

## TIPI RECORD

- Sintassi:
  - `TYPE mioTipoRecord IS RECORD (field[,field]*);`
  - `field ::= nome tipo [ [NOT NULL] := expr ]`
- I campi possono essere scalari, record, collezioni
- Sono tipi di prima classe
- I campi si aggiornano e si leggono con la sintassi `rec.campo`
- Due record dello stesso tipo (cioè con lo stesso nome di tipo) si possono assegnare per intero

## SELECT INTO

- Se una query ritorna una sola riga, si può metterne il risultato dentro un record o un insieme di campi:

```
type impiegato emp%ROWTYPE;
opp: type impiegato is record(a number, b char(30));
unImp impiegato;
x number; y char(30);
...
select * into unImp
from emp where codice=100;
```
- ma anche:
  - `select codice, nome into x, y`
  - `select * into x, y`
  - `select codice, nome into unImp`

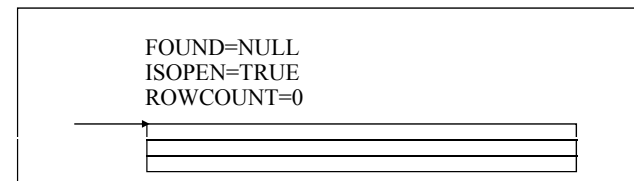
## SELECT INTO

- `select ... into ...` fallisce se la query dà n righe con  $n \neq 1$
- Per evitare problemi:

```
- select count(*) into i
from ... ecc.;
if i=1 then select ...;
else ....;
end if;
```

## CURSORI

- Un cursore è associato ad una query
- Dopo che subisce un OPEN, denota un'area di lavoro:



- Ogni operazione `fetch c into var` legge una riga ed avanza il puntatore; dopo l'ultima riga, l'effetto della `fetch` su `var` è indefinito (ma non fallisce)
- Dopo la OPEN, FOUND = Null; dopo le prime tre FETCH, FOUND=True; dopo la quarta FETCH, FOUND=False

## CURSORI

- Un cursore è associato ad una query dentro le dichiarazioni; può avere parametri:

```
1 illogin varchar2;
  cursor c is
    select ora, data
    from prenotazioni where login = illogin;
2 cursor c(nome varchar2) is
  select ora, data
  from prenotazioni where login = nome;
```

in (1), `illogin` è valutata al momento della `open` del cursore.

## Operazioni sul cursore:

- `open`: esegue la query
- `fetch c into <dest>`
  - `dest`: o una lista di variabili, o un record
- estrazione di attributi: `c%FOUND`, `c%NOTFOUND`, `c%ISOPEN`, `c%ROWCOUNT`,
- `close c`: libera il risultato; si può riaprire.

## CURSORI NEI FOR

- Se `c` è un cursore, allora:

```
for x in c loop
  body(x);
end loop;
```

equivale (più o meno) a:

```
declare
  x c%rowtype;
begin
  open c;
  loop
    fetch c into x;
    exit when c%notfound;
    body(x);
  end loop;
  close c;
end;
```

## CURSORI IMPLICITI

- Se `c` è un cursore, allora:

```
for x in ( query ) loop
  body(x);
end loop;
```

equivale a:

```
for x in c loop
  body(x);
end loop;
```

## Esempio di cursore implicito

```
GUI.openSelect('Studenti');
FOR s
IN (   SELECT nome, matricola
      FROM studenti)
LOOP
  GUI.addOption(s.nome,s.matricola);
END LOOP;
GUI.closeSelect;
```

## Esempio di cursore implicito

```
GUI.openTable;
GUI.openRow;
  GUI.addTitleCell('NOME');
  GUI.addTitleCell('COGNOME');
GUI.closeRow;
FOR s
IN (SELECT nome, matricola FROM studenti)
LOOP
  GUI.openRow;
    GUI.addTextCell(s.nome);
    GUI.addTextCell(s.matricola);
  GUI.closeRow;
END LOOP;
GUI.closeTable;
```

## SEQUENCE

- La generazione di chiavi numeriche si puo' fare come segue:

```
select max(codice)+1 into nuovocodice
from persone;
insert into persone
values (nuovocodice, nome, cognome)
```
- Questa tecnica si presta a deadlock; ORACLE mette a disposizione contatori persistenti, detti sequence:

```
create sequence codicePersone
increment by 1
start with 1
maxvalue 99999
cycle;
```

## SEQUENCE (CONTINUA)

- L'inserzione diventa:

```
insert into persone
values (codicePersone.nextval, nome,
cognome)
```
- In seguito `s.currval` restituisce l'ultimo valore ritornato da `s.nextval`
- Per leggere `s.currval` :

```
- select s.currval into x from dual
```

## SEQUENCE (CONTINUA)

- Per creare uno studente ed un esame

```
insert into studenti
values (seqStudenti.nextval, cognome);
```

```
insert into esami
values (seqStudenti.currval, voto);
```

- Oppure:

```
insert into studenti
values (seqStud.nextval, cognome);
```

```
select seqStud.currval into ultimaMatricola
from dual;
```

```
insert into esami
values (ultimaMatricola, voto);
```

## BINDING DI PL/SQL

- PL/SQL è compilato, per cui:
  - i nomi di tabelle e colonne devono essere specificati come costanti
  - può riferire solo tabelle e colonne già specificate
  - non può eseguire comandi del DDL
- Se lo schema è cambiato al momento di eseguire una funzione, il sistema riefettua il binding, che però può fallire se il nuovo schema è incompatibile con la procedura
- Esiste un package (DBMS\_SQL) per effettuare generazione e compilazione dinamica di PL/SQL

## SQL in PL/SQL

- Solo il DML ed il controllo delle transazioni
- Tutte le funzioni SQL, ma le funzioni aggregate solo nelle query
- Pseudocolonne nelle query:
  - **CURRVAL**, **NEXTVAL**: usano una SEQUENCE, dentro una select o dentro una insert / sei
  - **ROWID**: identifica una ennupla
  - **ROWNUM**: una query ne assegna uno diverso (crescente e consecutivo) ad ogni ennupla trovata
- Nella clausola where:
  - confronti, con eventualmente **some(any)** ed **all**
  - **between**, **exists**, **in**, **is null**
- Tra due **select**: **intersect**, **minus**, **union**, **union all**

## CURSORI VARIABILI

- Sono cursori su cui si possono fare assegnamenti, o puntatori assegnabili ad aree di lavoro.
- Utili per:
  - fare aprire un cursore da un subroutine
  - comunicazione tra ambiente esterno e PL/SQL
  - avere un cursore che può essere legato a tabelle, query o anche tipi diversi

## DEFINIZIONE DI CURSORI VARIABILI

- Prima si dichiara il tipo poi la variabile

```
declare
  type curtipo is ref cursor
                        return prenota%rowtype;
  curVar curtipo ;
```

- La parte `return` è opzionale
- Le variabili di cursore non possono essere variabili persistenti (variabili di package, colonne nel db)
- Anche di una variabile di cursore si può estrarre il `%rowtype`
- Operazioni:
  - `open cur for query;`
  - attributi, `fetch into, close`

## TIPI COLLEZIONE

- Array associativi (Tabella Index-By): hash table, chiave int o string, tipo collezione “storico”
- Nested table e Varray: meno flessibili, ma possono essere memorizzate in una casella del DB
- Tabelle annidate: simili alle index-by, ma:
  - Alcune procedure in più (trim,extend)
  - Una nested table vuota è uguale a NULL
  - Una nested table va creata ed estesa in modo esplicito
  - Set-semantics: quando è memorizzata nel DB perde l'ordine e la posizione dei buchi
- Varray: simili alle tabelle annidate, ma:
  - Hanno un maximum size
  - Non hanno buchi, ma solo un upper bound ( $\leq$  maximum size)
  - Conservano ordine e subscript nel DB

## TIPI TABELLA INDEX-BY

- Tabelle hash in memoria centrale:
  - `TYPE mioTipoTabella IS TABLE`
  - `OF tipoElem [NOT NULL]`
  - `INDEX BY [BINARY_INTEGER| VARCHAR2(size)]`
  - `miaTabella mioTipoTabella;`
- `tipoElem`: un tipo qualunque (anche T I-B), dichiarato altrove
- Una tabella può essere un parametro o il risultato di una funzione
- Accesso alle righe: `miaTabella (expr)`; tabelle di uguale tipo si possono assegnare per intero

## ATTRIBUTI DI UNA TABELLA I-B

- `EXISTS(i)`: bool
- `PRIOR(i), NEXT(i), FIRST, LAST, COUNT`: binary\_integer
- Esempio:

```
DECLARE
  i BINARY_INTEGER
BEGIN
  i := tab.FIRST;
  WHILE i IS NOT NULL
  LOOP ..; i := tab.NEXT(i);
  END LOOP;
```
- `tabella.DELETE, tabella.DELETE(i), tabella.DELETE(i,j)`

## INSERIMENTI IN UNA TABELLA I-B

- Assegnamento:

```
TYPE TipoTabVarChar IS TABLE
  OF VARCHAR2 INDEX BY BINARY_INTEGER;

tabNomi TipoTabVarChar;

tabNomi(4) := 'abc';
```

- Select - into:

```
TYPE TipoTabPers IS TABLE
  OF Persone%RecType INDEX BY BINARY_INTEGER;

tabPersone TipoTabPers;

select * into tabPersone(x)
from studenti where matricola = x;
```

## COPIARE UNA RELAZIONE

- Con un loop su di una query:

```
for s in
  (select nome, cognome, matricola from studenti)
loop
  tn(s.matricola) = s.nome;
  tc(s.matricola) = s.cognome;
  tncm(s.matricola) = s
end loop
- tn(456456) =>'Mario', tc(456459) =>'Rossi'
- tn(456459) =>'Luigi', tc(456459) =>'Bianchi'
```

- La clausola bulk collect into:

```
select nome, cognome, matricola bulk collect into tncm
from studenti s
- tncm(1) => (456456,'Mario','Rossi')
- tncm(2) => (456459,'Luigi','Bianchi')
```

## COPIARE UNA RELAZIONE

- Simulare la bulk collect con un loop:

```
DECLARE
  TYPE MioTipoTabella IS TABLE OF emp%ROWTYPE
    INDEX BY ...
  miaTab MioTipoTabella;
  i BINARY_INTEGER := 0;
  CURSOR c IS SELECT * FROM emp;
BEGIN
  OPEN c;
  LOOP
    i:=i+1;
    FETCH c INTO miaTab(i);
    EXIT WHEN c%NOTFOUND;
  END LOOP
```

## CICLI SULLE TABELLE

- Riempiamo due tabelle con un cursore implicito:

```
DECLARE
  TYPE TNomeTab IS TABLE OF emp.nome%TYPE...
  TYPE TSalTab IS TABLE OF emp.sal%TYPE INDEX...
  miaNomeTab TNomeTab ;
  miaSalTab TSalTab ;
  i BINARY_INTEGER := 0;
BEGIN
  FOR imp IN (SELECT nome, sal FROM emp)
  LOOP
    i:=i+1;
    miaNomeTab(i) := imp.nome;
    miaSalTab(i) := imp.sal;
  END LOOP
END
```

## TABELLE PASSATE COME PARAMETRO

- Un parametro tabella non può avere default null, ma:

```
CREATE OR REPLACE PACKAGE pp AS
  TYPE MyTableT IS TABLE OF varchar(80)
    INDEX BY binary_integer;
  myEmptyTable MyTableT;
  PROCEDURE test(
    t MyTableT DEFAULT myEmptyTable
  );
END pp;
```

## DICHIARAZIONE DI NT e VA

- `TYPE CourseList IS TABLE OF VARCHAR2(10);`
- `TYPE Project IS`  
    `OBJECT(`  
        `project_no NUMBER(2),`  
        `title VARCHAR2(35),`  
        `cost NUMBER(7,2));`
- `TYPE ProjectList IS VARRAY(50) OF Project;`

## INIZIALIZZAZIONE DI NT e VA

- Una Nested table o Varray vale `null` fino a che:

```
DECLARE my_courses CourseList;
BEGIN my_courses := CourseList('Econ 2010', 'Acct
  3401', 'Mgmt 3100', 'PoSc 3141', 'Mktg 3312',
  'Engl 2005');
```

- Per modificare la dimensione, usare il metodo `extend`
  - `my_courses.extend` (ovvero, `extend(1)`)
  - `my_courses.extend(3)`: aggiunge tre elementi nulli
  - `my_courses.extend(3,1)`: aggiunge tre elementi copiati dal primo
- `Trim` annulla l'effetto di `extend`

## BULK BINDS

- Lo statement sotto (var è un intero)
  - `forall var in e1..e2 sqlstatement`
  - `forall var in indices of collection sqlstatement`
  - `forall var in values of ind-coll sqlstatement`viene eseguito in modo molto più efficiente del loop:
  - `for var in e1..e2 sqlstatement`
- Dentro `sqlstatement` posso usare `var` solo in `mytable(var)` (non `mytable(expr(var)!)`)

## BULK SELECT INTO

```
TYPE MyTable IS TABLE OF char(15)
                index by binary_integer;
i binary_integer;
t myTable;
s myTable;
begin
    SELECT nome, cognome BULK COLLECT INTO t, s
    FROM persone WHERE ROWNUM <= 100;
    for i in t.first..t.last
    loop
        ...t(i)...;
        ...s(i)...;
    end loop;
end;
```