea variabilei ca un parametru OUT sau IN OUT într-un subprogram

DECLARE

v\_sal REAL(7,2);

PROCEDURE modif\_salary (emp\_id INT, salary IN OUT REAL) IS ...

BEGIN

SELECT AVG(salary) INTO v\_sal FROM employees;

modif\_salary(100, v\_sal);

-- atribuie o valoare nouă variabilei v\_sal



# Variabile legate (bind)

- Sunt create într-un mediu gazdă, motiv pentru care mai sunt denumite variabile *host*
- Sunt utilizate în instrucțiuni SQL și în blocuri PL/SQL
- Sunt accesate și după ce blocul PL/SQL a fost executat
- La creare se utilizează cuvântul cheie VARIABLE
  - pot fi create în SQL\*Plus sau iSQL\*Plus
- Sunt referite prin precedarea numelui cu simbolul (:)
- iSQL\*Plus poate afișa valoarea unei variabile legate prin comanda PRINT

# Exemplu 1

Introducere instrucțiuni:

```
variable rezultat number  
  
Begin  
    select (salary*12) into :rezultat  
    from employee where employee_id =144;  
end;  
/  
print rezultat
```

Execuție

Salvare script

Ștergere ecran

Renunțare

Procedură PL/SQL încheiată cu succes.

REZULTAT

30012

## Exemplu 2

Fișier sau URL:

Introducere instrucțiuni:

```
variable emp_sal number  
  
Begin  
    select salary into :emp_sal  
    from employee where employee_id =178;  
end;  
/  
print emp_sal  
select first_name, last_name from employee  
where salary =:emp_sal;
```

Procedură PL/SQL încheiată cu succes.

EMP_SAL	
	7001

FIRST_NAME	LAST_NAME
Kimberely	Grant



# Exemplul 3

Fișier sau URL:

Introducere instrucțiuni:

```
variable emp_sal number  
  
set autoprint on  
  
Begin  
    select salary into :emp_sal  
        from employee where employee_id =178;  
end;  
/
```

Procedură PL/SQL încheiată cu succes.

EMP_SAL
7001

# Substituire variabilelor 1

- Presupune introducerea valorilor de către utilizator in timpul execuției
- In blocurile PL/SQL se folosește semnul (&)

Introducere instrucțiuni:

```
variable emp_sal number  
set autoprint on  
DECLARE  
    empl_id NUMBER :=&empl_id;  
Begin  
    select salary into :emp_sal  
    from employee where employee_id =empl_id;  
end;  
/
```

ORACLE  
iSQL\*Plus

[Deconectare](#) [Sesiune nouă](#) [Renunțar](#)

[Ecran de lucru](#) > Variabile de substituție

### Variabile de substituție

Introduceți în script valori pentru variabilele de substituție în vederea execuției:

Variabilă	Valoare
empl_id	101

vechi 2: empl\_id NUMBER :=&empl\_id;  
nou 2: empl\_id NUMBER :=101;  
Procedură PL/SQL încheiată cu succes.

EMP_SAL
24890,7

OK Renunțare



# Blocuri imbricate în PL/SQL

---

- Blocurile PL/SQL pot fi imbricate în
  - Secțiunea executabilă (BEGIN...END) a unui bloc
  - Secțiunea de manipulare a excepțiilor



# Exemplu

```
SET SERVEROUTPUT ON  
DECLARE
```

```
    var_ext    VARCHAR2(20) := 'variabila globala';  
BEGIN
```

```
    DECLARE
```

```
        var_int VARCHAR2(20) := 'variabilă locala';
```

```
    BEGIN
```

```
        DBMS_OUTPUT.PUT_LINE(var_int);
```

```
        DBMS_OUTPUT.PUT_LINE(var_ext);
```

```
    END;
```

```
    DBMS_OUTPUT.PUT_LINE(var_ext);
```

```
END;
```

```
/
```



# Domeniul si vizibilitatea variabilelor

```
set serveroutput on
```

```
DECLARE
```

```
  nume_p VARCHAR2(15):='Mihai';
```

```
  ddn date := '20-mar-1970';
```

```
BEGIN
```

```
  DECLARE
```

```
    nume_c VARCHAR2(15):='Irina';
```

```
    ddn date := '01-mai-1994';
```

```
BEGIN
```

```
  DBMS_output.put_line('numele tatalui este: '|| nume_p);
```

```
  DBMS_output.put_line('data nasterii: '|| ddn);
```

```
  DBMS_output.put_line('numele copilului este: '|| nume_c);
```

```
END;
```

```
DBMS_output.put_line('data nasterii:'|| ddn);
```

```
END;
```

Execuție

Salvare script

Ștergere ecran

```
numele tatalui este: Mihai  
data nasterii: 01-05-1994  
numele copilului este: Irina  
data nasterii:20-03-1970  
Procedură PL/SQL încheiată cu succes.
```

# Calificarea unui identificador care apare în mai multe blocuri imbricate

- Calificarea presupune utilizarea unei etichete asociate unui bloc
  - Etichetarea nu este limitată – orice bloc poate fi etichetat
- calificatorii se pot folosi pentru a accesa variabile care deși sunt în domeniu nu sunt vizibile

# Calificarea unui identificador care apare în mai multe blocuri imbricate



```
set serveroutput on
```

```
<<outer>>
```

```
DECLARE
```

```
  nume_p VARCHAR2(15):='Mihai';
```

```
  ddn date := '20-mar-1970';
```

```
BEGIN
```

```
  DECLARE
```

```
    nume_c VARCHAR2(15):='Irina';
```

```
    ddn date := '01-mai-1994';
```

```
  BEGIN
```

```
    DBMS_output.put_line('numele tatalui este: '|| nume_p);
```

```
    DBMS_output.put_line('data nasterii: '|| outer.ddn);
```

```
    DBMS_output.put_line('numele copilului este: '|| nume_c);
```

```
    DBMS_output.put_line('data nasterii: '|| ddn);
```

```
  END;
```

```
END;
```

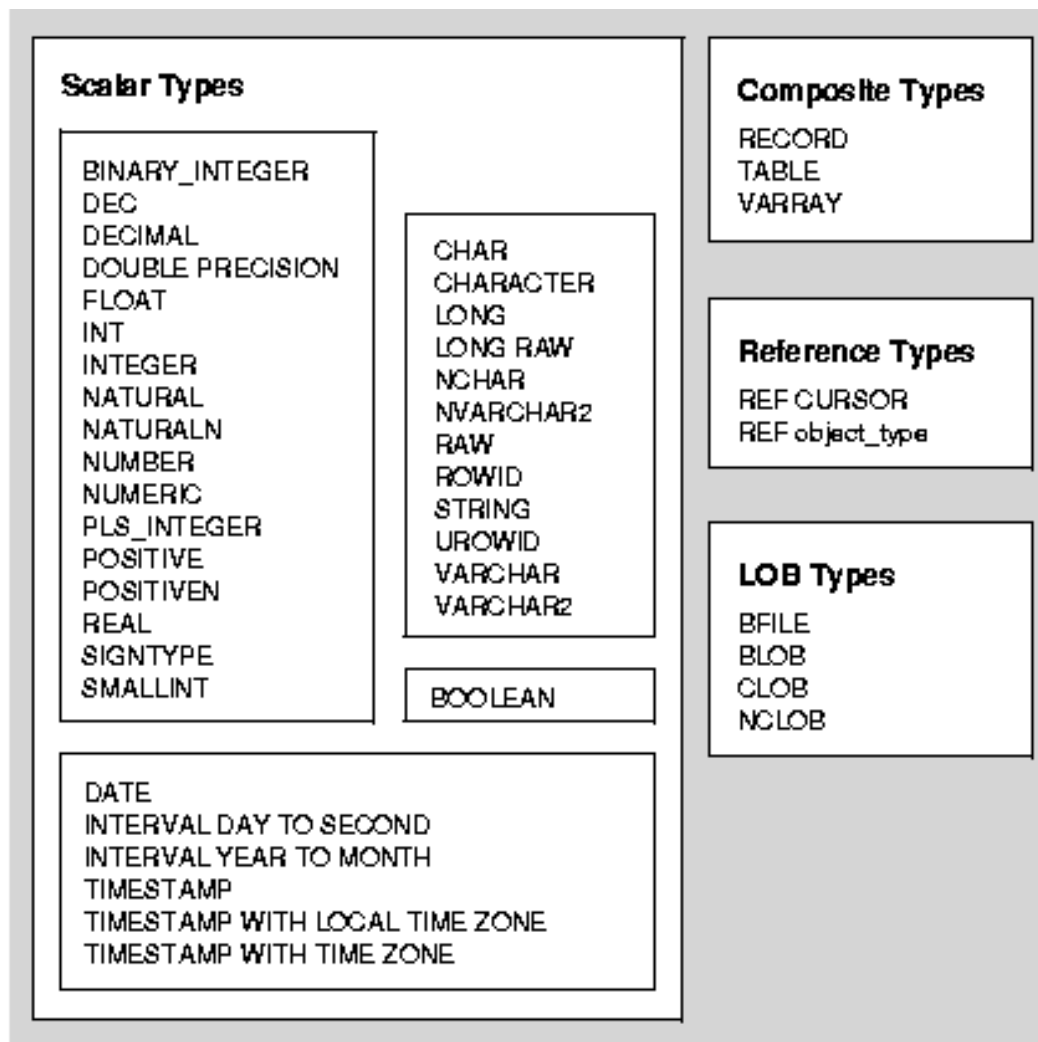
Execuție

Salvare script

Și

```
numele tatalui este: Mihai
data nasterii: 20-03-1970
numele copilului este: Irina
data nasterii: 01-05-1994
Procedură PL/SQL încheiată cu succes.
```

# Tipuri de date predefinite in PL/SQL







# Tipuri de date compozite

---

- TABLE
- ARRAY
- **RECORD**



# Abstractizarea datelor – tipul RECORD – definire și declarare

- Pentru a crea înregistrări (records)
  1. se definește un tip de data RECORD
  2. se declară înregistrări cu tipul respectiv
    - se pot defini tipuri RECORD în secțiunea declarativă a oricărui bloc PL/SQL, subprogram sau pachet

TYPE type\_name IS RECORD (field\_declaration[,field\_declaration]...);

unde *field\_declaration* conține

- field\_name field\_type [[NOT NULL] {:= | DEFAULT} expr]
  - type\_name specifică tipul ce va fi folosit ulterior pt a declara înregistrările
  - field\_type este orice tip e data PL/SQL cu excepția tipului REF CURSOR,
  - expr furnizează o valoare de același tip ca field\_type.
- **Notă:** spre deosebire de tipurile VARRAY si (nested) TABLE tipurile RECORD nu pot fi create si stocate în baza de date.



# Abstractizarea datelor – tipul RECORD – definire și declarare

- Se poate utiliza unul din attributele %TYPE sau %ROWTYPE pentru a specifica tipul unui câmp.

EXEMPLU:

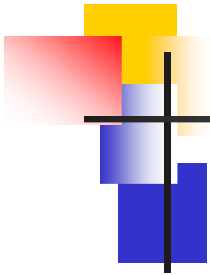
DECLARE

```
TYPE DeptRec IS RECORD (  
    dept_id dept.deptno%TYPE,  
    dept_name VARCHAR2(14),  
    dept_loc VARCHAR2(13));
```

BEGIN

...

END;



PL/SQL permite definirea de înregistrări care conțin obiecte, colecții, și alte înregistrări ( *nested* records).

- In acest caz obiectele nu pot avea atribute de tip RECORD.

DECLARE

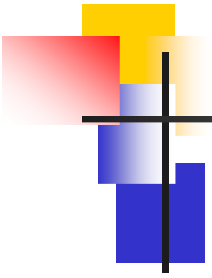
```
TYPE TimeRec IS RECORD (  
    seconds SMALLINT,  
    minutes SMALLINT,  
    hours SMALLINT);
```

```
TYPE FlightRec IS RECORD (  
    flight_no INTEGER,  
    plane_id VARCHAR2(10),  
    captain Employee,          -- poate fi declarat anterior ca obiect  
    passengers PassengerList,  -- - '' -          varray  
    depart_time TimeRec,       -- - '' -          nested record  
    airport_code VARCHAR2(10));
```

BEGIN

...

END;

- 
- Se poate specifica un tip RECORD in clauza RETURN din specificarea unei funcții.
    - Permite funcției să întoarcă o înregistrare de același tip cu o înregistrare definită de utilizator

DECLARE

```
TYPE EmpRec IS RECORD (  
    emp_id NUMBER(4) 1  
    ast_name VARCHAR2(10),  
    dept_num NUMBER(2),  
    job_title VARCHAR2(9),  
    salary NUMBER(7,2));
```

...

```
FUNCTION nth_highest_salary (n INTEGER) RETURN EmpRec IS
```

...

```
BEGIN
```

...

```
END;
```



# Declararea datelor de tip RECORD

- Odata ce un tip RECORD a fost definit, se pot declara înregistrări de acest tip

```
DECLARE
TYPE StockItem IS RECORD (
    item_no INTEGER(3),
    description VARCHAR2(50),
    quantity INTEGER,
    price REAL(7,2));
    item_info StockItem;          -- declară o înregistrare
BEGIN

...
END;
```

- Identificatorul item\_info reprezintă o întreagă înregistrare



# Declararea datelor de tip RECORD

- Inregistrările definite de utilizator pot fi declarate ca parametri formali pentru proceduri si funcții:

```
DECLARE
TYPE EmpRec IS RECORD (
    emp_id emp.empno%TYPE,
    last_name VARCHAR2(10),
    job_title VARCHAR2(9),
    salary NUMBER(7,2));
...
PROCEDURE raise_salary (emp_info EmpRec);
BEGIN
...
END;
```



# Inițializarea datelor RECORD

---

- ...poate fi făcută în definiția tipului
- Exemplu:

```
DECLARE
TYPE TimeRec IS RECORD (
    secs SMALLINT := 0,
    mins SMALLINT := 0,
    hrs SMALLINT := 0);
BEGIN
...
END;
```





# Inițializarea datelor RECORD

- Se pot impune constrângeri not null asupra oricăror câmpuri.
  - In acest caz, câmpurile trebuie inițializate

DECLARE

```
TYPE StockItem IS RECORD (  
    item_no INTEGER(3) NOT NULL := 999,  
    description VARCHAR2(50),  
    quantity INTEGER,  
    price REAL(7,2));
```

BEGIN

...

END;



# Referirea înregistrărilor

---

- Câmpurile unei înregistrări sunt accesate prin nume.

- Se folosește notația dot:

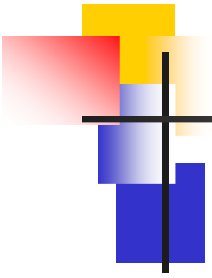
**record\_name.field\_name**

- Exemplu:

**emp\_info.hire\_date ...**

- Cand se apelează o funcție care întoarce o înregistrare definită de utilizator, se folosește următoarea sintaxă pentru a referi câmpurile din înregistrare:

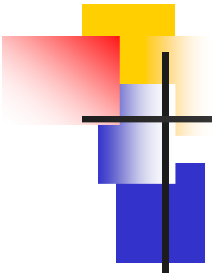
**function\_name(parameter\_list).field\_name**



```

DECLARE
TYPE EmpRec IS RECORD (
    emp_id NUMBER(4),
    job_title VARCHAR2(9),
    salary NUMBER(7,2));
    middle_sal NUMBER(7,2);
FUNCTION nth_highest_sal (n INTEGER) RETURN EmpRec IS
    emp_info EmpRec;
BEGIN
    ...
    RETURN emp_info;                                -- return record
END;
BEGIN
middle_sal := nth_highest_sal(10).salary;           -- call function
...
END;
■   Dacă funcția nu are parametri se folosește:
function_name().field_name                        -- note empty parameter list

```

- 
- Pentru a referi câmpuri *nested* într-o înregistrare întoarsă de o funcție se folosește notația dot extinsă:

**function\_name(parameter\_list).field\_name.nested\_field\_name**

- Exemplu:

```
DECLARE
```

```
TYPE TimeRec IS RECORD (
```

```
    minutes SMALLINT,
```

```
    hours SMALLINT);
```

```
TYPE AgendaItem IS RECORD (
```

```
    priority INTEGER,
```

```
    subject VARCHAR2(100),
```

```
    duration TimeRec);
```

```
FUNCTION item (n INTEGER) RETURN AgendaItem IS
```

```
    item_info AgendaItem;
```

```
    BEGIN
```

```
        .. .
```

```
        RETURN item_info; -- return record
```

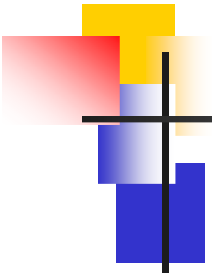
```
    END;
```

```
BEGIN
```

```
    ...
```

```
IF item(3).duration.minutes > 30 THEN ... -- call function
```

```
END;
```

- 
- Aceeași notație este folosită pentru a referi attributele unui obiect stocat într-un câmp:

```
DECLARE
```

```
TYPE FlightRec IS RECORD (  
    flight_no INTEGER,  
    plane_id VARCHAR2(10),  
    captain Employee,           -- declare object  
    passengers PassengerList,   -- declare varray  
    depart_time TimeRec,        -- declare nested record  
    airport_code VARCHAR2(10));  
    flight FlightRec;
```

```
BEGIN
```

```
...
```

```
IF flight.captain.name = 'H Rawlins' THEN ...
```

```
END;
```



# Asignarea de valori null unei înregistrări

- Pentru a seta toate câmpurile unei înregistrări ca null, i se asignează acestuia o înregistrare neinițializată, de același tip:

```
DECLARE
TYPE EmpRec IS RECORD (
    emp_id    emp.empno%TYPE,
    job_title  VARCHAR2(9),
    salary     NUMBER(7,2));
    emp_info   EmpRec;
    emp_null   EmpRec;
BEGIN
    emp_info.emp_id := 7788;
    emp_info.job_title := 'ANALYST';
    emp_info.salary := 3500;
    emp_info := emp_null;  -- nulls all fields in emp_info
    ...
END;
```



# Atribuirea de valori înregistrărilor

- Unui anumit câmp dintr-o înregistrare i se poate asocia valoarea unei expresii:

`record_name.field_name := expr;`

Exemplu: `emp_info.ename := UPPER(emp_info.ename);`

- Se pot asigna valori, deodată, tuturor câmpurilor dintr-o înregistrare în două moduri
  - Se poate atribui o înregistrare definită de utilizator unei alteia, dacă au același tip de date
    - Nu este suficient să existe câmpuri care corespund exact Consider the following example:
  - Second, you can use the SELECT or FETCH statement to fetch column values into a record, as the example below shows. The columns in the select-list must appear in the same order as the fields in your record.

Exemplu

DECLARE

TYPE DeptRec IS RECORD (  
    dept\_num NUMBER(2),  
    dept\_name VARCHAR2(14));

TYPE DeptItem IS RECORD (  
    dept\_num NUMBER(2),  
    dept\_name VARCHAR2(14));

dept1\_info DeptRec;

dept2\_info DeptItem;

BEGIN ...

`dept1_info := dept2_info;`

END;

-- gresit, deoarece sunt tipuri de date diferite



## Atribuirea de valori înregistrărilor 2

- Se poate atribui o înregistrare declarată cu %ROWTYPE unei înregistrări definite de utilizator dacă, câmpurile celor două corespund ca ordine și număr, iar câmpurile corespondente au tipuri de date compatibile:

```
DECLARE
TYPE DeptRec IS RECORD (
    dept_num NUMBER(2),
    dept_name VARCHAR2(14),
    location VARCHAR2(13));
dept1_info DeptRec;
dept2_info dept%ROWTYPE;
BEGIN
SELECT * INTO dept2_info FROM dept WHERE deptno = 10;
    dept1_info := dept2_info;
...
END;
```





## Atribuirea de valori înregistrărilor 3

- Utilizarea unei instrucțiuni SELECT sau FETCH pentru a extrage valorile coloanelor într-o înregistrare

```
DECLARE
```

```
TYPE DeptRec IS RECORD (
```

```
    dept_num NUMBER(2),
```

```
    dept_name VARCHAR2(14),
```

```
    location VARCHAR2(13));
```

```
dept_info DeptRec;
```

```
BEGIN
```

```
SELECT * INTO dept_info FROM dept WHERE deptno = 20;
```

```
...
```

```
END;
```

# Atribuirea de valori înregistrărilor 4

- NU este posibilă asignarea unei liste de valori unei înregistrări printr-o instrucțiune de atribuire.
- Ex. următoarea sintaxă este INCORECTĂ  
**record\_name := (value1, value2, value3, ...);**
- Se poate atribui o înregistrare nested alteia, dacă au același tip de date. Acest lucru este permis chiar dacă înregistrările încapsulate au tipuri de date diferite.

```
DECLARE
TYPE TimeRec IS RECORD (
    mins  SMALLINT,
    hrs   SMALLINT);
TYPE MeetingRec IS RECORD (
    day    DATE,
    time_of TimeRec,           -- nested record
    room_no INTEGER(4));
TYPE PartyRec IS RECORD (
    day    DATE,
    time_of TimeRec,           -- nested record
    place VARCHAR2(25));
Seminar    MeetingRec;
Party      PartyRec;
BEGIN
...
party.time_of := seminar.time_of;
END;
```