Lezione 13

DBMS attivi

Prima parte

DBMS attivi

- In molti casi conviene far eseguire codice applicativo direttamente dal DBMS
  - all'interno dunque del livello logico dei dati
  - con procedure interne alla base di dati che devono essere attivate
    - *stored procedure*
  - o che si avviano automaticamente
    - *trigger*
- A tal fine sono nati dei nuovi linguaggi di programmazione, quali il PL/SQL, che
  - espandono SQL con costrutti di controllo (es. if, for, ecc.) e strutture dati.
  - vengono interpretati ed eseguiti direttamente dal DBMS

Da statici a dinamici

- I DBMS nascono come applicazioni per il deposito di dati
  - per conservare in maniera sicura ed efficiente le informazioni di interesse di una specifica realtà
- Oggi non sono solo magazzini di informazioni ma sistemi attivi capaci di svolgere operazioni complesse eseguite *automaticamente e autonomamente*
- Dotati di componenti con comportamento *reattivo che* vengono definiti una sola volta e in maniera indipendente e condivisa da tutte le applicazioni che interagiscono con la base di dati
  - per uno sviluppo più veloce e una manutenzione più facile.
  - che possono però peggiorare le prestazioni perchè impegnano il server che esegue il DBMS con un carico aggiuntivo

Una base dati attiva

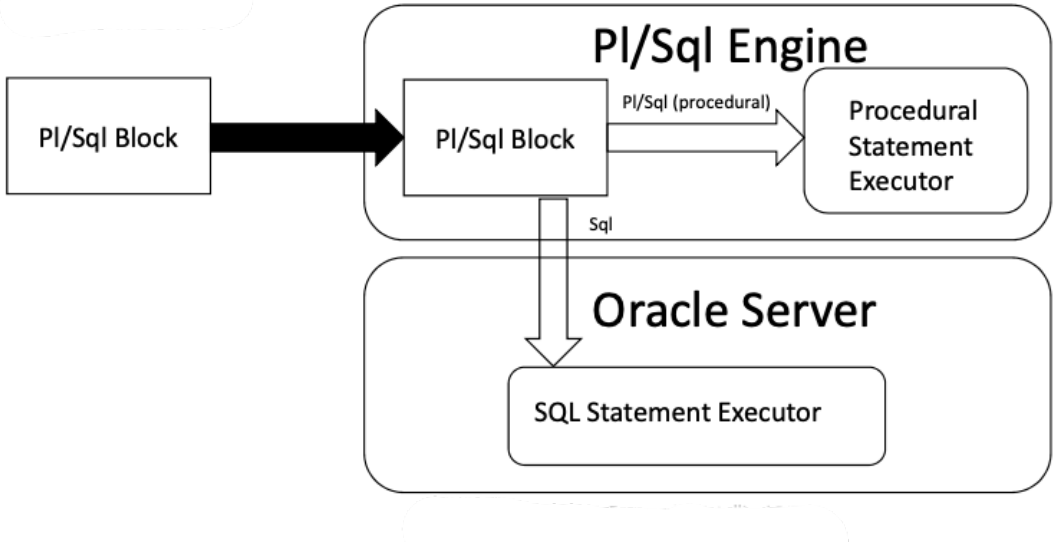
- Una base dati si dice attiva quando dispone di un sottosistema integrato di definizione e gestione di regole di produzione
- Comportamento reattivo invece che passivo
  - alterna l'esecuzione di transazioni (lanciate dagli utenti) e quella delle regole (lanciate dal sistema).
- Indipendenza della conoscenza
  - la conoscenza relativa al comportamento reattivo viene sottratta al programma applicativo e codificata, sotto forma di regole, nello schema del database.

PL/SQL

- PL/SQL è un linguaggio procedurale estensione del linguaggio SQL
- Gli statement SQL sono “nativamente” integrati nel linguaggio PL/SQL ed è possibile chiamare PL/SQL direttamente dalla linea di comando
- Oracle Database 11g ha anche evoluto PL/SQL da linguaggio interpretato a linguaggio nativamente compilato
- Permette di scrivere codice una volta e poi effettuare il deploy nel database insieme con i dati
- Può semplificare lo sviluppo di applicazioni, ottimizzare esecuzioni e migliorare l'utilizzo delle risorse nel database
- PL/SQL è un linguaggio di programmazione procedurale standard e quindi ha funzioni, procedure, dichiarazioni di variabili, loop, ricorsione, etc.
  - è simile ad altri linguaggi di programmazione procedurale non-object-oriented
  - ha stretta integrazione con SQL ed è facile e naturale incorporare Sql in PL/SQL piuttosto che farlo in qualsiasi altro linguaggio di programmazione

Il motore

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



- I blocchi PL/SQL sono processati dal motore PL/SQL che fa parte del Server Oracle
- Il motore PL/SQL filtra i comandi SQL e li invia al server Oracle mentre esegue direttamente i comandi procedurali
- Diminuisce il carico della rete in quanto raggruppa tutte le richieste in un blocco Pl/Sql che va in esecuzione con una sola chiamata

Struttura a blocchi (Block)

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

A cosa servono

- Blocco Anonimo: blocco senza nome
  - interno a una applicazione
  - o eseguito in modo interattivo
- Store procedure: blocco memorizzato all'interno di Oracle Server con parametri di Input/Output;
- Function: come le procedure ma prevedono il ritorno di un risultato
- Package: insieme di funzioni e procedure
- Trigger: blocco eseguito in automatico al verificarsi di un evento

Esecuzione di un blocco

- 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

Struttura di un blocco anonimo

- Sezione di dichiarazione (clausola DECLARE)
  - costanti, variabili, cursori, ecc.
  - nella definizione delle procedure e funzioni la clausola è implicita
- Sezione di esecuzione
  - obbligatoria
- Sezione di gestione delle eccezioni
  - eseguita quando si presentano errori
  - opzionale

Sintassi

```
[DECLARE]
    dichiarazioni
BEGIN
     istruzioni e comandi SQL
[EXCEPTION
     gestione errori]
END;
```

- Ogni istruzione termina con un punto e virgola (;), compreso END
- Fanno eccezione DECLARE, BEGIN ed EXCEPTION

Commenti

- Commento su singola riga

```
-- testo
```

- Commento su più righe

```
/* testo
   testo
    testo */
```

Delimitatori

Simbolo	Descrizione	Simbolo	Descrizione
+	Addizione	>=	Maggiore o uguale
-	Sottrazione	<=	Minore o uguale
*	Moltiplicazione	<>	Diverso
/	Divisione	!=	Diverso
=	Uguale	||	Concatenazione
@	Accesso remoto	--	Commento
;	Fine istruzione	/*	Inizio commento
>	Maggiore	*/	Fine commento
<	Minore	:=	Assegnazione

Dichiarazione di variabile

nome_var **[CONSTANT]** tipo **[NOT NULL] [:=** expr **| DEFAULT** expr**];**

- tutti gli identificatori di utente sono al massimo 30 caratteri, non sono case sensitive, cominciano sempre con una lettera, non possono contenere spazi ma il carattere _ sì
- := e DEFAULT fissano un valore inziale, altrimenti si inizializza a NULL se non è stato specificato NOT NULL
- CONSTANT congela il valore iniziale dato in fase di dichiarazione
- ogni variabile per essere usata deve essere stata dichiarata
- per ogni blocco esistono precise regole di visibilità
  - le variabili di un blocco sono locali a esso e visibili ai blocchi più interni
  - due variabili possono avere lo stesso nome a patto che siano dichiarate in blocchi diversi

%TIPE

- Nelle assegnazioni si deve rispettare la compatibilità dei tipi
- Quando si assegna a una variabile il valore di un elemento di una tabella può capitare che i tipi non corrispondano più se la definizione della tabella è stata modificata
- PL/SQL definisce il tipo *ancorato* (ancored) che fa si che se cambia la definizione della tabella cambia anche il tipo della variabile a runtime
- %TYPE consente di dichiarare una variabile in funzione
  - del tipo di una colonna di una tabella
  - del tipo di un'altra variabile precedentemente dichiarata
- Sintassi

```
nome_var1  nome_tab.nome_campo%TYPE
nome_var2  nome_var_predefinita%TYPE
```

Assegnazione di valore

- Semplice

nome_var := expr;

- Tramite select

SELECT valore INTO nome_var FROM nome_tabella;

- La select deve restituire un unico valore

  SELECT max(stipendio) INTO massimo_stipendio FROM dipendenti;

- Più in generale

SELECT colonna(e) INTO lista_variabili_corrispondenti
FROM tabella(e) WHERE condizione;

- Si possono assegnare i risultati di una query a variabili diverse dello stesso tipo dei valori ottenuti solo nel caso in cui la select produce una sola tupla
- devo garantire altresì che almeno una tupla venga generata

Gestione NULL

- I valori nulli si gestiscono tramite
  - IS NULL
  - oppure IS NOT NULL
- Il risultato dell'operazione NULL +-*/ (operando)
  - è sempre NULL
- Il risultato tra

  NULL <, >, <=, >=, =, <>, != (valore)
  - è sempre NULL

Funzioni predefinite sulle stringhe

- UPPER(<stringa>)
  - converte tutte le lettere della stringa in lettere maiuscole. Per es.
- LOWER(<stringa>)
  - converte tutte le lettere della stringa in lettere minuscole
- INITCAP(<stringa>)
  - converte la lettera iniziale di ogni parola della stringa in una lettera maiuscola.
- SUBSTR(<string>, n [,m])
  - estrae una sottostringa dalla stringa iniziale, partendo dalla posizione n inclusa, per m caratteri (o fino alla fine se m è omesso)
- CHAR(n)
  - restituisce il simbolo in posizine n della codifica ASCII
- LENGTH(s):
  - calcola la lunghezza della stringa s

Funzioni sulle stringhe

- CONCAT
  - La funzione CONCAT tra due sole stringhe è equivalente all'operatore di concatenazione ||
- INSTR
  - La funzione INSTR riceve in input due stringhe. Ritorna un numero che rappresenta a che posizione, all'interno della prima stringa, si trova la seconda stringa. La funzione torna zero nel caso in cui la seconda stringa non è contenuta nella prima.
- LTRIM, RTRIM e TRIM
  - La funzione LTRIM riceve in input due stringhe. Elimina dalla sinistra della prima tutte le occorrenze della seconda stringa. La funzione TRIM elimina gli spazi sia dalla destra che dalla sinistra di una stringa
- LPAD ed RPAD
  - La funzione LPAD riceve in input due stringhe ed una lunghezza lpad(nome,10,'*'). Effettua il riempimento della prima stringa con la seconda stringa fino a raggiungere la lunghezza data. La funzione RPAD si comporta esattamente come la LPAD ma il riempimento avviene sulla destra della stringa
- REPLACE
  - La funzione REPLACE riceve in input tre stringhe ed effettua una sostituzione, nella prima stringa tutte le occorrenze della seconda stringa vengono sostituite con occorrenze della terza.

Funzioni di conversione

- TO_CHAR
  - La funzione TO_CHAR consente di convertire in stringa un numero oppure una data
- TO_DATE
  - La funzione TO_DATE consente di convertire una stringa in data
- TO_NUMBER
  - La funzione TO_NUMBER converte una stringa in numero

Funzioni e costanti temporali

- DATE(v), TIME(v), TIMESTAMP(v)
  - convertono rispettivamente un valore scalare in una data, un tempo, un timestamp
- CHAR(d, [f])
  - converte un valore di data/tempo d in una stringa di caratteri; puo` inoltre ricevere una specifica di formato f.
- DAYS(v)
  - converte una data v in un intero che rappresenta il numero di giorni a partire dall'anno zero.
- CURRENT DATE
  - rappresenta la data corrente
- CURRENT TIME
  - rappresenta l'ora corrente
- CURRENT TIMESTAMP
  - rappresenta il timestamp corrente

La gestione delle date

- Per prima cosa bisogna capire come è stato configurato ORACLE

  SELECT * FROM nls_session_parameters;

- Oracle consente di memorizzare, processare e recuperare i dati nella lingua nativa, mediante le impostazioni del National Language Support (NLS)
  - consente che le utility del database, i messaggi d'errore, le date, le convenzioni numeriche, sulla valuta e relative al calendario siano automaticamente impostate sulla lingua locale

La gestione delle date

```
1  SELECT *  FROM nls_session_parameters;
2
```

PARAMETER	VALUE
NLS_LANGUAGE	AMERICAN
NLS_TERRITORY	AMERICA
NLS_CURRENCY	$
NLS_ISO_CURRENCY	AMERICA
NLS_NUMERIC_CHARACTERS	.,
NLS_CALENDAR	GREGORIAN
NLS_DATE_FORMAT	DD-MON-RR
NLS_DATE_LANGUAGE	AMERICAN
NLS_SORT	BINARY
NLS_TIME_FORMAT	HH.MI.SSXFF AM
NLS_TIMESTAMP_FORMAT	DD-MON-RR HH.MI.SSXFF AM
NLS_TIME_TZ_FORMAT	HH.MI.SSXFF AM TZR
NLS_TIMESTAMP_TZ_FORMAT	DD-MON-RR HH.MI.SSXFF AM TZR
NLS_DUAL_CURRENCY	$
NLS_COMP	BINARY
NLS_LENGTH_SEMANTICS	BYTE
NLS_NCHAR_CONV_EXCP	FALSE

Download CSV

17 rows selected.

La gestione delle date

- I parametri possono essere modificati con:
  ALTER SESSION SET *campo=espressione*;

Esempi:

```
ALTER SESSION SET nls_date_format = 'dd/mm/yyyy HH:MI:SS’;

ALTER SESSION SET nls_language= ‘ITALIAN’;

ALTER SESSION SET nls_TERRITORY= ‘ITALY’;
```

La gestione delle date

- Per operare sulla rappresentazione interna delle date si devono usare le due funzioni
- TO_DATE(*stringa, formato*)
  - che converte una stringa in un valore di tipo data secondo il formato specificato
  - to_date('11-11-2000','DD-MM-YYYY')
- TO_CHAR(*data, formato*)
  - che converte il campo data in una stringa secondo il formato specificato
  - to_char(data_nascita,'DD-MM-YYYY')

Formati per le date e il tempo

Maschera	Descrizione
DD	Giorno del mese a due cifre
MON	Abbreviazione (tre lettere) del mese
YY	Anno a due cifre
YYYY	Anno a quattro cifre
RR	Anno a due cifre compatibile con l'anno 2000
CC	Secolo a due cifre
HH	Ore con AM e PM
HH24	Ore con formato 24
MI	Minuti
SS	Secondi

Esempi:

'YYYYMMDD HH24:MI:SS'

'YYYY-MM-DD HH24:MI:SS’

‘DD/MON/YYYY HH24:MI:SS'

Esempio gestione delle date

```
CREATE TABLE agenda(
        nome      PRIMARY KEY,
        telefono  VARCHAR2(20),
        nascita   DATE,
        CONSTRAINT data_errata CHECK(nascita>TO_DATE('01-01-2000','DD-MM-YYYY')
                                                AND
                                                nascita<TO_DATE('31-12-2100','DD-MM-YYYY'))
);

INSERT INTO prova VALUES('Angelo','348445566',TO_DATE('11-FEB-2000','DD-MON-YYYY'));

SELECT * FROM prova;

INSERT INTO prova VALUES('Antonio','347334455',TO_DATE('19-10-1900','DD-MM-YYYY'));
```

Funzioni matematiche

ABS(numero) Restituisce il valore assoluto
COS(radianti) Restituisce il coseno del numero indicato in radianti
SIN(radianti) Restituisce il seno del numero indicato in radianti
TAN(radianti) Restituisce la tangente del numero indicato in radianti
LOG(numero) Restituisce il logaritmo naturale del numero
LOG10(numero Restituisce il logaritmo decimale del numero
MOD(num1,num2) Restituisce il resto di num1 diviso num2 equivale a: num1%num2
ASCII(char) Restituisce il valore decimale del carattere
BIN(decimale) Restituisce il valore binario di numero indicato in base dieci
FLOOR(numero) Restituisce l'approssimazione per difetto di numero, es. FLOOR(5.3) restituisce 5
EXP(potenza) Restituisce il numero e elevato a potenza
HEX(numero) Restituisce la stringa di conversione in esadecimale di numero
POW(num1,num2) oppure POWER(num1,num2) Restituisce il valore di num1 elevato a num2
PI() Restituisce il valore di pi greco
RAND([seme]) Restituisce un valore casuale compreso tra 0 e 1
ROUND(numero,[decimali]) Restituisce il valore di numero arrotondato al numero di cifre decimali
TRUNCATE(numero,decimali) Restituisce il numero con il numero di decimali troncato
SIGN(numero) Restituisce –1 se il numero è negativo, 1 se positivo, 0 se zero
SQRT(numero) Restituisce la radice quadrata di numero

Tutte le funzioni definite da ORACLE

https://docs.oracle.com/cloud/help/it/reportingcs_use/BILPD/GUID-4CBCE8D4-CF17-43BD-AAEF-C5D614A8040A.htm#BILUG779

Funzioni

Sono disponibili diversi tipi di funzioni che è possibile utilizzare nelle espressioni.

Argomenti:

- Funzioni di aggregazione
- Funzioni analitiche
- Funzioni calendario
- Funzioni di conversione
- Funzioni di visualizzazione
- Funzioni di valutazione
- Funzioni matematiche
- Funzioni di stringa
- Funzioni di sistema
- Funzioni di serie temporali

Costrutti di controllo

- Selezione

IF...THEN...END IF;
IF...THEN...ELSE...END IF;

IF...THEN...ELSIF... ELSIF... ELSE…. END IF;

- Iterazione

LOOP	senza terminazione
FOR LOOP	con conteggio
WHILE LOOP	con condizione di terminazione

- L’istruzione EXIT termina un loop (forzatamente)

Selezione

if <condizione>

then <sequenza di istruzioni>

[elsif] <condizione> then <sequenza di istruzioni>

...

[else] <sequenza di istruzioni>

end if;

LOOP

```
LOOP
        [istruzioni;]
        EXIT [WHEN condizione];
        [istruzioni;]
END LOOP;
```

- Senza EXIT genera un ciclo senza terminazione

WHILE LOOP

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

- Il ciclo termina quando la condizione diventa falsa

FOR LOOP

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

- Ciclo con conteggio da lower a upper e viceversa se è presente REVERSE
  - counter è dichiarata implicitamente

Etichettatura dei LOOP

- I costrutti si possono innescare l'uno nell'altro
- Si possono etichettare tutti i tipi di LOOP con una label racchiusa tra << e >>
  - la stessa label può accompagnare l'END LOOP senza << e >>

```
<<ciclo_esterno>>
WHILE condizione1 LOOP
    statement_a;
    <<ciclo_interno>>
    WHILE condizione2 LOOP
        statement_x;
        IF (condizione) then EXIT ciclo_esterno;
        END IF;
    END LOOP ciclo_interno;
    statement_b;
END LOOP ciclo_esterno;
```

- Allora con exit [<etichetta blocco>] [when <condizione>]
  si esce dal blocco indicato

Tipi di variabili

- Scalar
  - sono i tipi previsti per le tabelle Oracle più pochi altri (es: Boolean) per contenere un singolo valore
- Composite
  - sono strutture come record e tabelle
- Reference
  - sono i classici puntatori
- LOB (Large OBjects)
  - sono elementi, chiamati locators, che specificano la posizione di oggetti di grosse dimensioni (es. immagini) che sono memorizzati separatamente

Definizione di tipo

TYPE nome_tipo IS definizione;

- E nome_tipo può essere usato per dichiarare una variabile

- Esempio

```
TYPE contatto      IS  RECORD (nome        VARCHAR2(40),
                               cognome     VARCHAR2(60),
                               numero_tel  VARCHAR2(15));
TYPE agenda        IS TABLE OF contatto ;

--dichiarazione di una variabile di tipo agenda
mia_rubrica  agenda;
```

I principali tipi di dati scalari

```
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
……
```

I record

- Sono le struct del C
  - un insieme di campi con proprio nome e tipo individuati da un unico nome
- La struttura di un record corrisponde allo schema di una relazione
- Il valore di un record corrisponde a una tupla dell'istanza dello schema di relazione corrispondente alla definizione del record
- Il record si presta a memorizzare il risultato completo di una query nel caso essa fornisca una sola tupla
  - O a gestire una tupla alla volta di quelle prodotte

Dichiarazione di record

```
TYPE tipo_record IS
        RECORD(nome_campo {tipo_campo |
                                    variabile%TYPE |
                                    tabella.colonna%TYPE }
                                    [[NOT NULL]
                                   { := | DEFAULT } espressione ]
                  [,nome_campo ... ]
                  ……
                  );

-- dichiarazione di variabile di tipo record
var_record   tipo_record;
```

%ROWTYPE

var_record nometabella%ROWTYPE;

- Dichiara una variabile di tipo record basandosi sull'insieme dei campi appartenenti a una tabella, vista o cursore
- Basta anteporre a %ROWTYPE il nome della tabella, vista o cursore a cui il record è associato
- I campi nel record assumono il nome e il tipo di quelli della tabella, vista o cursore.
- Si accede a un campo del record con

  var_record.nome_colonna_della_tabella

- Anche se il tipo e il numero delle colonne nel data base cambia, le due strutture restano ancorate

Assegnare valore

```
DECLAIRE
   var_record   nometabella%ROWTYPE;

BEGIN
    ……
    SELECT * INTO var_record FROM nometabella
                                    WHERE condizione;
    ……
END;
```

- Ovviamente la select deve restituire una sola tupla

Dichiarazione di tabella

```
TYPE tipo_tab IS TABLE
        OF tipo_campo |
            variabile%TYPE |
            nome_tabella.nome_colonna%TYPE
            [NOT NULL]
            [INDEX BY BINARY INTEGER];

-- dichiarazione di una variabile di tipo tabella
nome_var_tabella   tipo_tab;
```

La tabella

- Si compone
  - di una sola colonna con un campo scalare
  - oppure di un insieme di record (le tuple della tabella)
- È indicizzabile con una chiave primaria di tipo BINARY INTEGER
  - L'accesso ai dati di una tabella con una sola colonna è:
    - nome_var_tabella(valorechiave)
  - L'accesso ai dati di una tabella basata su Record è
    - nome_var_tabella(valorechiave).nome_colonna
- Sono dinamiche, cioè non hanno una dimensione fissa

Esempio di vettore

```
TYPE vettore
        IS TABLE OF INTEGER
        INDEX BY INTEGER;

-- dichiarazione
v   vettore;
-- uso
v(1) := espressione
```

Esempio di array di record

```
TYPE Tipo_Impiegato
IS RECORD (codice fiscale CHAR(16),
                nome          VARCHAR2(50),
                cognome       VARCHAR2(50)
               );

TYPE Tabella_Impiegati
        IS TABLE OF Tipo_Impiegato
        INDEX BY INTEGER;

IMPIEGATI Tabella_Impiegati;
IMPIEGATI(1).Nome:='Moscato';
```

Metodi di accesso alla tabella

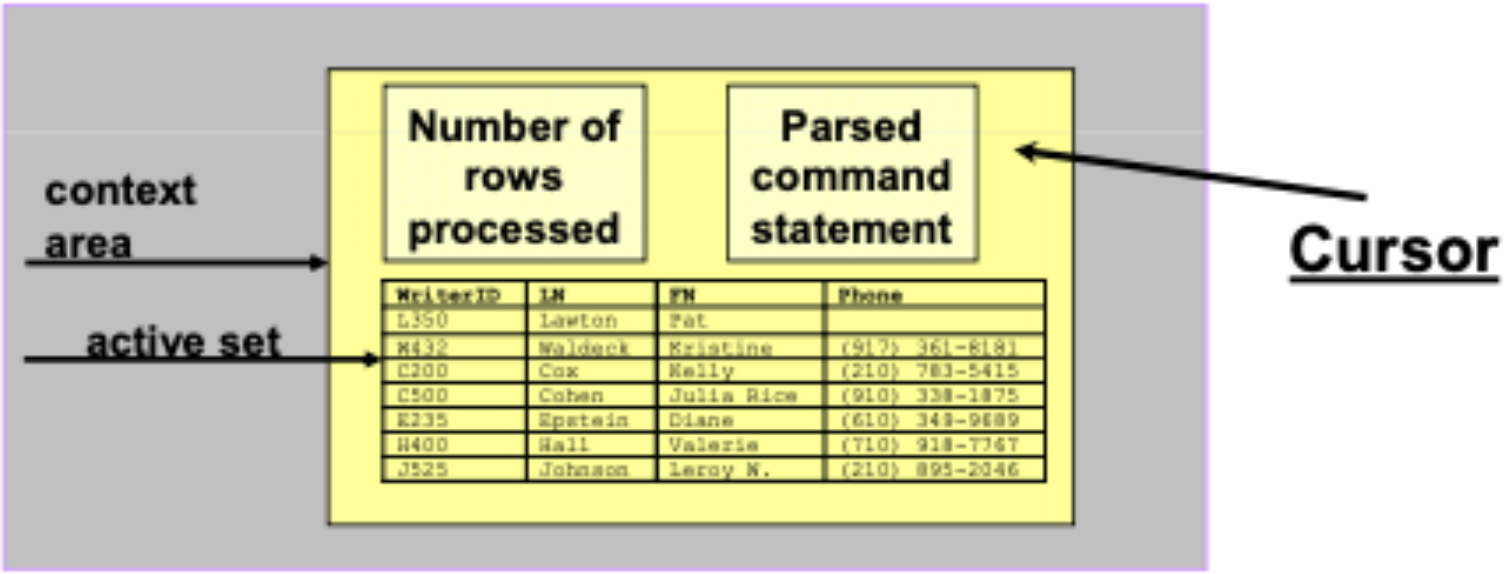
- Un metodo è una operazione predefinita che opera sulla tabella

METODO	DESCRIZIONE
EXISTS(*n*)	Restituisce TRUE se esiste l'elemento *n* della tabella
COUNT	Numero di elementi contenuti in tabella
FIRST - LAST	Restituisce il primo e l'ultimo elemento, NULL se la tabella è vuota
PRIOR(*n*)	Restituisce l'indice che precede l'indice *n*
NEXT(*n*)	Restituisce l'indice che segue l'indice *n*
TRIM(*n*)	Rimuove *n* elementi dalla fine della tabella
DELETE	Rimuove tutti gli elementi di una tabella; DELETE(*n*) rimuove l'elemento *n*; DELETE(*m*,*n*) rimuove gli elementi compresi tra *m* a *n*

nome_tabella.nome_metodo[(parametri)]

Il cursore: cosa è

- Per l'esecuzione di un comando SQL ORACLE assegna ad esso un'area di memoria detta *context area*



- L'area *active set* contiene il risultato della query
- I cursori sono puntatori all'area *active set*
  - per cui ad ogni comando SQL è associato un cursore

Due tipi di cursori

- **Cursori Impliciti**
  - PL/SQL fa riferimento al più recente cursore implicito come ***cursore SQL***
  - 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
- **Cursori Espliciti**
  - Sono dichiarati e gestiti direttamente dal codice
  - Sono utilizzati per processare le singole righe restituite da un comando SQL multiple-row
  - Puntano alla riga corrente nell'active set

Attributi dei cursori impliciti

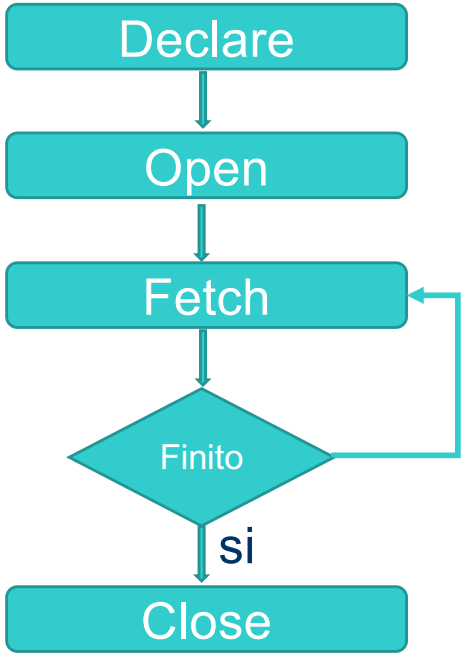
- Oracle consente di verificare alcuni elementi del ***cursore SQL***

Attributo	Descrizione
SQL%ROWCOUNT	Numero di righe coinvolte dal più recente comando SQL
SQL%FOUND	Attributo Booleano 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

Cursori Espliciti

- Usati per accedere all'active set di una SELECT contenente i risultati della query
- L'accesso è ad ogni riga prodotta dalla SELECT, una sola per volta

- Operazioni sui cursori
  - Dichiarazione
  - Apertura
  - Fetch
  - Chiusura



Dichiarazione

CURSOR cursor_name IS select_statement;

- Come select_statement si può usare un qualsiasi comando SELECT
  - Può includere join, operatori di set e subquery
- Se è necessario processare le righe in una sequenza ordinata si deve utilizzare nella query la clausola ORDER BY
- L'istruzione select NON deve contenere la clausola INTO perchè la variabile a cui si da valore è il cursore stesso

Apertura

OPEN cursor_name;

- Alloca la memoria necessaria per memorizzare il risultato della query
- Esegue l'analisi sintattica e semantica della query
- Esegue l'interrogazione e identifica l'active set
- Posiziona il puntatore prima della prima riga dell'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

FETCH

FETCH cursor_name

INTO [variable1, variable2, ...] | record_name];

- I dati possono essere inseriti in un record o in un insieme di variabili
- La FETCH recupera le righe dell'active set una alla volta
- Dopo un FETCH, il cursore avanza alla riga successiva dell'active set e la riga letta può essere manipolata per ulteriori elaborazioni
- Dopo ogni FETCH è necessario verificare se il cursore contiene ancora 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

Chiusura

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 l'eccezione INVALID_CURSOR
  - La riapertura del cursore provocherà la riesecuzione della select

Attributi

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)

- cursor_name%ATTRIBUTO

Modo di uso 1

```
BEGIN
    OPEN cursor_name;
    LOOP
        FETCH cursor_name INTO rec_name;
        EXIT WHEN cursor_name%NOTFOUND;
        ……. elaborazione;
     END LOOP;
     CLOSE cursor_name;
END;
```

Modo di uso 2: rapido

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

- Si evita di:
  - Dichiarare il record
  - Aprire il cursore
  - Effettuare la fetch dei dati
  - Chiudere il cursore

Cursori FOR LOOP

```
FOR rec_name IN (SELECT . . . FROM . . . )
LOOP
    statement1;
    statement2;
    ...
END LOOP;
```

- Si evita di:
  - Dichiarare il cursore
  - Aprire il cursore
  - Effettuare la fetch dei dati
  - Chiudere il cursore

Cursori parametrici

CURSOR cursor_name [(nome_parametro tipo, ...)]
IS select_statement;

- Si possono passare i valori dei parametri ad un cursore quando esso viene aperto e la query viene eseguita
- Per esempio per aprire un cursore esplicito più volte, utilizzando ogni volta un active set diverso

Uso dei parametri

```
CREATE PROCEDURE esempio IS
    CURSOR writer_cursor (p_flstatus IN writer.freelancer%TYPE)
                IS SELECT name, 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.name,40) || vr_writer.phone);
    END LOOP;
    CLOSE writer_cursor;
END;
```

- Per cercare gli scrittori free o meno

Clausola FOR UPDATE della SELECT

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

- Consente di bloccare le righe selezionate in modo che altri utenti non possono bloccarle o aggiornarle fino al termine della transazione.
- Non è in grado di bloccare le righe se sono già bloccate da un'altra transazione. Per impostazione predefinita, la sessione attenderà il rilascio dei blocchi da parte dell'altra transazione.
- Si può indicare di non attendere il rilascio dei blocchi con NOWAIT
- Per rilasciare il blocco, è necessario emettere COMMIT o ROLLBACK nella sessione che tiene il blocco
- Non può essere utilizzato con l'operatore DISTINCT, CURSOR, GROUP BY o funzioni aggregate.
- Con join di più tabelle, se si desidera bloccare le righe di una particolare tabella, basta specificare il nome di una sua colonna. Senza questa clausola sono bloccate le righe selezionate di tutte le tabelle del join.

Clausola FOR UPDATE in un CURSORE

CURSOR name SELECT ... FROM ...

FOR UPDATE [OF column_reference][NOWAIT];

- 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)

WHERE CURRENT OF

WHERE CURRENT OF cursor_name;

- 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 poichè questa viene applicata alla riga corrente
- È necessario utilizzare FOR UPDATE nella definizione del cursore in modo da applicare un lock sulla tabella
- In caso contrario si verificherà un errore

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 = '718' || substr(phone,4)
                     WHERE CURRENT OF c_stud_zip;
     END LOOP;
     COMMIT;
END;
```

OUTPUT al video

DBMS_OUTPUT.PUT_LINE('stringa caratteri'
[|| variabile | stringa]);

- La procedura scrive l'output su un buffer dell'SGA da cui può essere letto mediante il comando .get_line
- In SQL Developer il risultato del comando compare nella finestra Output DBMS
- DBMS_OUTPUT è un *package e* .PUT_LINE è una sua *procedure*

System Global Area (**SGA**) forms the part of the system memory (RAM) shared by all the processes belonging to a single **Oracle** database instance

Ricordarsi di attivare la modalità

SET SERVEROUT ON

- Il comando serve ad abilitare la visualizzazione dei messaggi stampati dai programmi PL/SQL
- Non è un comando SQL ed è necessario solo per il debugging delle procedure e dei trigger
- Se non si attiva i messaggi non vengono mostrati al video

Esempio

```
DECLAIRE
   CURSOR stampa_fatture IS
         SELECT numero_fattura, sum(qta*prezzo) as importo
          FROM vendite WHERE numero_fattura BETWEEN 1 AND 5
         GROUP BY numero_fattura;
    record_importi stampa_fatture%ROWTYPE;
BEGIN
OPEN stampa_fatture;
LOOP
     FETCH stampa_fatture into record_importi;
     EXIT WHEN stampa_fatture%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE('La fattura: ' || record_importi.numero_fattura
                || ' e'' di importo: ' || record_importi.importo);
END LOOP;
CLOSE stampa_fatture;
END;
```

- Fornisce in output gli importi delle fatture da 1 a 5
- Notare che per stampare e' si devono usare due apici

ACCEPT

ACCEPT *variable* [NUMBER | CHAR | DATE | BINARY_FLOAT | BINARY_DOUBLE] [FORMAT *format*] [DEFAULT *default*] [PROMPT *text* | NOPROMPT] [HIDE]

- Legge una linea dell'input e la memorizza in una variabile

```
ACCEPT pswd CHAR PROMPT 'Password: ' HIDE;
ACCEPT salary NUMBER FORMAT '999.99' DEFAULT '000.0'
            PROMPT 'Enter weekly salary: ';
ACCEPT hired DATE FORMAT 'dd/mm/yyyy' DEFAULT '01/01/2003'
            PROMPT 'Enter date hired: ';
ACCEPT lastname CHAR FORMAT 'A20'
            PROMPT 'Enter employee lastname: ';
```

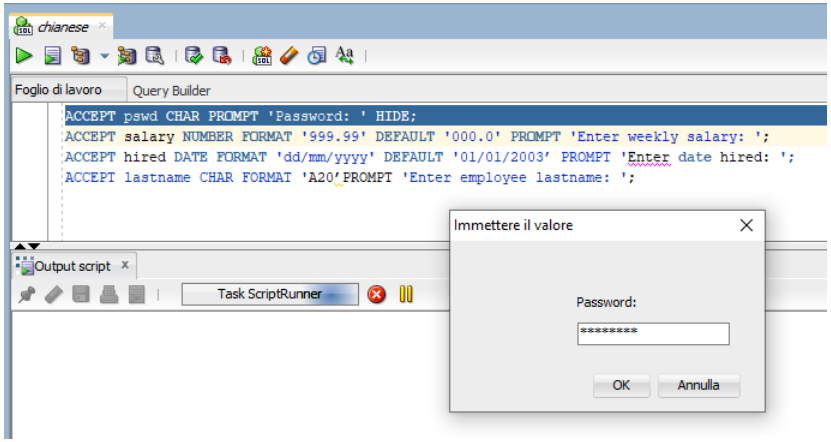
Modalità d’uso

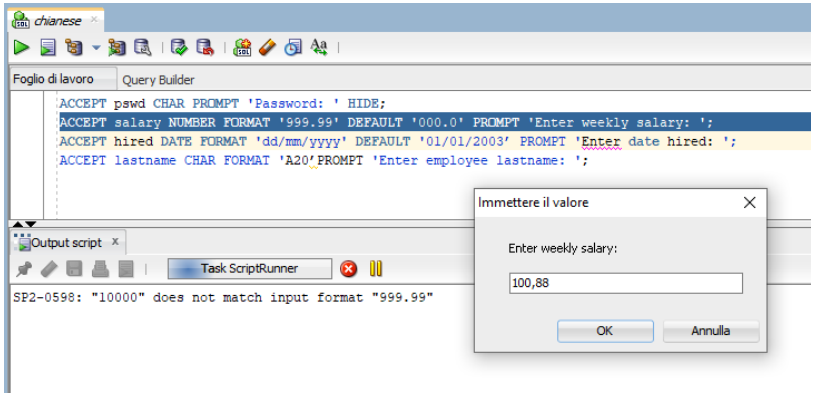
- PL / SQL non è progettato per essere utilizzato in modo interattivo
- ACCEPT è un comando per SQL DEVELOPER, non è un comando PL/SQL
- Si assegnano valori a variabili di sostituzione da usare nei blocchi anonimi facendo riferimento ad esse con

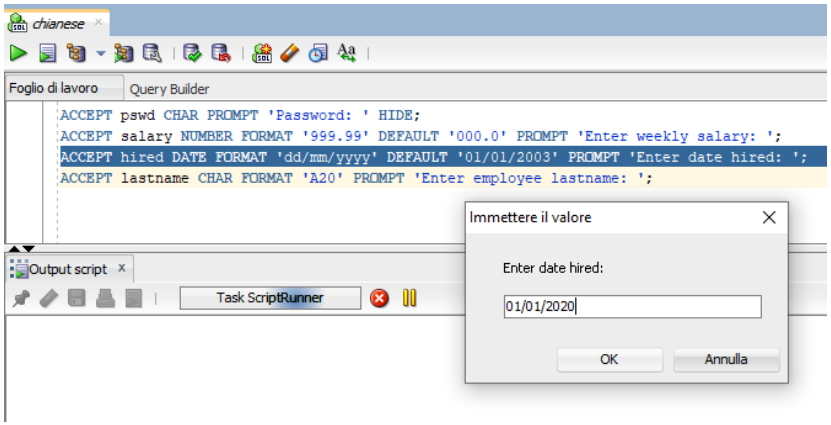
  ‘&nome_var_accept’
- È memorizzata nella tabella di sistema dual

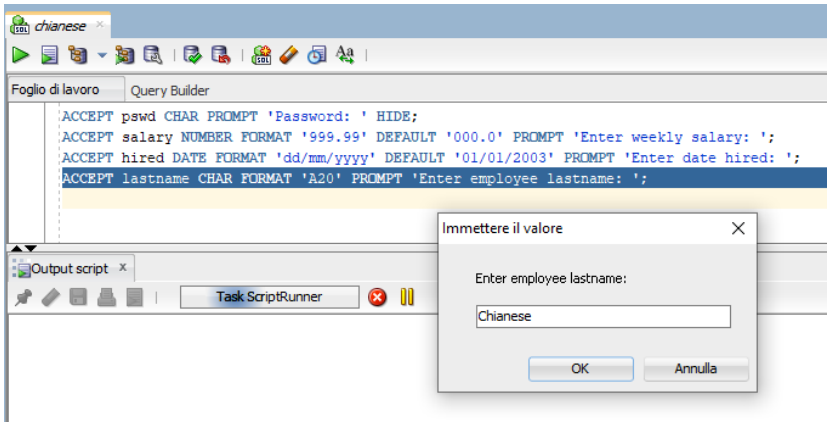
  select '&nome_var_accept' from dual

Esempio

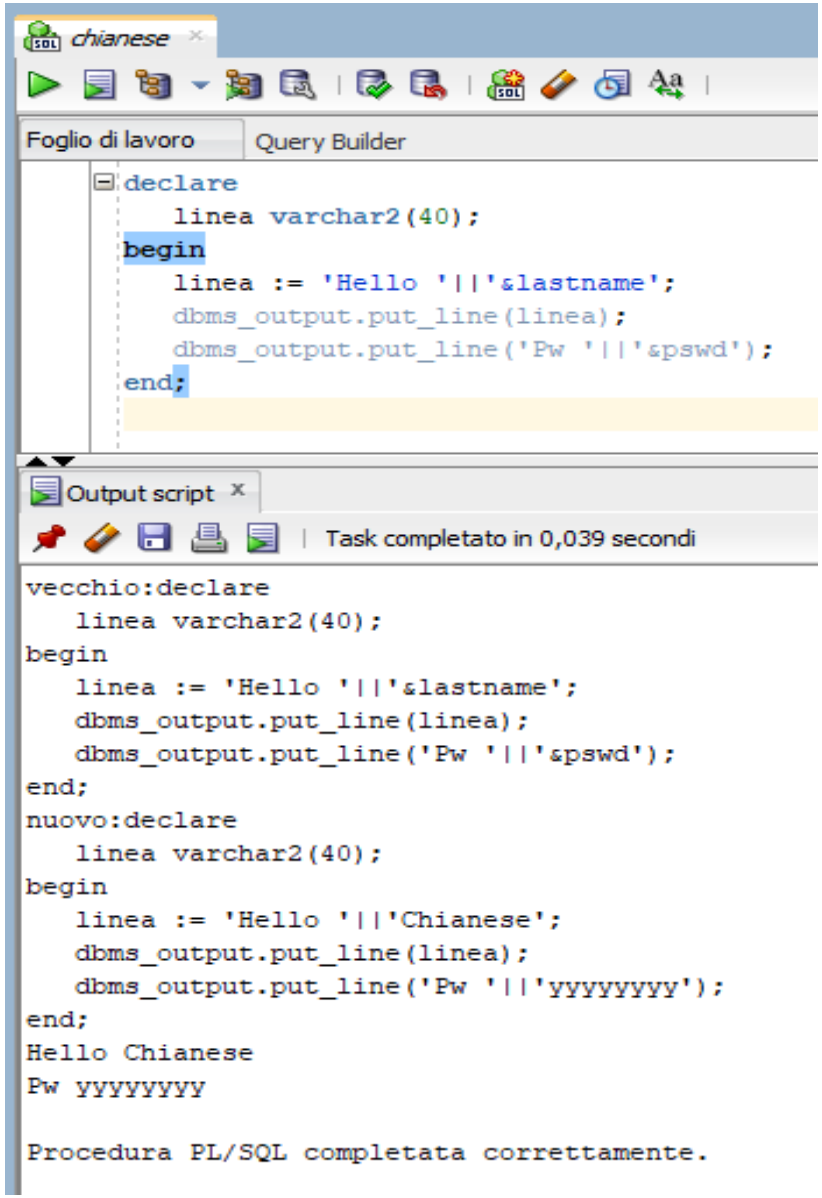

chianese
Foglio di lavoro
Query Builder
ACCEPT pswd CHAR PROMPT 'Password: ' HIDE;
ACCEPT salary NUMBER FORMAT '999.99' DEFAULT '000.0' PROMPT 'Enter weekly salary: ';
ACCEPT hired DATE FORMAT 'dd/mm/yyyy' DEFAULT '01/01/2003' PROMPT 'Enter date hired: ';
ACCEPT lastname CHAR FORMAT 'A20' PROMPT 'Enter employee lastname: ';
Output script
Task ScriptRunner
Immettere il valore
Password:
****
OK
Annulla


chianese
Foglio di lavoro
Query Builder
ACCEPT pswd CHAR PROMPT 'Password: ' HIDE;
ACCEPT salary NUMBER FORMAT '999.99' DEFAULT '000.0' PROMPT 'Enter weekly salary: ';
ACCEPT hired DATE FORMAT 'dd/mm/yyyy' DEFAULT '01/01/2003' PROMPT 'Enter date hired: ';
ACCEPT lastname CHAR FORMAT 'A20' PROMPT 'Enter employee lastname: ';
Output script
Task ScriptRunner
SP2-0598: "10000" does not match input format "999.99"
Immettere il valore
Enter weekly salary:
100,88
OK
Annulla

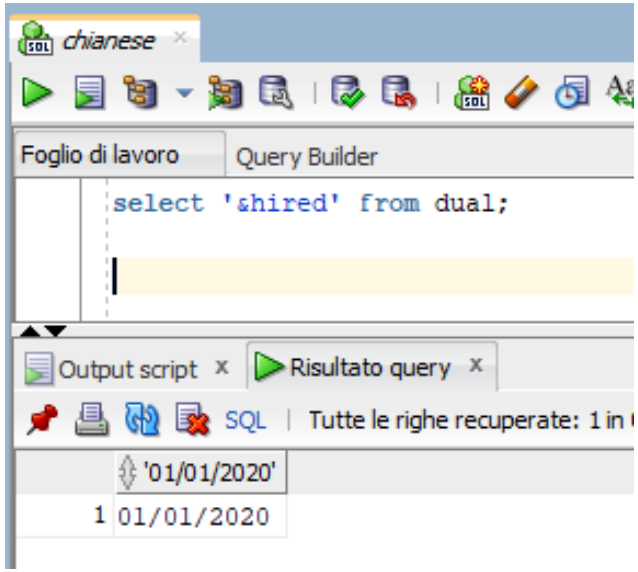

chianese
Foglio di lavoro
Query Builder
ACCEPT pswd CHAR PROMPT 'Password: ' HIDE;
ACCEPT salary NUMBER FORMAT '999.99' DEFAULT '000.0' PROMPT 'Enter weekly salary: ';
ACCEPT hired DATE FORMAT 'dd/mm/yyyy' DEFAULT '01/01/2003' PROMPT 'Enter date hired: ';
ACCEPT lastname CHAR FORMAT 'A20' PROMPT 'Enter employee lastname: ';
Output script
Task ScriptRunner
Immettere il valore
Enter date hired:
01/01/2020
OK
Annulla


chianese
Foglio di lavoro
Query Builder
ACCEPT pswd CHAR PROMPT 'Password: ' HIDE;
ACCEPT salary NUMBER FORMAT '999.99' DEFAULT '000.0' PROMPT 'Enter weekly salary: ';
ACCEPT hired DATE FORMAT 'dd/mm/yyyy' DEFAULT '01/01/2003' PROMPT 'Enter date hired: ';
ACCEPT lastname CHAR FORMAT 'A20' PROMPT 'Enter employee lastname: ';
Output script
Task ScriptRunner
Immettere il valore
Enter employee lastname:
Chianese
OK
Annulla

Due modalità di accesso




```
declare
   linea varchar2(40);
begin
   linea := 'Hello '||'&lastname';
   dbms_output.put_line(linea);
   dbms_output.put_line('Pw '||'&pswd');
end;
```



Le eccezioni

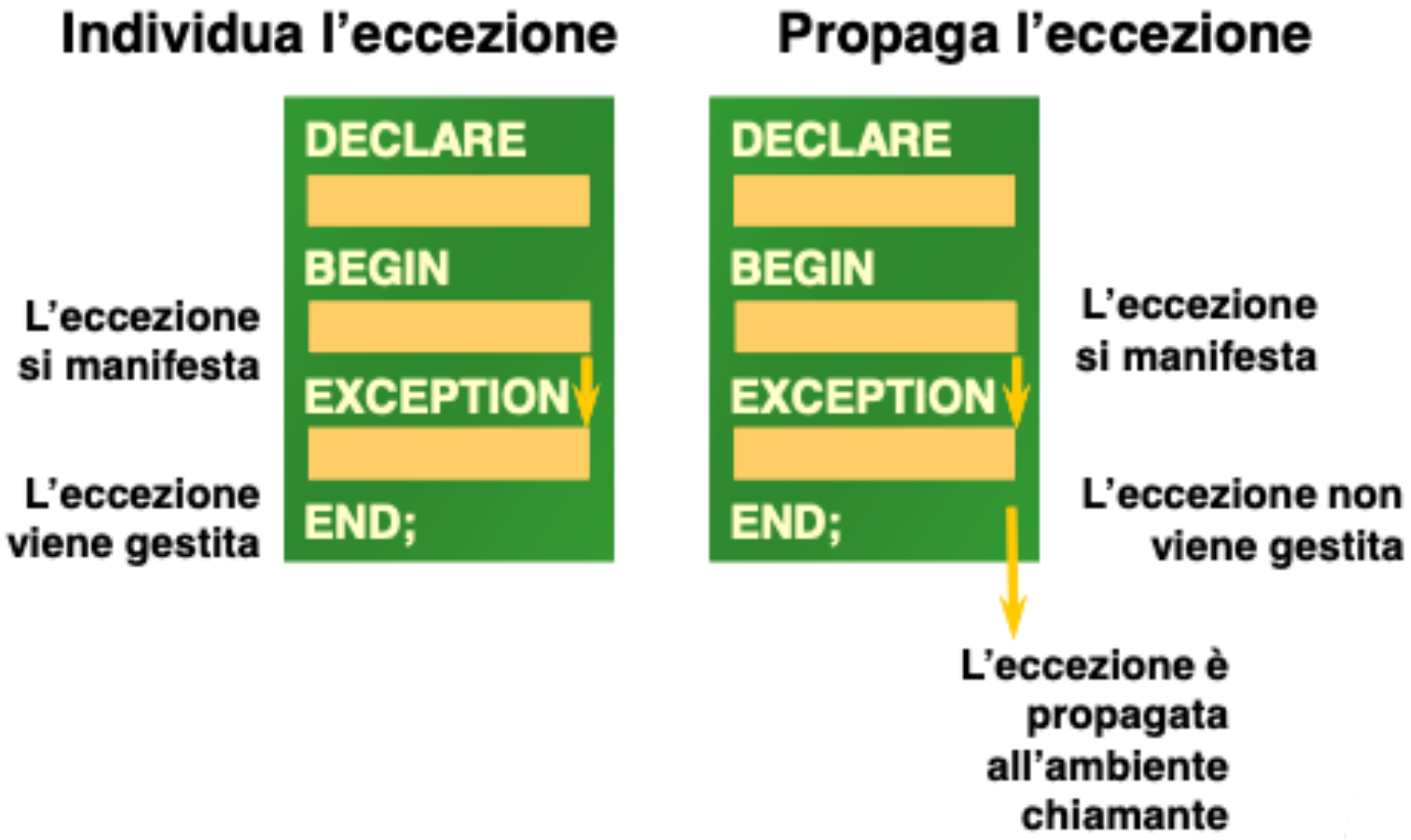
Un qualsiasi blocco può avere EXCEPTION

```
[DECLARE]
    dichiarazioni
BEGIN
     istruzioni e comandi SQL
[EXCEPTION
     gestione errori]
END;
```

Il concetto di eccezione

- Cosa è una exception?
  - Una condizione anomala che si verifica durante l'esecuzione di un blocco valorizzando un identificatore
  - L'esecuzione viene trasferita al gestore dell'eccezione nella sezione exception del blocco
- Come avviene la valorizzazione?
  - Automaticamente (implicitamente) quando si verifica un errore a runtime
  - Esplicitamente se nel codice è presente l'istruzione RAISE di simulazione dell'errore
- Come vengono gestite?
  - Includendo una routine corrispondente nella sezione exception
- Cosa avviene in caso contrario?
  - Il blocco termina con un errore
  - L'eccezione è propagata all'applicazione chiamante
  - Viene mostrato il corrispondente messaggio di errore

Modalità di gestione



La gestione delle eccezioni

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

- 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

Considerazioni

- Si può verificare una sola eccezione per volta
- OTHERS
  - Controlla ogni eccezione non trattata esplicitamente
  - Deve essere l'ultima eccezione nella lista
- Le eccezioni possono essere:
  - Internally defined: vengono attivate dal sistema automaticamente e sono associate a un codice di errore
  - Predefined: vengono attivate dal sistema automaticamente e sono associate a un codice di errore e a un nome
  - User-defined: sono definite e attivate da un utente tramite il comando RAISE

Eccezioni predefinite

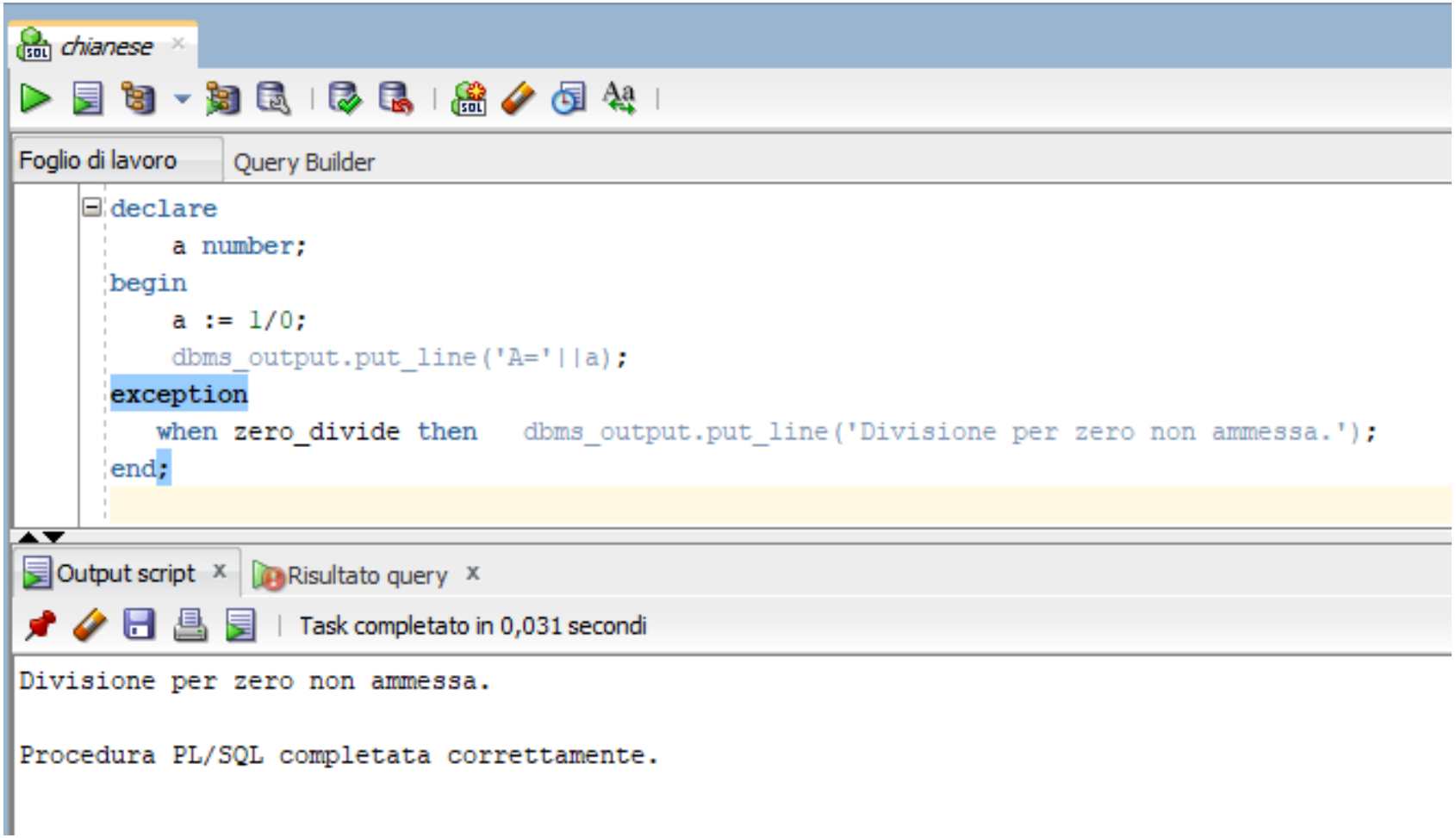
Eccezione	Descrizione
NO_DATA_FOUND (ORA-01403)	SELECT **NON** ha tornato alcun valore (*0 righe*)
TOO_MANY_ROWS (ORA-01422)	SELECT ha tornato più valori (*1 o più righe*)
VALUE_ERROR (ORA-06502)	Si è verificato un errore aritmetico, numerico, di conversione o su un vincolo
ZERO_DIVIDE (ORA-01476)	Tentativo di divisione per 0
DUP_VAL_ON_INDEX (ORA-00001)	Tentativo di duplicare un valore in una colonna soggetta ad un vincolo di univocità
INVALID_NUMBER (ORA-01722)	Conversione da stringa a numero non riuscita

- Due funzioni predefinite per la gestione degli errori
  - **SQLCODE**
    - Restituisce il valore numerico del codice di errore
  - **SQLERRM**
    - Restituisce il messaggio associato al numero di errore
- Possono essere assegnate a variabili e/o mandate in Output

Un semplice esempio

```
DECLARE
    a number;
BEGIN
    a := 1/0;
    dbms_output.put_line('A='||a);
EXCEPTION
    when zero_divide then
            dbms_output.put_line('Divisione per zero non ammessa.');
END;
```

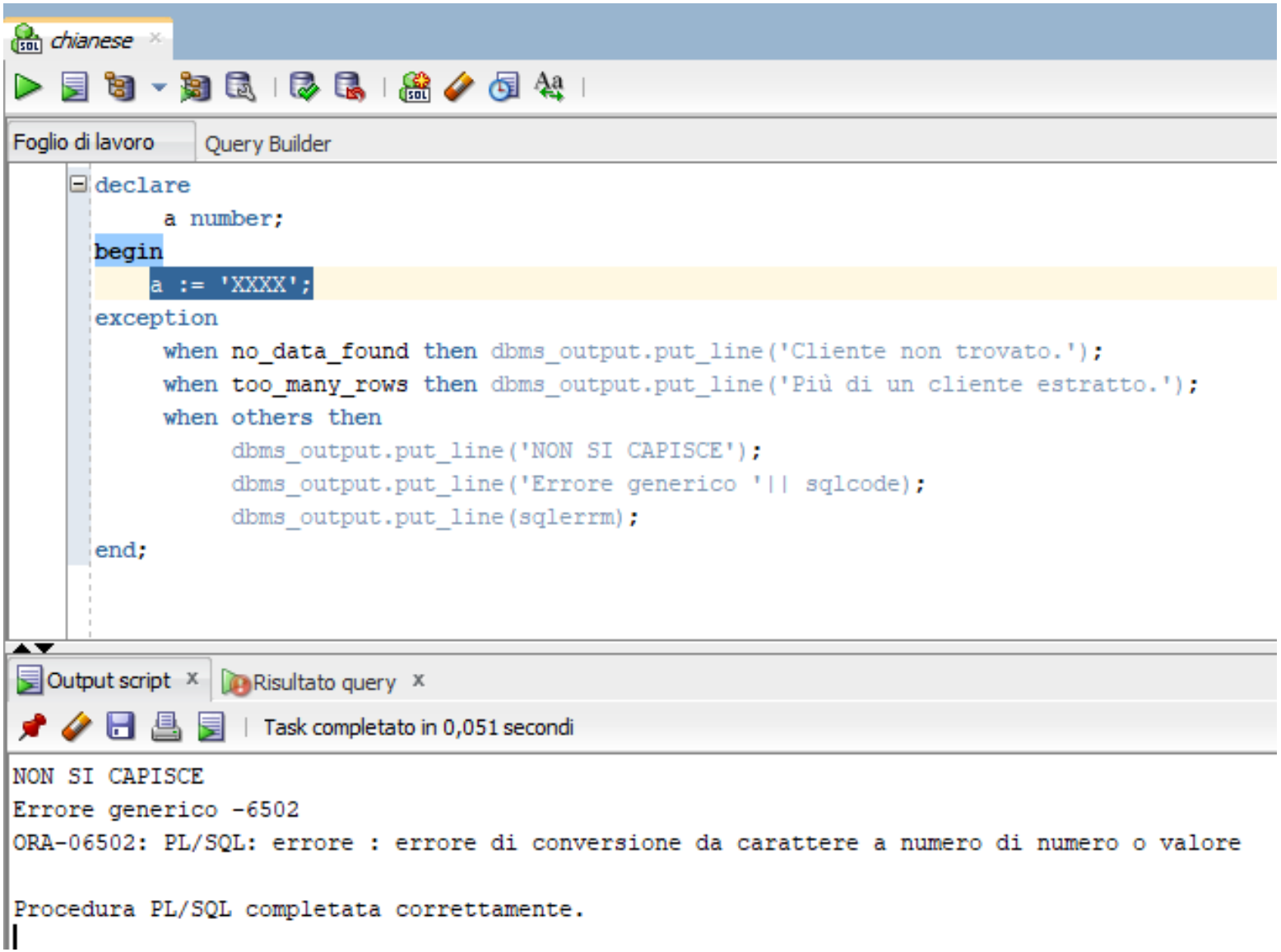
Esempio


chianese
Foglio di lavoro
Query Builder
declare
a number;
begin
a := 1/0;
dbms_output.put_line('A='||a);
exception
when zero_divide then dbms_output.put_line('Divisione per zero non ammessa.');
end;
Output script
Risultato query
Task completato in 0,031 secondi
Divisione per zero non ammessa.
Procedura PL/SQL completata correttamente.

Un altro esempio

```
declare
     a number;
begin
    a := 'XXXX';
exception
     when no_data_found then
          dbms_output.put_line('Cliente non trovato.’);
     when too_many_rows then
          dbms_output.put_line('Più di un cliente estratto.’);
     when others then
          dbms_output.put_line('NON SI CAPISCE');
          dbms_output.put_line('Errore generico ‘|| sqlcode);
          dbms_output.put_line(sqlerrm);
end;
```

Esempio


chianese
Foglio di lavoro
Query Builder
declare
a number;
begin
a := 'XXXX';
exception
when no_data_found then dbms_output.put_line('Cliente non trovato.');
when too_many_rows then dbms_output.put_line('Più di un cliente estratto.');
when others then
dbms_output.put_line('NON SI CAPISCE');
dbms_output.put_line('Errore generico '|| sqlcode);
dbms_output.put_line(sqlerrm);
end;
Output script
Risultato query
Task completato in 0,051 secondi
NON SI CAPISCE
Errore generico -6502
ORA-06502: PL/SQL: errore : errore di conversione da carattere a numero di numero o valore
Procedura PL/SQL completata correttamente.

Eccezioni definite dall'utente

```
DECLARE
    exception_utente EXCEPTION;
BEGIN
    UPDATE nome_tab SET modifiche WHERE condizione;
    IF SQL%NOTFOUND THEN
        RAISE exception_utente;
    END IF;
    COMMIT;
EXCEPTION
   WHEN exception_utente THEN
            statement;
    ...
END;
```

Esempio

chianese | CURATORI

Foglio di lavoro | Query Builder

```
DECLARE
   gestione_curatori EXCEPTION;
BEGIN
   update curatori set anniesperienza=50 where cognome='Russo';
   IF SQL%NOTFOUND THEN
      RAISE gestione_curatori;
   END IF;
   COMMIT;
EXCEPTION
   WHEN gestione_curatori THEN
        dbms_output.put_line('Russo non esiste tra i curatori');
END;
```

Output script | Risultato query

Task completato in 0,039 secondi

```
Russo non esiste tra i curatori

Procedura PL/SQL completata correttamente.
```

Note

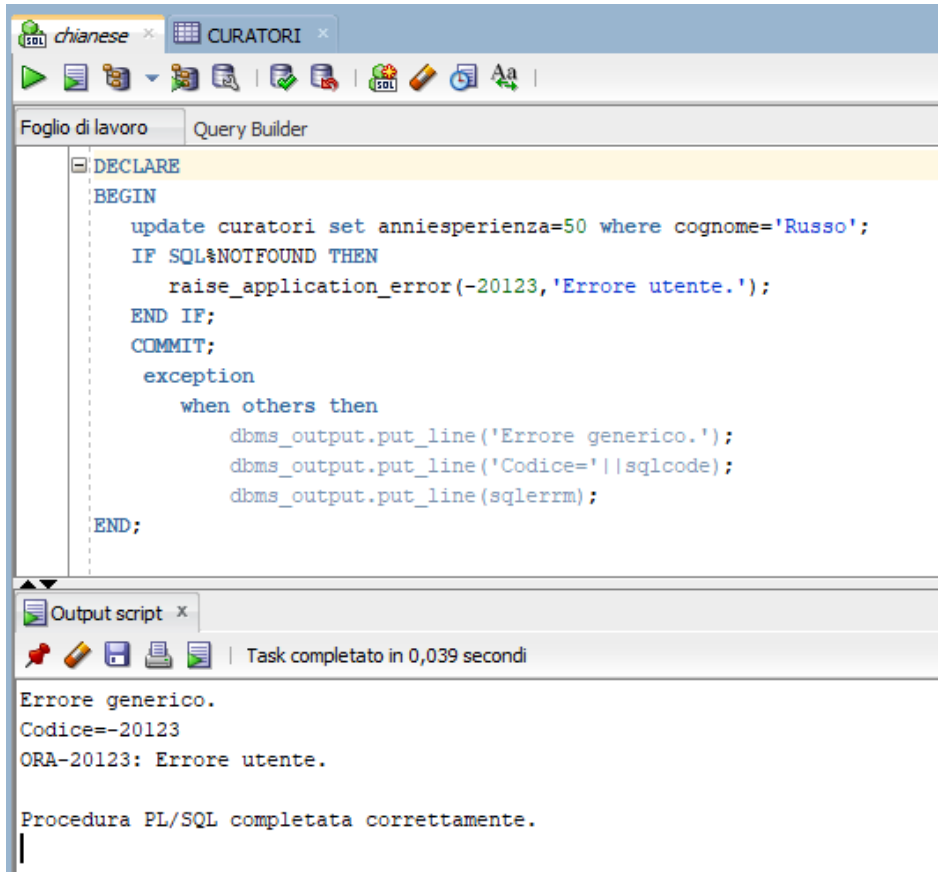
- Il tipo EXCEPTION serve per dichiarare le variabili che possono generare eccezioni
- L'istruzione RAISE serve ad attivare l'eccezione che poi viene gestita nella sezione EXCEPTION
- Per sollevare un errore utente utilizzando un codice d'errore a piacere tra -20000 e -20999 è possibile utilizzare il comando

  RAISE_APPLICATION_ERROR(*codice, messaggio*)

Esempio:

```
RAISE_APPLICATION_ERROR(-20123,'Errore utente.');
```

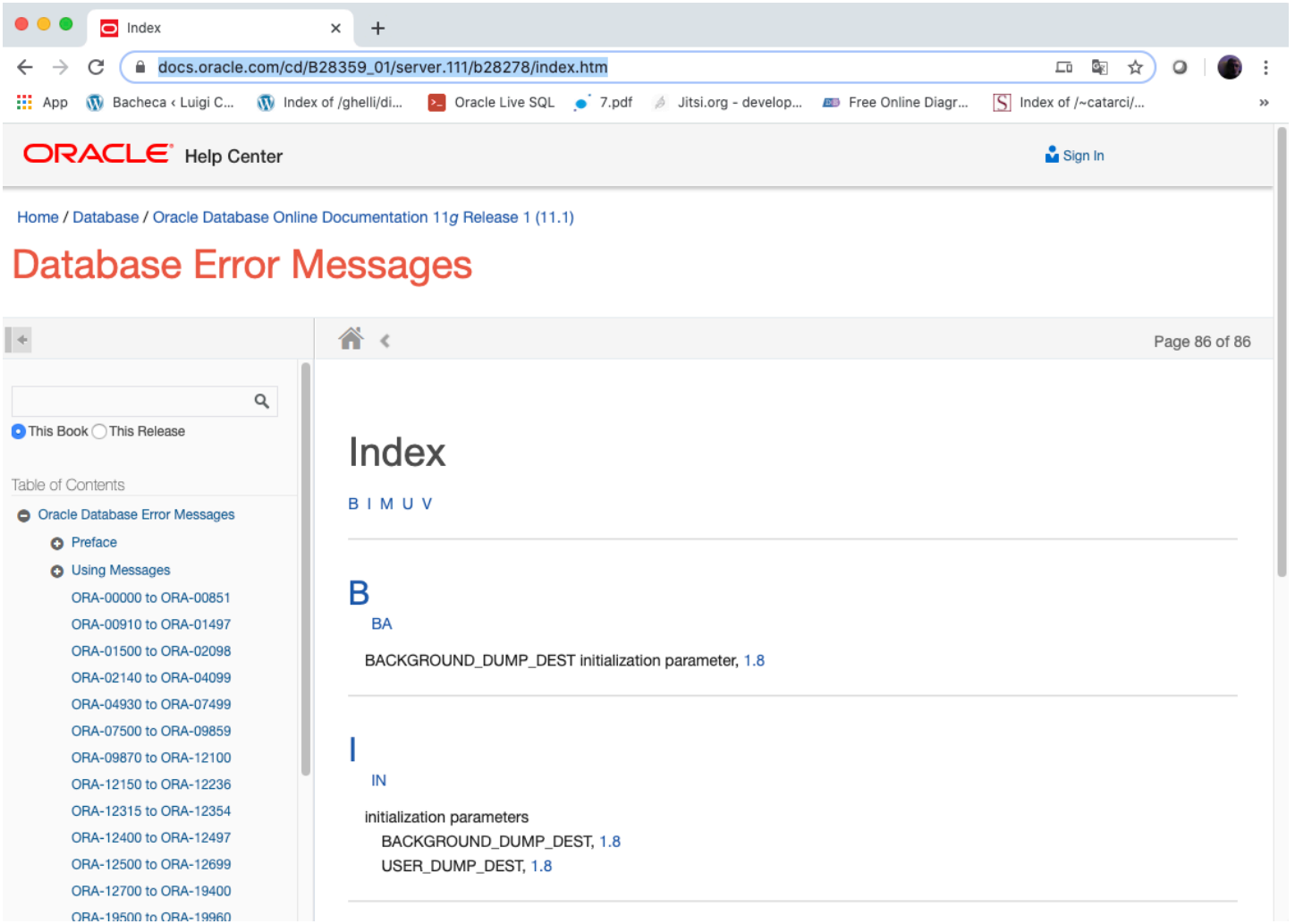
Esempio




```
DECLARE
BEGIN
  update curatori set anniesperienza=50 where
cognome='Russo';
  IF SQL%NOTFOUND THEN
    raise_application_error(-20123,'Errore utente.');
  END IF;
  COMMIT;
   exception
     when others then
        dbms_output.put_line('Errore generico.');
        dbms_output.put_line('Codice='||sqlcode);
        dbms_output.put_line(sqlerrm);
END;
```

Oracle Database Error Message

https://docs.oracle.com/cd/B28359_01/server.111/b28278/index.htm



Exception Name

Oracle Exception Name	Oracle Error	Explanation
DUP_VAL_ON_INDEX	ORA-00001	You tried to execute an INSERT or UPDATE statement that has created a duplicate value in a field restricted by a unique index.
TIMEOUT_ON_RESOURCE	ORA-00051	You were waiting for a resource and you timed out.
TRANSACTION_BACKED_OUT	ORA-00061	The remote portion of a transaction has rolled back.
INVALID_CURSOR	ORA-01001	You tried to reference a cursor that does not yet exist. This may have happened because you've executed a FETCH cursor or CLOSE cursor before OPENing the cursor.
NOT_LOGGED_ON	ORA-01012	You tried to execute a call to Oracle before logging in.
LOGIN_DENIED	ORA-01017	You tried to log into Oracle with an invalid username/password combination.
NO_DATA_FOUND	ORA-01403	You tried one of the following: 1. You executed a SELECT INTO statement and no rows were returned. 2. You referenced an uninitialized row in a table. 3. You read past the end of file with the UTL_FILE package.
TOO_MANY_ROWS	ORA-01422	You tried to execute a SELECT INTO statement and more than one row was returned.
ZERO_DIVIDE	ORA-01476	You tried to divide a number by zero.
INVALID_NUMBER	ORA-01722	You tried to execute a SQL statement that tried to convert a string to a number, but it was unsuccessful.
STORAGE_ERROR	ORA-06500	You ran out of memory or memory was corrupted.
PROGRAM_ERROR	ORA-06501	This is a generic "Contact Oracle support" message because an internal problem was encountered.
VALUE_ERROR	ORA-06502	You tried to perform an operation and there was a error on a conversion, truncation, or invalid constraining of numeric or character data.
CURSOR_ALREADY_OPEN	ORA-06511	You tried to open a cursor that is already open.

https://docs.oracle.com/database/121/TTPLS/exceptions.htm#TTPLS191