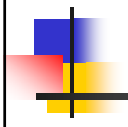
PL/SQL



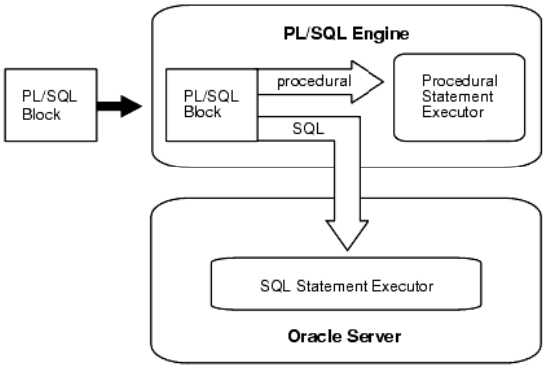
Cos'è PL/SQL?

- Il linguaggio procedurale per l'estensione del linguaggio SQL di proprietà di Oracle
- Lo standard SQL è esteso dai principali sistemi commerciali:
  - Da Oracle con PL/SQL (e Java)
  - Da Access con Visual Basic
  - Da SQL Server con Transact-SQL

2

Il motore di PL/SQL I

- Esegue le porzioni procedurali del codice ma invia al server oracle i comandi SQL



3

Il motore di PL/SQL II

- Può essere posizionato sia sul lato client, sia sul lato server
  - Molti tool Oracle (es: Oracle Forms, Oracle Reports) dispongono di un proprio motore PL/SQL
- Server-side
  - I blocchi PL/SQL sono processati dal motore PL/SQL che fa parte del Server Oracle
- Client-side
  - Il motore PL/SQL filtra i comandi SQL e li invia SQL al server Oracle mentre esegue direttamente i comandi procedurali.
- ATTENZIONE: SQL*Plus non è dotato di un proprio motore PL/SQL

4

Blocchi PL/SQL

- I blocchi PL/SQL (Block)
  - Rappresentano l'unità elementare di codice PL/SQL
  - Normalmente contengono i comandi sufficienti a eseguire uno specifico compito
- Esistono due tipi di blocchi PL/SQL
  - anonymous
  - Named: Si tratta di blocchi PL/SQL precompilati che vengono memorizzati nel database
    - stored procedure
    - function
    - trigger
    - package

5

Struttura di un blocco PL/SQL

- Sezione di dichiarazione
  - Per dichiarare, variabili, costanti, cursori,ecc.
  - E' opzionale
- Sezione di esecuzione
  - Descrive la logica dei comandi
  - Può contenere istruzioni SQL
  - E' obbligatoria
- Sezione di gestione delle eccezioni
  - Viene eseguita quando si presentano degli errori
  - E' opzionale

```
DECLARE
  • • •
BEGIN
  • • •
EXCEPTION
  • • •
END;
```

Attenzione nella definizione delle procedure e funzioni la clausola DECLARE è implicita

6

Procedure I

- Una procedura è un blocco di codice PL/SQL dotato di un nome che viene mantenuto all'interno del database (stored procedure)

```
CREATE PROCEDURE nome_procedura [(parametri)] IS
       Definizioni;
BEGIN
       Corpo procedura;
END;
```

- La clausola IS sostituisce la clausola DECLARE

7

Procedure II

- Una procedura può essere richiamata utilizzando il comando call

```
CALL nome_procedura([parametri]);
```

- Parametri è una sequenza di

```
Nome_variabile TIPO_DATO
```

che specifica eventuali valori passati in input

- TIPO_DATO non deve specificare lunghezza, precisione o scala.

  VARCHAR2(10) non è un tipo di dato valido VARCHAR2 si!
- Oracle deriva lunghezza, precisione o scala degli argomenti dall'ambiente da cui la procedura è chiamata.

8

Funzioni

- Le funzioni sono del tutto simili a procedure a meno della clausola RETURN che specifica il tipo di tato restituito

```
CREATE FUNCTION nome_funzione … RETURN BOOLEAN IS
       Definizioni;
BEGIN
       Corpo procedura;
       RETURN Variabile;
END;
```

9

Esempio: anonymous

```
DECLARE
   qty_on_hand NUMBER(5);
BEGIN
   SELECT quantity INTO qty_on_hand
      FROM inventory
      WHERE product = 'TENNIS RACKET' FOR UPDATE OF quantity;

   IF qty_on_hand > 0 THEN -- check quantity
      UPDATE inventory SET quantity = quantity – 1
         WHERE product = 'TENNIS RACKET';
      INSERT INTO purchase_record
         VALUES ('Tennis racket purchased', SYSDATE);
   ELSE
      INSERT INTO purchase_record
        VALUES ('Out of tennis rackets', SYSDATE);
   END IF;
   COMMIT;
END;
```

10

Esempio: named

```
CREATE OR REPLACE PROCEDURE Esempio IS
   qty_on_hand NUMBER(5);
BEGIN
   SELECT quantity INTO qty_on_hand
      FROM inventory
      WHERE product = 'TENNIS RACKET' FOR UPDATE OF quantity;

   IF qty_on_hand > 0 THEN -- check quantity
      UPDATE inventory SET quantity = quantity - 1
         WHERE product = 'TENNIS RACKET';
      INSERT INTO purchase_record
         VALUES ('Tennis racket purchased', SYSDATE);
   ELSE
      INSERT INTO purchase_record
        VALUES ('Out of tennis rackets', SYSDATE);
   END IF;
   COMMIT;
END;
```

11

Esempio2: anonymous

```
--WHOIS.sql
ACCEPT sv_writerid CHAR PROMPT 'Which Writerid? '

DECLARE
   v_writerid CHAR(4);
   v_writername VARCHAR2(50);
BEGIN
   v_writerid := '&sv_writerid';
   SELECT fn || ' ' || ln INTO v_writername
      FROM writer
      WHERE writerid = v_writerid;
   DBMS_OUTPUT.PUT_LINE('Writer ' || v_writerid || ' is ' || v_writername);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('No such writer: '|| v_writerid);
END;
```

12

Sostituzione di variabili

- ACCEPT
  - Crea una variabile di sostituzione e permette all'utente di inserirne il valore
  - SQL*Plus sostituisce il valore della variabile prima di inviare il blocco al motore PLSQL
- SET VERIFY [ON|OFF]
  - Permette di attivare|disattivare i messaggi di sostituzione

13

Esempio2: named

```
CREATE OR REPLACE PROCEDURE WHOIS(v writerid VARCHAR2) IS
   v_writername VARCHAR2(50);
BEGIN
   SELECT fn || ' ' || ln INTO v_writername
      FROM writer
      WHERE writerid = v_writerid;
   DBMS_OUTPUT.PUT_LINE('Writer ' || v_writerid || ' is ' || v_writername);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('No such writer: '|| v_writerid);
END;
```

14

DBMS_OUTPUT.PUT_LINE

- Mostra l'output a video
- La procedura scrive l''output su un buffer dell'SGA da cui può essere letto mediante il comando `.get_line`
- In SQL Plus l'output può essere visualizzato ponendo `SET SERVEROUTPUT ON`
- DBMS_OUTPUT è un *package*
- .PUT_LINE è una *procedure* all'interno del package

15

Come viene eseguito PL/SQL ?

- Dato che SQL*Plus non è dotato di un motore PL/SQL deve inviare un blocco anonimo a Oracle
- Un blocco deve essere compilato prima che possa essere eseguito
  - Controllo sintattico
    - Struttura del comando, parole riservate e variabili
  - Binding
    - Controlla che gli oggetti referenziati esistano
  - Generazione del p-code
    - Istruzioni che il motore PL/SQL può eseguire

16

Dichiarazione di una variabile PL/SQL

```
identifier [CONSTANT] datatype [NOT NULL]
     [:= | DEFAULT expr];
```

- La dichiarazione deve essere effettuata nella sezione DECLARE
- Per deafualt le variabili sono inizializzate a NULL
- Le variabili sono dichiarate e inizializzate ogni volta che si accede al blocco
- Due variabili con lo stesso nome devono essere dichiarate in blocchi diversi
- Consigli:
  - Naming Conventions
    - Fino a 30 caratteri, non case sensitive, cominciano con una lettera e non possono contenere spazi
    - Non definire una variabile con il nome della colonna se queste vengono usate contemporaneamente
  - Dichiarate una variabile per riga

17

Assegnamento di un valore a una variabile

- Viene effettuato tramite comando di assegnamento nella sezione di esecuzione ...

```
variablename := expression;
```

```
CREATE PROCEDURE Esempio IS
   c_tax_rate  CONSTANT NUMBER(3,2) := 8.25;
...
BEGIN
...
  v_hiredate := '31-DEC-98';
  v_fullname := ln || ', ' || fn;
...
```

18

Assegnamento di un valore a una variabile

- ... oppure tramite il comando SELECT INTO

```
CREATE PROCEDURE Esempio IS
     v_max_len   number(7);
BEGIN
    SELECT max(length) INTO v_max_len
      FROM article;
...
```

19

4 tipi di variabili

- **Scalar**
  - Possono contenere un singolo valore
  - Corrispondono ai tipi di dati previsti per le tabelle Oracle più poche altre (es: Boolean)
- **Composite**
  - Permettono di manipolare gruppi di campi
  - es: una variabile di tipo %ROWTYPE memorizza un'intera riga
- **Reference**
  - Contengono puntatori
- **LOB** (Large OBjects)
  - Contengono elementi, chiamati *locators,* che specificano la posizione di oggetti di grosse dimensioni (es. immagini) che sono memorizzati separatamente

20

I principali tipi di dati scalari I

- **VARCHAR2 (lung. max.)**
  - Fino a 32,767 byte
- **CHAR [(lung. max.)]**
  - Fino a 32,767 byte
- **NUMBER [(precisione, scala)]**
  - precisione: 0-38
  - scala: -84 to 127
  - NUMBER(5,2) -> ddd.dd
- **DATE**
  - Da: January 1, 4712 BC A: December 31, 9999 AD
- **BOOLEAN**
  - TRUE o FALSE o NULL
  - Non ha nessun tipo corrispondente nei tipi degli attributi

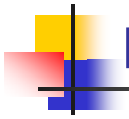
21

I principali tipi di dati scalari III

- Esempi

```
CREATE PROCEDURE Esempio IS
    v_job         VARCHAR2(9);
    v_total_sal   NUMBER(9,2) := 0;
    v_duedate     DATE := SYSDATE + 7;
    v_valid       BOOLEAN NOT NULL := TRUE;
    c_tax_rate    CONSTANT NUMBER(3,2) := 8.25;
BEGIN
……..
```

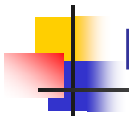
22



Le variabili %TYPE I

- Quando si definisce una variabile PL/SQL per memorizzare il valore di una colonna è necessario assicurarsi la corrispondenza tra i due tipi di dato
  - In caso contrario si verificherà un errore PL/SQL durante l'esecuzione
- Un tipo di dato "anchored" evita questo problema
  - Se cambia la definizione della colonna, cambia anche runtime la definizione della variabile. Si realizza così l'indipendenza dei dati e si permette ai programmi di adattarsi ai cambiamenti del database
- %TYPE dichiara una variabile in base a:
  - La definizione di una colonna del database
  - Un'altra variabile definita precedentemente
- Possibili prefissi per %TYPE sono:
  - I nomi della tabella e della colonna
  - Il nome della variabile precedentemente definita

23



Le variabili %TYPE II

```
CREATE PROCEDURE Esempio IS
  v_writerid    writer.writerid%TYPE;
  v_length      article.length%TYPE;
  v_min_length  v_length%TYPE := 0;
BEGIN
……
```

24

Lifetime

- Indica l'intervallo durante il quale una variabile esiste in memoria e può contenere un valore
- Lo spazio in memoria è **allocato** quando la variabile viene dichiarata
- Lo spazio in memoria è **deallocato** quando quando il programma raggiunge il comando END del blocco in cui è stata creata

25

Scope (Visibilità)

- La regione del programma in cui referenziare una variabile
- Le variabili dichiarate in una blocco PL/SQL sono locali al blocco e sono considerate globali per tutti i sotto blocchi
- La visibilità è inibita se nel blocco viene dichiarata una variabile con lo stesso nome.
  - Un blocco può fare riferimento a variabili dichiarate nei blocchi padre
  - Un blocco NON può fare riferimento a variabili dichiarate nei blocchi figli

26

Esempio

```
DECLARE
  v_sal          NUMBER(7,2) := 60000;
  v_comm         NUMBER(7,2) := v_sal * .20;
  v_message      VARCHAR2(255) := ' eligible for commission';

BEGIN
  DECLARE
    v_sal         NUMBER(7,2) := 50000;
    v_comm        NUMBER(7,2) := 0;
    v_total_comp  NUMBER(7,2) := v_sal + v_comm;
  BEGIN
    v_message := 'CLERK not'|| v_message;
  END;

  v_message := 'SALESMAN'|| v_message;

END;
```

27

SELECT INTO

```
SELECT  select_list
INTO    {variable_name[, variable_name]...
        | record_name}
FROM    table
WHERE   condition;
```

- E' necessario indicare ordinatamente il nome di una variabile per ogni colonna selezionata.
- L'interrogazione deve restituire una e una sola tupla
  - In caso contrario si genererà un errore
  - PL/SQL gestisce questi due errori generando due exception predefinite, che possono quindi essere trattate nella sezione EXCEPTION
    - NO_DATA_FOUND
    - TOO_MANY_ROWS

28

Esempio SELECT INTO

```
CREATE PROCEDURE Esempio(v_writerid writer.writerid%TYPE) IS
    v_lastname  varchar2(90);
    v_amount    number(9,2);
BEGIN
    SELECT ln, amount
    INTO   v_lastname, v_amount
    FROM   writer
    WHERE  writerid = v_writerid;
    DBMS_OUTPUT.PUT_LINE('Writer: ' || v_lastname || ' earns
$' || v_amount);
END;
```

29

Comandi SQL in PL/SQL

Non sono permessi comandi di tipo:

- DDL non è permesso
  - CREATE TABLE, CREATE INDEX, ALTER TABLE, DROP VIEW
- DCL non è permesso
  - GRANT, REVOKE, CREATE USER, DROP ROLE, ALTER USER

Sono permessi:

- DML è permesso
  - INSERT, UPDATE, DELETE
- TCL è permesso
  - COMMIT, ROLLBACK, SAVEPOINTC

30

Esempio I

```
…
BEGIN
  INSERT INTO  emp(empno, ename, job, deptno)
  VALUES      (empno_sequence.NEXTVAL,
                 'HARDING', 'CLERK', 10);
  COMMIT;
…
END;
```

```
DECLARE
  v_sal_increase   emp.sal%TYPE := 2000;
BEGIN
  UPDATE  emp
  SET      sal = sal + v_sal_increase
  WHERE   job = 'ANALYST';
  COMMIT;
END;
```

31

Esempio II

```
CREATE OR REPLACE PROCEDURE Esempio IS
  v_deptno   emp.deptno%TYPE := 10;

BEGIN
  DELETE FROM   emp
  WHERE         deptno = v_deptno;

  COMMIT;
END;
```

32

DEBUGGING

- L'ambiente di PL-SQL non fornisce strumenti di debugging evoluti.
- Il sistema indica se una procedura o funzione è stata creata correttamente o con errori
- Tramite il comando SHOW ERRORS è possibile avere una lista degli errori presenti nella procedura

33

Esercizi in aula

- EX1: Scrivere una procedura che scriva in output la stringa ESAME DI SISTEMI INFORMATIVI come concatenazione di 4 variabili
- EX2: Scrivere una procedura/funzione che, dato il codice fiscale di un fornitore restituisca il suo nome

34

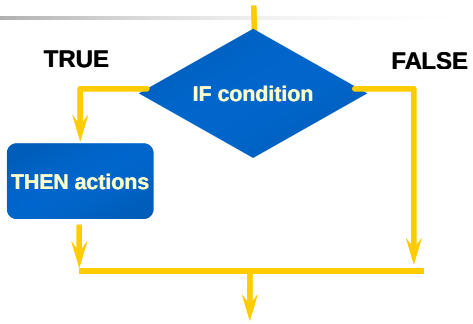
Controllo del flusso di elaborazione

- Per cambiare il flusso di esecuzione all'interno di un blocco di codice sono disponibili i seguenti comandi
  - IF-THEN
    - Seleziona se eseguire o non un comando
  - IF-THEN-ELSE
    - Seleziona quale di due comandi debbano essere eseguiti in mutua esclusione
  - IF-THEN-ELSIF
    - Seleziona quale di più comandi debbano essere eseguiti in mutua esclusione
- Attenzione:
  - ELSIF è una parola
  - END IF sono due parole

35

IF-THEN

- Se la condizione è TRUE, allora viene eseguito il ramo THEN
- Se la condizione è FALSE o **NULL**, allora il ramo THEN non viene eseguito
- In entrambi i casi il flusso viene ripreso al primo comando che segue END IF




```
IF condition THEN
  statement(s);
END IF;
```

36

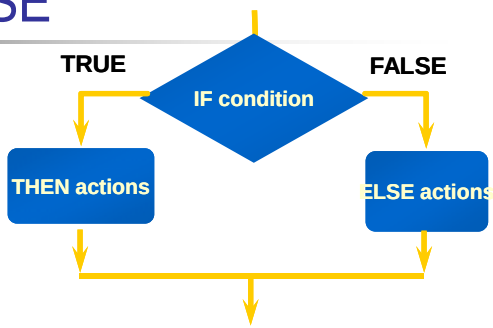
Esempio

```
...
IF sales > quota THEN
   bonus:=compute_bonus(empid);
   UPDATE payroll
     SET pay = pay + bonus
     WHERE empno = emp_id;
END IF;
...
```

37

IF-THEN-ELSE

- Se la condizione è TRUE, allora viene eseguito il ramo THEN
- Se la condizione è FALSE o **NULL**, allora viene eseguito il ramo ELSE
- In entrambi i casi il flusso viene ripreso al primo comando che segue END IF




```
IF condition THEN
  statement(s);
ELSE
  statement(s);
END IF;
```

38

Esempio

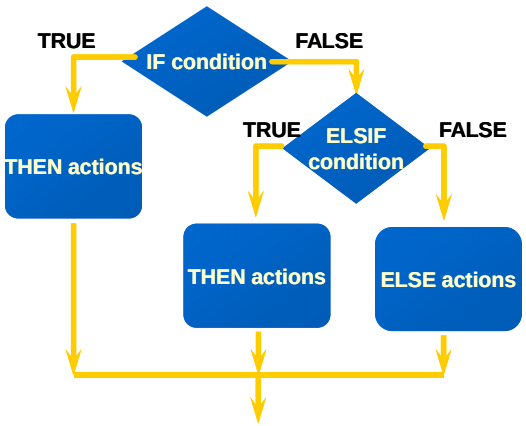
```
CREATE OR REPLACE PROCEDURE Esempio IS
   qty_on_hand NUMBER(5);
BEGIN
   SELECT quantity INTO qty_on_hand
      FROM inventory
      WHERE product = 'TENNIS RACKET' FOR UPDATE OF
   quantity;

   IF qty_on_hand > 0 THEN -- check quantity
      UPDATE inventory SET quantity = quantity – 1
         WHERE product = 'TENNIS RACKET';
      INSERT INTO purchase_record
         VALUES ('Tennis racket purchased', SYSDATE);
   ELSE
      INSERT INTO purchase_record
        VALUES ('Out of tennis rackets', SYSDATE);
   END IF;
   COMMIT;
END;
```

39

IF-THEN-ELSIF

- Simile a un comando SWITCH
- Quando è possibile si usi ELSIF invece che un insieme di IF in cascata
  - Il codice risultante sarà più leggibile
- Le condizioni devono essere mutualmente esclusive
- Dopo l'esecuzione il flusso viene ripreso al primo comando che segue END IF



40

IF-THEN-ELSIF Esempio

```
CREATE OR REPLACE PROCEDURE Esempio IS
   v_title         article.title%TYPE;
   v_length        article.length%TYPE;
   v_descr         VARCHAR2(6);
BEGIN
   SELECT title, length INTO v_title, v_length
      FROM article
      WHERE articlenum = &sv_articlenum;
   IF v_length <=1500 THEN
      v_descr := 'Brief';
   ELSIF v_length BETWEEN 1501 and 2500 THEN
      v_descr := 'Short';
   ELSIF v_length BETWEEN 2501 and 4000 THEN
      v_descr := 'Medium';
   ELSE
      v_descr := 'Long';
   END IF;
   DBMS_OUTPUT.PUT_LINE('Article ' || v_title || ' is ' || v_descr);
END;
.
/
```

41

Condizioni complesse

- I valori null sono gestiti tramite l'operatore IS NULL
  - es: `IF v_gender IS NULL THEN`
- Qualsiasi espressione aritmetica che comprenda un NULL comporta il risultato NULL
- Nella concatenazione di più variabili la presenza di un NULL viene trattata come una stringa vuota
- Condizioni complesse vengono create utilizzando gli **operatori logici** NOT, AND, and OR
  - es: `IF v_length > 500 AND v_type = 'BUS' THEN ...`
  - es: `IF v_length > 500 OR  v_type = 'BUS' THEN ...`
  - es:
    ```
    IF v_length > 500 OR  v_type = 'BUS'  AND
        v_type = 'LAW' THEN ...
    ```
  - La precedenza è fissata come: NOT, AND, OR

42

Logica a tre valori

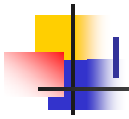
AND	*TRUE*	*FALSE*	*NULL*
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	*TRUE*	*FALSE*	*NULL*
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

Esercizi in aula

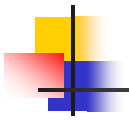
- EX3: Scrivere una procedura che calcoli il valore totale degli ordini e indichi in output se tale valore è superiore o inferiore a 1000
- EX4: modificare il codice di EX3 in modo da ottenere il seguente output:
  1. Se totale < 100 ‘Risultato scarso’
  2. Se 100< totale < 1000 ‘Risultato in media’
  3. Se 1000 < totale ‘Risultato buono’



Il concetto di eccezione I

- Cosa è una exception?
  - Un identificatore PL/SQL che viene valorizzato durante l'esecuzione di un blocco
  - L'esecuzione viene trasferita al corrispondente gestore dell'eccezione nella sezione exception del blocco
- Come avviene la valorizzazione?
  - Automaticamente (implicitamente) quando si verifica un errore runtime
  - Esplicitamente se nel codice è presente l'istruzione RAISE
- Come vengono gestite?
  - Includendo una routine corrispondente nella sezione exception
- Cosa avviene in caso contrario?
  - Il blocco PL/SQL termina con un errore
  - L'eccezione è propagata all'applicazione chiamante
  - SQL*Plus mostra il corrispondente messaggio di errore

45



Il concetto di eccezione II

- Possono essere definiti molti tipi di eccezioni ognuno associato a un proprio insieme di comandi
  - Ogni gestore è identificato da una clausola WHEN, che specifica una o più eccezioni, seguita da un insieme di comandi
- Si può verificare una sola eccezione per volta
- Il gestore **OTHERS**
  - Controlla ogni eccezione non trattata esplicitamente
  - Deve essere l'ultima eccezione nella lista

46

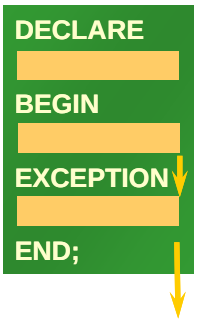
Gestione delle eccezioni

Individua l'eccezione

L'eccezione si manifesta

L'eccezione viene gestita

Propaga l'eccezione



L'eccezione si manifesta

L'eccezione non viene gestita

L'eccezione è propagata all'ambiente chiamante

47

Gestione delle eccezioni

```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    statement1;
    statement2;
    . . .
  [WHEN exception3 [OR exception4 . . .] THEN
    statement1;
    statement2;
    . . .]
  [WHEN OTHERS THEN
    statement1;
    statement2;
    . . .]
```

48

Eccezioni predefinite I

- PL/SQL predefinisce alcune eccezioni comuni
- Esempi:
  - NO_DATA_FOUND (ORA-01403)
    - Una SELECT INTO ha restituito 0 righe
  - TOO_MANY_ROWS (ORA-01422)
    - Una SELECT INTO ha restituito più di una riga
  - VALUE_ERROR (ORA-06502)
    - Si è verificato un errore aritmetico, numerico, di conversione o su un vincolo
    - es: si è tentato di assegnare il valore NULL a una variabile definita NOT NULL, oppure si è tentato di assegnare 9876 a una variabile definita NUMBER(2)
  - ZERO_DIVIDE (ORA-01476)
  - DUP_VAL_ON_INDEX (ORA-00001)

49

Eccezioni predefinite II

```
BEGIN
...
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    statement1;
    statement2;
  WHEN TOO_MANY_ROWS THEN
    statement1;
    statement2;
  WHEN OTHERS THEN
    statement1;
    statement2;
END;
```

50

NO_DATA_FOUND

```
CREATE OR REPLACE PROCEDURE Esempio(v_writerid varchar2) IS
   v_writername VARCHAR2(50);
BEGIN
   SELECT fn || ' ' || ln INTO v_writername
       FROM writer
       WHERE writerid = v_writerid;

   DBMS_OUTPUT.PUT_LINE('Writer ' || v_writerid || ' is '
   || v_writername);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
       DBMS_OUTPUT.PUT_LINE('No such writer: '||
   v_writerid);
END;
```

51

NO_DATA_FOUND

- Le funzioni di aggregazione SQL (es. AVG, SUM) restituiscono sempre un valore o NULL
- Un comando SELECT INTO che nella select list include solo funzioni di aggregazione non attiva mai l'eccezione NO_DATA_FOUND.
- Ovviamente ciò non è vero se il comando SELECT INTO prevede anche un raggruppamento

52

Esercizi in aula

- EX5: Scrivere una procedura che calcoli il valore totale degli ordini effettuati in una certa data. Nel caso in cui non sia presente nessun ordine viene visualizzato il messaggio “Nessun ordine presente per la data: ---------”
- EX6: Scrivere una procedura che permetta di visualizzare il nome del cliente relativo a una data fattura. Nel caso la fattura non sia presente visualizzare tramite il comando RAISE “La fattura --- non è stata registrata”

Cicli

- PL/SQL mette a disposizioni 4 istruzioni per il controllo dei cicli:
  - Cicli semplici
  - Ciclo WHILE
  - Cicli FOR numerici
  - Cicli FOR per cursori

Cicli semplici (Pre-Test)

- La condizione di uscita viene controllata prima dell'esecuzione di uno qualsiasi dei comandi
  - Se la condizione è verificata fin dall'inizio nessun comando verrà eseguito

```
CREATE OR REPLACE PROCEDURE Esempio(p_end_at NUMBER) IS
    v_counter    NUMBER(2) := 1;
BEGIN
  LOOP
    EXIT WHEN v_counter > p_end_at;
    DBMS_OUTPUT.PUT_LINE(v_counter);
    v_counter := v_counter + 1;
END LOOP;
END;
```

55

Cicli semplici (Post-Test)

- La condizione di uscita viene controllata dopo la prima esecuzione dei comandi
  - I comandi vengono eseguiti almeno una volta

```
CREATE OR REPLACE PROCEDURE Esempio(p_end_at NUMBER) IS
v_counter NUMBER(2) :=1;
BEGIN
   LOOP
     DBMS_OUTPUT.PUT_LINE(v_counter);
     v_counter := v_counter + 1;
     EXIT WHEN v_counter > p_end_at;
   END LOOP;
END;
```

56

Cicli WHILE

- Ripete una sequenza di comandi finché la condizione è TRUE
  - La condizione viene verificata prima di eseguire i comandi
    - Ciclo pre-test
  - Il ciclo termina quando la condizione diviene FALSE o NULL
- Può essere utilizzato il comando EXIT per terminare in maniera anticipata il ciclo

```
WHILE condition
LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

```
WHILE condition1
LOOP
  statement1;
  statement2;
  EXIT WHEN condition2
  . . .
END LOOP;
```

57

Esempio

```
CREATE PROCEDURE Esempio(p_end_at NUMBER) IS
v_counter     NUMBER(2) := 1;
BEGIN
   WHILE v_counter <= p_end_at
   LOOP
      DBMS_OUTPUT.PUT_LINE(v_counter);
      v_counter := v_counter + 1;
   END LOOP;
END;
```

58

Cicli FOR numerici

- Un contatore implicito viene incrementato automaticamente ad ogni ciclo
  - L'incremento è sempre di 1
- Il ciclo continua finché il contatore è ≤ all'upper bound.
- Se il lower bound ha un valore superiore all'upper bound i comandi non vengono eseguiti

```
FOR counter IN [REVERSE] lower..upper
LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

- *lower* e *upper bound* possono essere numeri, variabili, o espressioni che possano essere sempre valutati come interi
- Il *counter* è definito e può essere referenziato solo all'interno del ciclo
- Può essere utilizzato il comando EXIT per terminare in maniera anticipata il ciclo

59

Esempio

```
CREATE PROCEDURE Esempio(p_end_at NUMBER) IS
BEGIN
  FOR v_count IN 1..p_end_at
  LOOP
     DBMS_OUTPUT.PUT_LINE(v_count);
  END LOOP;
END;
```

60

Cicli Nested e Label

- I cicli FOR, WHILE, e simple possono essere innestati
- E' possibile associare a un loop una label per semplificare la lettura del codice. La label potrà essere infatti inclusa dopo il comando END LOOP
- Per dichiarare una label vengono utilizzati i delimitatori (<<*label*>>)

61

Cicli Nested e Label

```
CREATE PROCEDURE Esempio IS
    v_outercounter     NUMBER(2) := 1;
    v_product          NUMBER(4);

BEGIN
   <<Outer_loop>>
   WHILE v_outercounter <= 3
   LOOP
     <<Inner_loop>>
     FOR v_innercounter IN 1.. 5
     LOOP
        v product := v outercounter* v innercounter;
        DBMS_OUTPUT.PUT_LINE(v_outercounter || ' x ' ||
                v_innercounter || ' = ' || v_product);
     END LOOP Inner_loop;

   DBMS_OUTPUT.PUT_LINE('-----------------------');
   v_outercounter := v_outercounter + 1;
   END LOOP Outer_loop;
END;
```

62

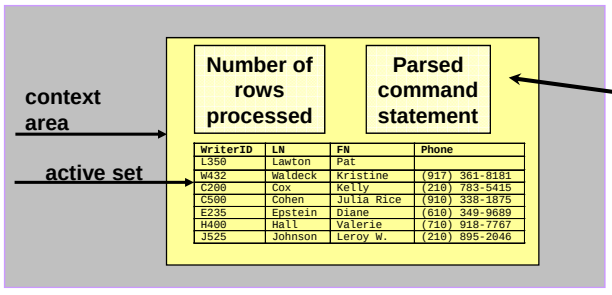
Esercizi in aula

- EX7: Calcolare il valore totale delle fatture con codice da 1 a 5 se una fattura non è presente visualizzare un messaggio di errore tramite RAISE

63

Il concetto di cursore

- Ogniqualvolta si sottoponga al sistema un comando SQL, Oracle alloca un'area di memoria in cui il comando viene analizzato ed eseguito. Tale area è detta ***context area***.
- Un ***cursore*** è un puntatore alla locazione di memoria di una context area
- Ogni comando SQL eseguito da Oracle ha associato un proprio cursore



64

Due tipi di Cursori

- **Cursori Impliciti**
  - Il server Oracle apre implicitamente un cursore durante l'esecuzione di un comando DML o di ogni query PL/SQL SELECT INTO
  - Il cursore è gestito automaticamente
    - Non si può utilizzare OPEN, FETCH, CLOSE per controllarlo
  - PL/SQL fa riferimento al più recente cursore implico come **cursore SQL**
- **Cursori Espliciti**
  - Sono dichiarati e maneggiati direttamente dal codice
  - Sono utilizzati per processare le singole righe restituite da un comando SQL multiple-row
  - Puntano alla riga corrente nell' *active set*

65

Attributi dei cursori impliciti I

- E' possibile utilizzare gli attributi del cursore sql per verificare il risultato di un comando SQL

SQL%ROWCOUNT	**Numero di righe coinvolte dal più recente comando SQL**
SQL%FOUND	**Attributo Boolean che è TRUE se l'ultimo comando SQL ha coinvolto almeno una riga**
SQL%NOTFOUND	**Attributo Booleano che è TRUE se l'ultimo comando SQL non ha coinvolto nemmeno una riga**
SQL%ISOPEN	**E' sempre FALSE poiché PL/SQL chiude i cursori impliciti immediatamente dopo l'esecuzione**

66

Attributi dei cursori impliciti II

- Dai a ogni scrittore freelance un aumento del 25% e mostra il numero di righe modificate.

```
CREATE PROCEDURE Esempio IS
-- nessun cursore è dichiarato esplicitamente
BEGIN
   UPDATE writer
      SET amount = amount * 1.25
    WHERE freelancer = 'Y';
  DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT ||' rows changed.');
 COMMIT;
END;
```

```
SQL> /
6 rows changed.

PL/SQL procedure successfully completed.
```

67

Record PL/SQL

- Un *record* PL/SQL è un gruppo di attributi correlati memorizzati in una tabella, ognuno col proprio nome e tipo
- Un record PL/SQL è quindi un tipo composto in cui i singoli campi sono trattati come un'unità logica
- Sono convenienti per gestire le righe dell' active set, poiché permettono di eseguire il FETCH di un'intera riga.
  - I valori della riga vengono caricati nei campi corrispondenti
- I campi senza un valore iniziale sono inizializzati a NULL.

68

%ROWTYPE

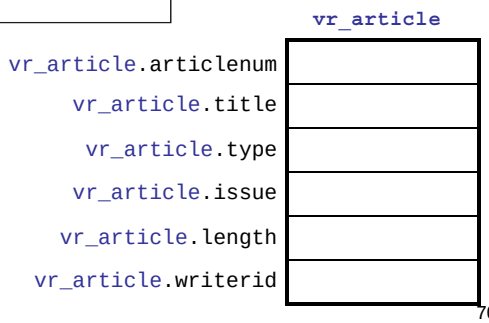
- Dichiara una variabile di tipo *record* basandosi su un insieme di campi appartenenti a una tabella, vista o cursore.
- I campi nel record assumono il nome e il tipo di quelli nella tabella, vista o cursore.
- Ci si riferisce a un membro di un campo utilizzando la sintassi
  - recordvariable_name.fieldname
- Il tipo e il numero delle colonne nel database può cambiare.
- E' necessario anteporre a %ROWTYPE il nome della tabella, vista o cursore a cui il record è associato.

69

Creazione di record utilizzando %ROWTYPE

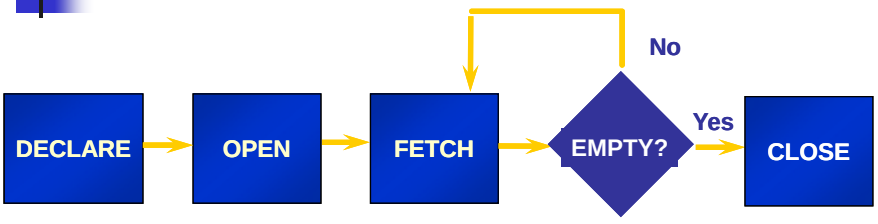
- Per dichiarare una variabile di tipo record al fine di memorizzare le informazioni presenti nella tabella ARTICLE

```
DECLARE
   vr_article  article%ROWTYPE;
   . . .
```



70

Controllo di un cursore esplicito



Crea la context area

Definisce la query che produce i dati

Esegue la query

Identifica l'active set

Posiziona il cursore *prima* della riga iniziale

Carica la riga corrente nelle variabili

Fa avanzare il cursore alla prossima riga

Controlla l'esistenza di ulteriori righe

Esegue nuovamente il FETCH se è stata trovata una riga

Rilascia l'active set

Disabilita il cursore e rilascia le risorse

71

Attributi dei Cursori espliciti

- Permettono di ottenere informazioni sui cursori espliciti

Attributo	Tipo	Descrizione
%ISOPEN	Boolean	Restituisce TRUE se il cursore è open
%NOTFOUND	Boolean	Restituisce TRUE se il FETCH più recente non ha restituito righe
%FOUND	Boolean	Restituisce TRUE se il FETCH più recente ha restituito righe.
%ROWCOUNT	Number	Restituisce il numero totale di righe restituite (ossia fetched)

72

Dichiarazione di un cursore esplicito

```
CURSOR cursor_name IS
     select_statement;
```

- *select_statement* è un qualsiasi comando SELECT
  - Può includere join, operatori di set e subquery
  - Se è necessario processare le righe in una determinata sequenza si può utilizzare la clausola ORDER BY nella query.
- E' possibile fare riferimento a variabili all'interno della query, ma queste devono essere definite anticipatamente.

73

Esempio

```
CREATE PROCEDURE Esempio IS
   CURSOR writer_cursor IS
      SELECT  ln, phone, amount
      FROM    writer;

   v_length   NUMBER(4,0) := 1500;

   CURSOR article_cursor IS
      SELECT *
      FROM   article
      WHERE  length < v_length;
. . .
```

74

Apertura di un cursore

```
OPEN  cursor_name;
```

- Esegue l'interrogazione e identifica l'active set.
- Posiziona il puntatore *prima* della prima riga nell'active set.
  - Le righe non vengono caricate nelle variabili fino all'esecuzione del comando FETCH
- Non si verifica alcuna eccezione se la query non restituisce valori.

75

Leggere i dati dal cursore

```
FETCH cursor name INTO  [variable1, variable2, ...]
                        | record_name];
```

- I dati possono essere inseriti in un record o in un insieme di variabili
- Dopo un FETCH, il cursore avanza alla prossima riga dell'active set
- Dopo ogni FETCH è necessario verificare se il cursore contiene delle righe
  - Se un cursore non acquisisce valori l'active set è stato completamente elaborato
  - Non vengono create delle eccezioni
  - Le variabili/record mantengono i valori precedenti

76

Caricamento dei dati dei cursori in variabili PL/SQL

```
CREATE PROCEDURE Esempio IS
   CURSOR writer_cursor IS
      SELECT ln, phone
      FROM   writer
      ORDER BY ln;
   v_ln     writer.ln%TYPE;
   v_phone  writer.phone%TYPE;

BEGIN
   OPEN writer_cursor;
   LOOP
      FETCH writer_cursor INTO v_ln, v_phone;
      EXIT WHEN writer_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(RPAD(v_ln,40) || v_phone);
   END LOOP;
   CLOSE writer_cursor;
END;
```

77

Caricamento dei dati dei cursori in record PL/SQL

```
CREATE PROCEDURE Esempio IS
   CURSOR writer_cursor IS
      SELECT ln, phone
      FROM   writer
      ORDER BY ln;
   vr_writer  writer_cursor%ROWTYPE;
BEGIN
   OPEN writer_cursor;
   LOOP
      FETCH writer_cursor INTO vr_writer;
      EXIT WHEN writer_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(RPAD(vr_writer.ln,40) ||
                           vr_writer.phone);
   END LOOP;
   CLOSE writer_cursor;
END;
```

78

Chiusura di un Cursore

```
CLOSE    cursor_name;
```

- Chiude il cursore dopo aver completato l'elaborazione.
- Disabilita il cursore rendendo indefinito l'active set.
- Non è possibile eseguire FETCH su un cursore chiuso.
  - Provocherebbe una eccezione di tipo INVALID_CURSOR
  - La riapertura del cursore provocherà la riesecuzione dell'interrogazione

79

Cicli FOR e cursori

```
FOR record_name IN cursor_name LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

- Semplifica l'utilizzo di cursori espliciti
  - Il cursore è aperto e ne viene recuperata una riga per ogni iterazione; il cursore è chiuso automaticamente dopo l'elaborazione dell'ultima riga.
  - Il record PL/SQL che conterrà i dati viene definito automaticamente
  - Le operazioni di OPEN, FETCH, e CLOSE avvengono automaticamente

80

ESEMPIO

- Recupera nome e cognome di ogni scrittore

```
CREATE PROCEDURE Esempio IS
   CURSOR writer_cursor IS
      SELECT ln, phone
      FROM   writer
      ORDER BY ln;
BEGIN
  FOR vr_writer IN writer_cursor LOOP  -- implicit open/fetch
     DBMS_OUTPUT.PUT_LINE(RPAD(vr_writer.ln,40) ||
                          vr_writer.phone);
  END LOOP; -- Chiusura implicita
END;
```

- Si noti la riduzione nel numero dei comandi
  - Nessuna dichiarazione per `vr_writer`
  - Esecuzione automatica di OPEN, FETCH e CLOSE

81

Cursori con Parametri

```
CURSOR cursor_name
  [(parameter_name datatype, ...)]
IS
  select_statement;
```

- I parametri permettono di passare al cursore dei valori utilizzati nella query che carica i dati durante l'apertura.
- Un cursore può essere aperto più volte nello stesso blocco producendo active set diversi
- Non possono essere utilizzati assieme a un ciclo FOR poiché i parametri devono essere specificati durante l'operazione esplicita di OPEN

82

Cursori con Parametri

```
CREATE PROCEDURE Esempio IS
  CURSOR writer_cursor (p_flstatus IN writer.freelancer%TYPE) IS
    SELECT     ln, phone
    FROM       writer
    WHERE      freelancer = p_flstatus;
  vr_writer  writer_cursor%ROWTYPE;

BEGIN
  OPEN writer_cursor('Y');
  LOOP
      FETCH writer_cursor INTO vr_writer;
      EXIT WHEN writer_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(RPAD(vr_writer.ln,40) ||
                           vr_writer.phone);
   END LOOP;
   CLOSE writer_cursor;
END;
```

83

Esercizi in aula

- EX8: Creare un cursore che restituisca separatamente l'importo di tutte le fatture
- EX9: Creare un cursore che restituisca separatamente la somma degli importi delle fatture di importo minore e maggiore di mille.

84

FOR UPDATE

```
SELECT ...
FROM   ...
FOR UPDATE [OF column_reference][NOWAIT];
```

- Applica un lock alle righe selezionate dal cursore in modo che sia possibile modificare o cancellare i valori all'interno del codice
- Il lock è applicato al momento dell'apertura del cursore non durante la fase di fetch
- Il lock è rilasciato al momento del COMMIT o ROLLBACK da eseguire al termine del ciclo
  - L'esecuzione di COMMIT o ROLLBACK per ogni riga provoca errore (ORA-01002)
- Se il cursore applica una selezione su più tabelle tramite FOR UPDATE è possibile limitare il lock a una sola tabella. Il lock è applicato solo alle righe delle tabelle di cui è citato un campo nella clausola FOR UPDATE.
- La clausola FOR UPDATE è l'ultima di ogni query di SELECT.

85

FOR UPDATE

```
SELECT ...
FROM   ...
FOR UPDATE [OF column_reference][NOWAIT];
```

- NOWAIT indica al server di non attendere se sulle tabelle è attivo un lock di un'altra sessione.
  - Si verifica una exception
  - Il controllo è restituito al programma che può eseguire altre operazioni prima di tentare di riacquisire il lock

86

FOR UPDATE Esempio

```
CREATE PROCEDURE Esempio IS
   CURSOR c_stud_zip IS
      SELECT s.student_id, z.city
        FROM student s, zipcode z
        WHERE z.city = 'Brooklyn'
        AND   s.zip = z.zip
        FOR UPDATE OF phone;
BEGIN
   FOR r_stud_zip IN c_stud_zip
   LOOP
       UPDATE student
          SET phone = '333'|| substr(phone,4)
          WHERE student_id = r_stud_zip.student_id;
   END LOOP;
   COMMIT;
END;
```

- Cosa viene bloccato?
- Quando?

- Il COMMIT è eseguito alla fine

87

WHERE CURRENT OF

```
WHERE CURRENT OF cursor ;
```

- Referenzia la riga corrente di un cursore esplicito.
- Permette di eseguire UPDATE o DELETE della riga corrente utilizzando una clausola WHERE semplificata.
  - Non richiede di creare la condizione che specifichi a quale riga applicare l'operazione poiché questa viene applicata alla riga corrente.
- E' necessario utilizzare FOR UPDATE nella definizione del cursore in modo da applicare un lock sulla tabella
  - In caso contrario si verificherà un errore

88

WHERE CURRENT OF

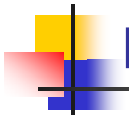
```
CREATE PROCEDURE Esempio IS
    CURSOR c_stud_zip IS
       SELECT s.student_id, z.city
         FROM student s, zipcode z
         WHERE z.city = 'Brooklyn'
         AND   s.zip = z.zip
         FOR UPDATE OF phone;
BEGIN
     FOR r_stud_zip IN c_stud_zip
     LOOP
        UPDATE student
           SET phone = '718'|| substr(phone,4)
           WHERE CURRENT OF c_stud_zip;
     END LOOP;
     COMMIT;
END;
```

89

Esercizi in aula

- EX10: Alzare del 10% il prezzo di tutti i prodotti forniti dal fornitore ‘xxx’

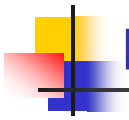
90



Procedurale vs Dichiarativo

- La modalità di calcolo da preferire è quella che massimizza le prestazioni (purchè non complichi eccessivamente il codice)
- La principale regola di massima prevede che sia demandata all'ottimizzatore la modalità di accesso ai dati
  - E' meglio far eseguire al sistema una query complessa piuttosto che molte query semplici
  - Una valutazione più approfondita richiede di conoscere le modalità di accesso e di ottimizzazione utilizzate dal DBMS… E' anche per questo motivo che le studieremo

91



Procedurale vs Dichiarativo

- Un esempio: restituire in output separatamente l'importo di tutte le fatture con codice compreso tra 1 e 5

```
CURSOR cursore_importi IS
     SELECT D_NUMF,sum(D_QTA*D_PREZZO) as IMPORTO
     FROM   dettaglio
          WHERE D_NUMF BETWEEN 1 AND 5
          GROUP BY D_NUMF;
…
   LOOP
      FETCH cursore_importi into vr_importi;
      EXIT WHEN cursore_importi%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('La fattura: ' || vr_importi.D_NUMF
               || ' e'' di importo: ' || vr_importi.IMPORTO);
  END LOOP;
```

92

Procedurale vs Dichiarativo

- Un esempio: restituire in output separatamente l'importo di tutte le fatture con codice compreso tra 1 e 5

```
…
   FOR counter IN 1.. 5 LOOP
        SELECT sum(D_QTA*D_PREZZO) as IMPORTO INTO v_importo
        FROM   dettaglio
        WHERE D_NUMF = counter
        GROUP BY D_NUMF;
       DBMS_OUTPUT.PUT_LINE('La fattura: ' || counter
              || ' e'' di importo: ' || v_importo);
   END LOOP;
```

- **Meno efficiente scandisce il database 5 volte (in assenza di indici)**

93

Procedurale vs Dichiarativo

- Un esempio: restituire in output l'importo totale delle fatture che hanno singolarmente un importo > 1000 e <= 1000

```
CURSOR cursore_importi IS
      SELECT D_NUMF,sum(D_QTA*D_PREZZO) as IMPORTO
      FROM   dettaglio
      group by D_NUMF;
…
    open cursore_importi;
    LOOP
       FETCH cursore_importi into vr_importi;
       EXIT WHEN cursore_importi%NOTFOUND;
       if vr_importi.IMPORTO < 1000 then
          v_TotSmall := v_TotSmall + vr_importi.IMPORTO;
       else
         v_TotBig := v_TotBig + vr_importi.IMPORTO;
       end if;
   END LOOP;
```

94

Procedurale vs Dichiarativo

- Un esempio: restituire in output l'importo totale delle fatture che hanno singolarmente un importo > 1000 e <= 1000

```
CURSOR cursore_importi IS

 SELECT SUM(IMPORTO) INTO v_TotBig
 FROM (SELECT D_NUMF,sum(D_QTA*D_PREZZO) as IMPORTO
       FROM   dettaglio
       GROUP BY D_NUMF
       HAVING IMPORTO > 1000);
…
 SELECT SUM(IMPORTO) INTO v_TotSmall
 FROM (SELECT D_NUMF,sum(D_QTA*D_PREZZO) as IMPORTO
       FROM   dettaglio
       GROUP BY D_NUMF
       HAVING IMPORTO <= 1000);
```

- **Meno efficiente scandisce 2 volte il database: il calcolo della clausola having non può sfruttare strutture a indice**

95

Sommario

- Tipi di cursore:
  - Impliciti: Utilizzati in tutti i comandi DML e per le query single-row.
  - Espliciti: Utilizzabili per le query a 0,1 o più righe.
- I cursori espliciti devono essere gestiti dal codice
  - DECLARE
  - OPEN
  - FETCH
  - CLOSE
- Lo stato del cursore può essere valutato utilizzando i suoi attributi

96

Esercizi in aula

- EX11: Scrivere una funzione che verifichi se un certo prodotto p è presente in quantità > q
- EX12: Scrivere una procedura che emetta un ordine per ogni prodotto presente in quantità < 100;

97

Esercizi sui trigger

- EX13: Scrivere trigger che all'inserimento di un nuovo dettaglio d'ordine aggiorni la quantità disponibile per il prodotto
- EX14: Scrivere un trigger che prima di accettare un ordine verifichi che il cliente non abbia superato il valore massimo degli acquisti, ossia che la somma degli importi acquistati e non pagati non superi 10000. In tal caso l'ordine non deve essere inserito e deve essere visualizzato il messaggio "Superato il credito massimo"

98