Linguaggio SQL: costrutti avanzati

SQL per le applicazioni

SQL per le applicazioni

- Introduzione
- Concetto di cursore
- Aggiornabilità
- SQL statico e dinamico
- Embedded SQL
- Call Level Interface (CLI)
- Stored Procedure
- Confronto tra le alternative

SQL per le applicazioni

Introduzione

Esempio applicativo

- Operazioni bancarie
  - operazione di prelievo dal proprio conto corrente mediante bancomat
  - operazione di prelievo dal proprio conto corrente presso uno sportello bancario

Prelievo mediante bancomat

- Operazioni svolte
  - verificare la validità del bancomat e del codice PIN
  - selezionare l'operazione di prelievo
  - specificare l'importo richiesto
  - verificare la disponibilità
  - memorizzare il movimento
  - aggiornare il saldo
  - erogare la somma richiesta

Prelievo mediante bancomat

- Per svolgere molte delle operazioni indicate è necessario accedere alla base di dati
  - esecuzione di istruzioni SQL
- Le operazioni devono essere svolte nell'ordine corretto

Prelievo presso uno sportello bancario

- Operazioni svolte
  - verificare l'identità dell'utente
  - comunicare l'intenzione di effettuare un prelievo
  - verificare la disponibilità
  - memorizzare il movimento
  - aggiornare il saldo
  - erogare la somma richiesta

Prelievo presso uno sportello bancario

- Per svolgere molte delle operazioni indicate è necessario accedere alla base di dati
  - esecuzione di istruzioni SQL
- Le operazioni devono essere svolte nell'ordine corretto

Esempio: operazioni bancarie

- Le operazioni bancarie richiedono di accedere alla base di dati e di modificarne il contenuto
  - esecuzione di istruzioni SQL
    - i clienti e il personale della banca non eseguono direttamente le istruzioni SQL
  - un'applicazione nasconde l'esecuzione delle istruzioni SQL
- La corretta gestione delle operazioni bancarie richiede di eseguire una sequenza precisa di passi
  - un'applicazione permette di specificare l'ordine corretto di esecuzione delle operazioni

Applicazioni e SQL

- Per risolvere problemi reali non è quasi mai sufficiente eseguire singole istruzioni SQL
- Servono applicazioni per
  - acquisire e gestire i dati forniti in ingresso
    - scelte dell'utente, parametri
  - gestire la logica applicativa
    - flusso di operazioni da eseguire
  - restituire i risultati all'utente in formati diversi
    - rappresentazione non relazionale dei dati
      - documento XML
    - visualizzazione complessa delle informazioni
      - grafici, report

Integrazione tra SQL e applicazioni

- Le applicazioni sono scritte in linguaggi di programmazione tradizionali di alto livello
  - C, C++, Java, C#, ...
  - il linguaggio è denominato *linguaggio ospite*
- Le istruzioni SQL sono usate nelle applicazioni per accedere alla base di dati
  - interrogazioni
  - aggiornamenti

Integrazione tra SQL e applicazioni

- È necessario integrare il linguaggio SQL e i linguaggi di programmazione
  - SQL
    - linguaggio dichiarativo
  - linguaggi di programmazione
    - tipicamente procedurali

Conflitto di impedenza

- Conflitto di impedenza
  - le interrogazioni SQL operano su una o più tabelle e producono come risultato una tabella
    - approccio set oriented
  - i linguaggi di programmazione accedono alle righe di una tabella leggendole *una a una*
    - approccio tuple oriented
- Soluzioni possibili per risolvere il conflitto
  - uso di cursori
  - uso di linguaggi che dispongono in modo naturale di strutture di tipo "insieme di righe"

SQL e linguaggi di programmazione

- Tecniche principali di integrazione
  - Embedded SQL
  - Call Level Interface (CLI)
    - SQL/CLI, ODBC, JDBC, OLE DB, ADO.NET, ..
  - Stored procedure
- Classificabili in
  - client side
    - embedded SQL, call level interface
  - server side
    - stored procedure

Approccio client side

- L'applicazione
  - è esterna al DBMS
  - contiene tutta la logica applicativa
  - richiede al DBMS di eseguire istruzioni SQL e di restituirne il risultato
  - elabora i dati restituiti

Approccio server side

- L'applicazione (o una parte di essa)
  - si trova nel DBMS
  - tutta o parte della logica applicativa si sposta nel DBMS

Approccio client side vs server side

- Approccio client side
  - maggiore indipendenza dal DBMS utilizzato
  - minore efficienza
- Approccio server side
  - dipendente dal DBMS utilizzato
  - maggiore efficienza

SQL per le applicazioni

Concetto di cursore

Conflitto di impedenza

- Principale problema di integrazione tra SQL e linguaggi di programmazione
  - le interrogazioni SQL operano su una o più tabelle e producono come risultato una tabella
    - approccio set oriented
  - i linguaggi di programmazione accedono alle righe di una tabella leggendole *una a una*
    - approccio tuple oriented

Cursori

- Se un'istruzione SQL restituisce una sola riga
  - è sufficiente specificare in quale variabile del linguaggio ospite memorizzare il risultato dell'istruzione
- Se un'istruzione SQL restituisce una tabella (insieme di tuple)
  - è necessario un metodo per leggere (e passare al programma) una tupla alla volta dal risultato dell'interrogazione
    - uso di un *cursore*

DB forniture prodotti

P

CodP	NomeP	Colore	Taglia	Magazzino
P1	Maglia	Rosso	40	Torino
P2	Jeans	Verde	48	Milano
P3	Camicia	Blu	48	Roma
P4	Camicia	Blu	44	Torino
P5	Gonna	Blu	40	Milano
P6	Bermuda	Rosso	42	Torino

FP

CodF	CodP	Qta
F1	P1	300
F1	P2	200
F1	P3	400
F1	P4	200
F1	P5	100
F1	P6	100
F2	P1	300
F2	P2	400
F3	P2	200
F4	P3	200
F4	P4	300
F4	P5	400

F

CodF	NomeF	NSoci	Sede
F1	Andrea	2	Torino
F2	Luca	1	Milano
F3	Antonio	3	Milano
F4	Gabriele	2	Torino
F5	Matteo	3	Venezia

Esempio n.1

- Visualizzare nome e numero di soci del fornitore con codice F1

```
SELECT NomeF, NSoci
FROM   F
WHERE CodF='F1';
```

- L'interrogazione restituisce *al massimo* una tupla

NomeF	NSoci
Andrea	2

- È sufficiente specificare in quali variabili del linguaggio ospite memorizzare la tupla selezionata

Esempio n.2

- Visualizzare nome e numero di soci dei fornitori di Torino

```
SELECT NomeF, NSoci
FROM   F
WHERE Sede='Torino';
```

- L'interrogazione restituisce un insieme di tuple

NomeF	NSoci
Andrea	2
Gabriele	2

Cursore

- È necessario definire un *cursore* per leggere separatamente le tuple del risultato

Esempio n.2

- Definizione del cursone mediante la sintassi del linguaggio PL/SQL di Oracle

```
CURSOR FornitoriTorino IS
SELECT NomeF, NSoci
FROM   F
WHERE Sede='Torino';
```

Cursore

- Il cursore permette di leggere singolarmente le tuple che fanno parte del risultato di un'interrogazione
  - deve essere associato a un'interrogazione specifica
- Ogni interrogazione SQL che può restituire un insieme di tuple *deve essere associata* a un cursore

Cursore

- Non necessitano di cursori
  - le interrogazione SQL che restituiscono al massimo una tupla
    - selezioni sulla chiave primaria
    - operazioni di aggregazione senza clausola GROUP BY
  - i comandi di aggiornamento e di DDL
    - non generano tuple come risultato

SQL per le applicazioni

Aggiornabilità

Aggiornabilità

- È possibile aggiornare o cancellare la tupla corrente puntata dal cursore
  - più efficiente rispetto all'esecuzione di un'istruzione SQL separata di aggiornamento
- L'aggiornamento di una tupla tramite cursore è possibile solo se è aggiornabile la vista che corrisponderebbe all'interrogazione associata
  - deve esistere una corrispondenza uno a uno tra la tupla puntata dal cursore e la tupla da aggiornare nella tabella della base di dati

Esempio: cursore non aggiornabile

- Si supponga l'esistenza del cursore *DatiFornitori* associato all'interrogazione

```
SELECT DISTINCT CodF, NomeF, NSoci
FROM   F, FP, P
WHERE F.CodF=FP.CodF
        AND P.CodP=FP.CodP
        AND Colore='Rosso';
```

- Il cursore DatiFornitori *non* è aggiornabile
- Scrivendo in modo diverso l'interrogazione, il cursore può diventare aggiornabile

Esempio: cursore aggiornabile

- Si supponga di associare al cursore *DatiFornitori* la seguente interrogazione

```
SELECT CodF, NomeF, NSoci
FROM   F
WHERE CodF IN (SELECT CodF
                      FROM   FP, P
                       WHERE FP.CodP=P.CodP
                                AND Colore='Rosso');
```

- Le due interrogazioni sono equivalenti
  - il risultato della nuova interrogazione è identico
- Il cursore DatiFornitori è aggiornabile

SQL per le applicazioni

SQL statico e dinamico

SQL statico

- Le istruzioni SQL da eseguire sono note durante la scrittura dell'applicazione
  - è nota la definizione di ogni istruzione SQL
  - le istruzioni possono contenere variabili
    - il valore delle variabili è noto solo durante l'esecuzione dell'istruzione SQL

SQL statico

- La definizione delle istruzioni SQL avviene durante la scrittura dell'applicazione
  - semplifica la scrittura dell'applicazione
    - è nota a priori la struttura di interrogazioni e risultati
  - rende possibile l'ottimizzazione a priori delle istruzioni SQL
    - durante la fase di compilazione dell'applicazione, l'ottimizzatore del DBMS
      - compila l'istruzione SQL
      - crea il piano di esecuzione
    - queste operazioni non sono più necessarie durante l'esecuzione dell'applicazione
      - esecuzione più efficiente

SQL dinamico

- Le istruzioni SQL da eseguire *non* sono note durante la scrittura dell'applicazione
  - le istruzioni SQL sono definite dinamicamente dall'applicazione in fase di esecuzione
    - dipendono dal flusso applicativo eseguito
  - le istruzioni SQL possono essere fornite in ingresso dall'utente

SQL dinamico

- La definizione a tempo di esecuzione delle istruzioni SQL
  - permette di definire applicazioni più complesse
    - offre una maggiore flessibilità
  - rende più difficile la scrittura delle applicazioni
    - durante la scrittura non è noto il formato del risultato dell'interrogazione
  - rende l'esecuzione meno efficiente
    - durante ogni esecuzione dell'applicazione, è necessario compilare e ottimizzare ogni istruzione SQL

SQL dinamico

- Se la stessa interrogazione dinamica deve essere eseguita più volte nella *stessa sessione* di lavoro
  - è possibile ridurre i tempi di esecuzione
    - si effettua una sola volta la compilazione e la scelta del piano di esecuzione
    - si esegue l'interrogazione più volte (con valori diversi delle variabili)

SQL per le applicazioni

Embedded SQL

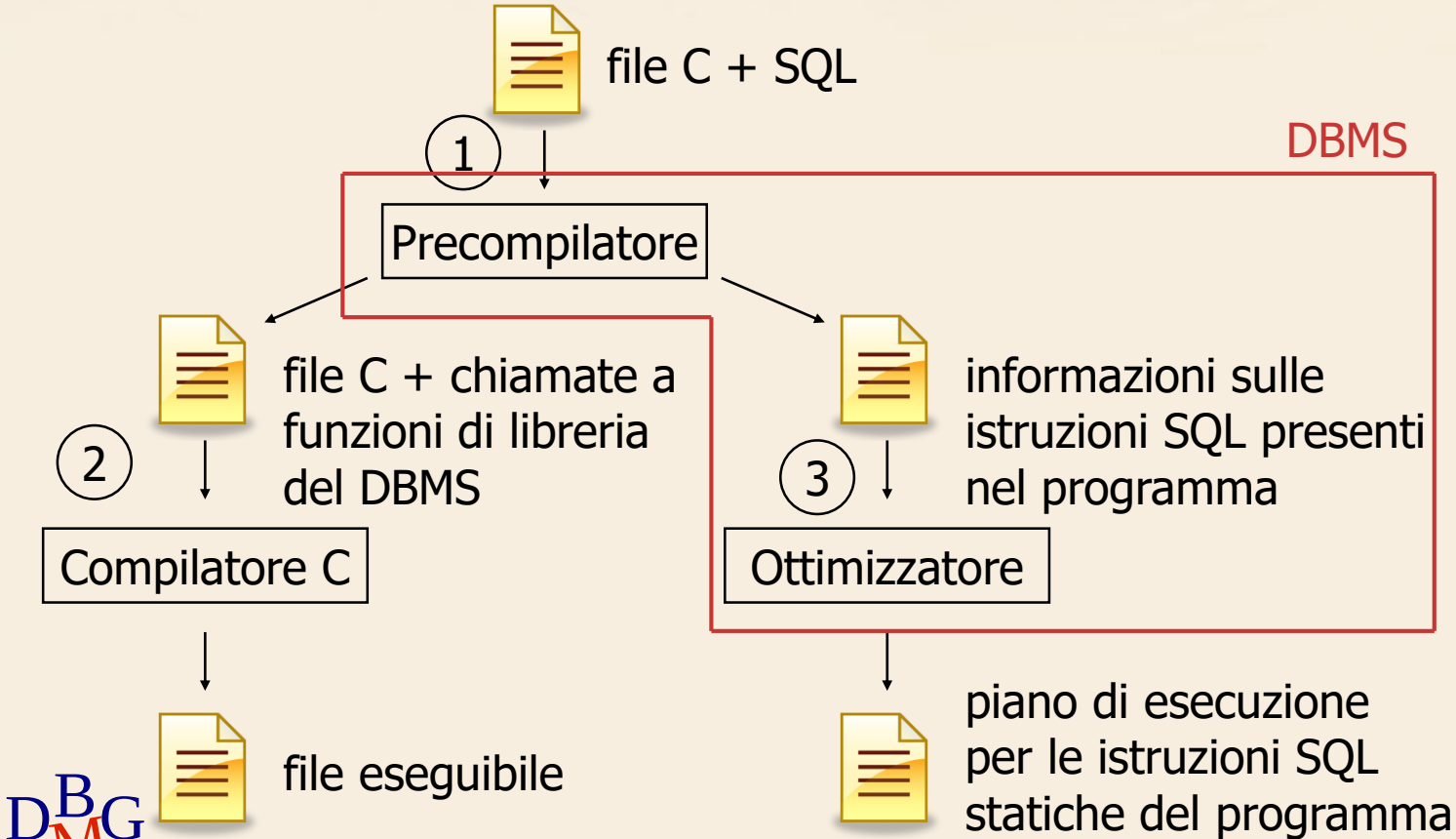
Embedded SQL

- Le istruzioni SQL sono "incorporate" nell'applicazione scritta in un linguaggio di programmazione tradizionale (C, C++, Java, ..)
  - la sintassi SQL è diversa da quella del linguaggio ospite
- Le istruzioni SQL non sono direttamente compilabili da un compilatore tradizionale
  - devono essere riconosciute
    - sono precedute dalla parola chiave EXEC SQL
  - devono essere sostituite da istruzioni nel linguaggio di programmazione ospite

Precompilazione

- Il precompilatore
  - identifica le istruzioni SQL incorporate nel codice
    - parti precedute da EXEC SQL
  - sostituisce le istruzioni SQL con chiamate a funzioni di una API specifica del DBMS prescelto
    - funzioni scritte nel linguaggio di programmazione ospite
  - (opzionale) invia le istruzioni SQL statiche al DBMS che le compila e le ottimizza
- Il precompilatore è legato al DBMS prescelto

Embedded SQL: compilazione



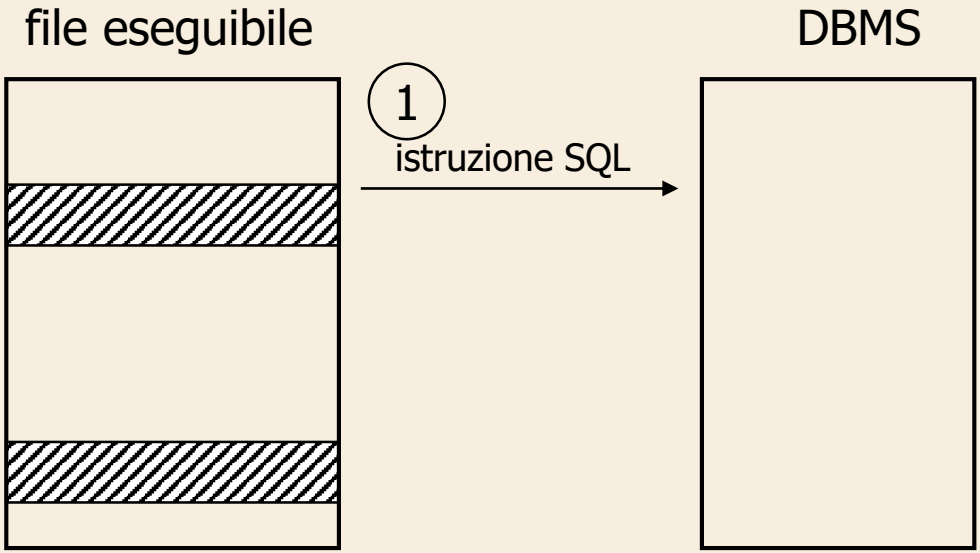
Precompilatore

- Il precompilatore dipende da tre elementi dell'architettura del sistema
  - linguaggio ospite
  - DBMS
  - sistema operativo
- È necessario disporre del precompilatore adatto per l'architettura prescelta

Embedded SQL: esecuzione

- Durante l'esecuzione del programma
  1. Il programma invia l'istruzione SQL al DBMS
     - esegue una chiamata a una funzione di libreria del DBMS

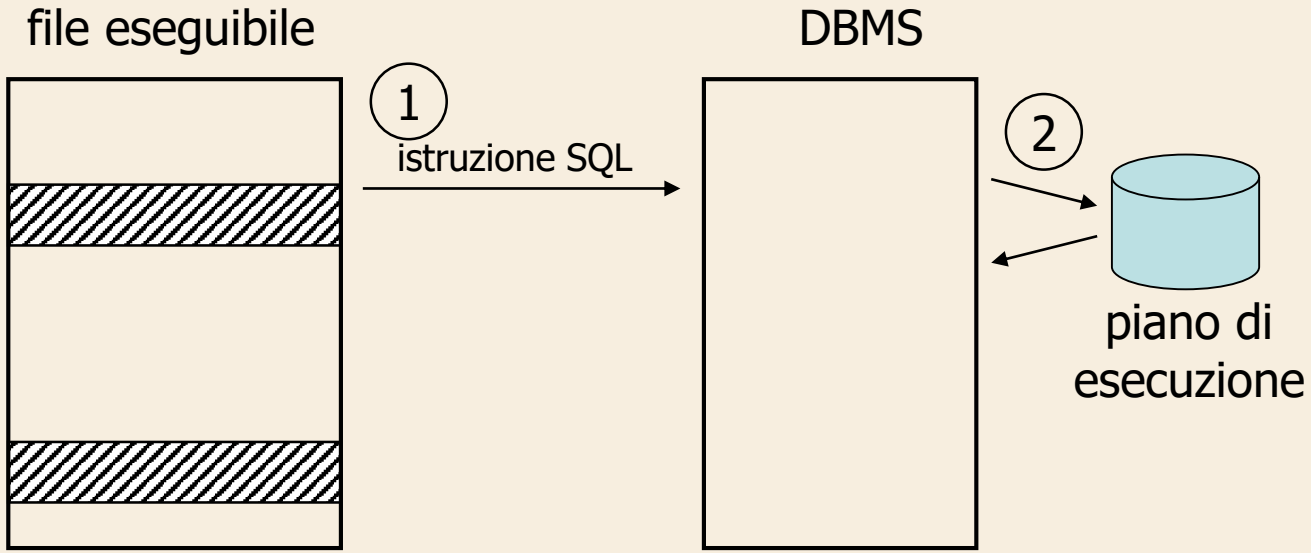
Embedded SQL: esecuzione



Embedded SQL: esecuzione

- Durante l'esecuzione del programma
  1. Il programma invia l'istruzione SQL al DBMS
     - esegue una chiamata a una funzione di libreria del DBMS
  2. Il DBMS genera il piano di esecuzione dell'istruzione
     - se è già stato predefinito deve solo recuperarlo

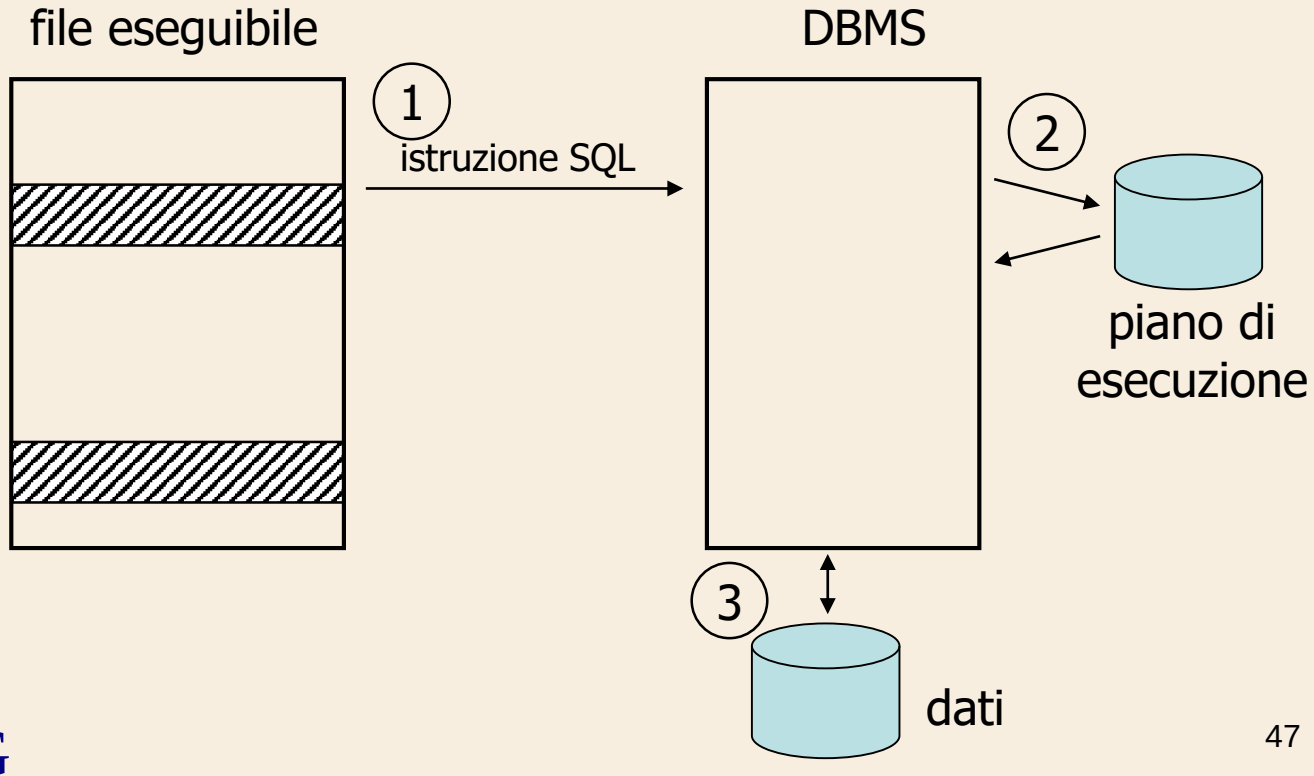
Embedded SQL: esecuzione



Embedded SQL: esecuzione

- Durante l'esecuzione del programma
  1. Il programma invia l'istruzione SQL al DBMS
     - esegue una chiamata a una funzione di libreria del DBMS
  2. Il DBMS genera il piano di esecuzione dell'istruzione
     - se è già stato predefinito deve solo recuperarlo
  3. Il DBMS esegue l' istruzione SQL

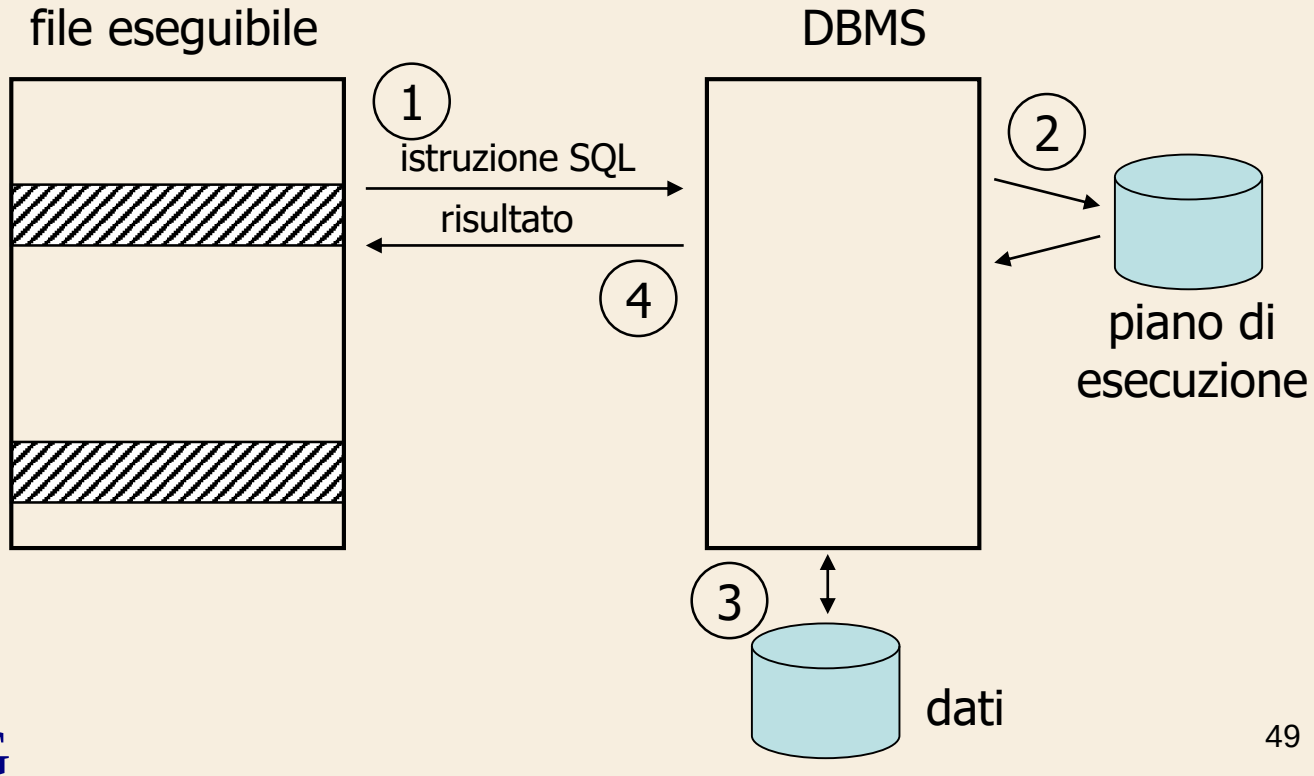
Embedded SQL: esecuzione



Embedded SQL: esecuzione

- Durante l'esecuzione del programma
  1. Il programma invia l'istruzione SQL al DBMS
     - esegue una chiamata a una funzione di libreria del DBMS
  2. Il DBMS genera il piano di esecuzione dell'istruzione
     - se è già stato predefinito deve solo recuperarlo
  3. Il DBMS esegue l' istruzione SQL
  4. Il DBMS restituisce il risultato dell'istruzione SQL
     - utilizza un'area di transito per la memorizzazione temporanea dei dati

Embedded SQL: esecuzione



Embedded SQL: esecuzione

- Durante l'esecuzione del programma
  1. Il programma invia l'istruzione SQL al DBMS
     - esegue una chiamata a una funzione di libreria del DBMS
  2. Il DBMS genera il piano di esecuzione dell'istruzione
     - se è già stato predefinito deve solo recuperarlo
  3. Il DBMS esegue l'istruzione SQL
  4. Il DBMS restituisce il risultato dell'istruzione SQL
     - utilizza un'area di transito per la memorizzazione temporanea dei dati
  5. Il programma elabora il risultato

Esempio di codice embedded SQL

```
#include <stdlib.h>
.....
EXEC SQL BEGIN DECLARE SECTION
char VarCodF[6];
int NumSoci;
char Sede[16];
EXEC SQL END DECLARE SECTION

int alpha, beta;
....
EXEC SQL DECLARE F TABLE (CodF CHAR(5) NOT NULL,
                          NomeF CHAR(20) NOT NULL,
                          NSoci SMALLINT NOT NULL,
                          Sede CHAR(15) NOT NULL);
.....
```

Esempio di codice embedded SQL

```
EXEC SQL INCLUDE SQLCA;
.....
if (alpha>beta) {
  EXEC SQL SELECT NSoci, Sede
                    INTO :NumSoci, :Sede
                    FROM F
                    WHERE CodF=:VarCodF;

  printf("%d %s", NumSoci, Sede);
  ......
}
.....
```

Esempio di codice embedded SQL

```
#include <stdlib.h>
.....
EXEC SQL BEGIN DECLARE SECTION
char VarCodF[6];
int NumSoci;
char Sede[16];
EXEC SQL END DECLARE SECTION

int alpha, beta;
....
EXEC SQL DECLARE F TABLE (CodF CHAR(5) NOT NULL,
                          NomeF CHAR(20) NOT NULL,
                          NSoci SMALLINT NOT NULL,
                          Sede CHAR(15) NOT NULL);
.....
```

Dichiarazione variabili del linguaggio ospite usate nelle istruzioni SQL

Esempio di codice embedded SQL

```
#include <stdlib.h>
.....
EXEC SQL BEGIN DECLARE SECTION
char VarCodF[6];
int NumSoci;
char Sede[16];
EXEC SQL END DECLARE SECTION

int alpha, beta;
....
EXEC SQL DECLARE F TABLE (CodF CHAR(5) NOT NULL,
                          NomeF CHAR(20) NOT NULL,
                          NSoci SMALLINT NOT NULL,
                          Sede CHAR(15) NOT NULL);
.....
```

(Opzionale)
Dichiarazione delle tabelle usate nell'applicazione

Esempio di codice embedded SQL

Dichiarazione dell'area di comunicazione

```
EXEC SQL INCLUDE SQLCA;
.....
if (alpha>beta) {
  EXEC SQL SELECT NSoci, Sede
                    INTO :NumSoci, :Sede
                    FROM F
                    WHERE CodF=:VarCodF;

  printf("%d %s", NumSoci, Sede);
  ......
}
.....
```

Esempio di codice embedded SQL

```
EXEC SQL INCLUDE SQLCA;
.....
if (alpha>beta) {
  EXEC SQL SELECT NSoci, Sede
                  INTO :NumSoci, :Sede
                  FROM F
                  WHERE CodF=:VarCodF;

  printf("%d %s", NumSoci, Sede);
  ......
}
.....
```

Esecuzione di un'istruzione SQL

Esempio di codice embedded SQL

```
EXEC SQL INCLUDE SQLCA;
.....
if (alpha>beta) {
  EXEC SQL SELECT NSoci, Sede
                  INTO :NumSoci, :Sede
                  FROM F
                  WHERE CodF=:VarCodF;

  printf("%d %s", NumSoci, Sede);
  ......
}
.....
```

Variabili del linguaggio ospite

Variabili del linguaggio ospite

- È possibile introdurre nelle istruzioni SQL riferimenti a variabili del linguaggio ospite
  - variabili in lettura
    - permettono l'esecuzione interattiva delle istruzioni
    - le variabili sono usate come parametri nei predicati di selezione al posto delle costanti
  - variabili in scrittura
    - variabili in cui è memorizzata la tupla corrente
    - indicate dopo la parola chiave INTO nelle istruzioni SELECT e FETCH

Variabili del linguaggio ospite

- Nei programmi
  - la dichiarazione delle variabili è delimitata dalla coppia di istruzioni
    - EXEC SQL BEGIN DECLARE SECTION
    - EXEC SQL END DECLARE SECTION
  - nelle istruzioni SQL le variabili sono precedute dal simbolo "**:**" per distinguerle dai nomi delle colonne

Controllo di tipo

- Il tipo delle variabili deve essere compatibile con il tipo delle colonne SQL corrispondenti
  - i nomi delle variabili e delle colonne SQL possono essere uguali

Controllo semantico

- Ogni istruzione SQL DML deve far riferimento a oggetti già definiti nella base di dati
- Il precompilatore effettua il controllo semantico delle istruzioni SQL
  - accedendo alla base di dati per reperire nel dizionario dei dati lo schema degli oggetti referenziati
    - è necessaria la possibilità di connettersi al DBMS durante la precompilazione del codice
  - oppure considerando le definizioni delle tabelle presenti nel codice
    - istruzione EXEC SQL DECLARE

Area di transito

- È necessario definire un'area di comunicazione tra DBMS e linguaggio ospite
  - alcuni precompilatori includono in automatico la definizione dell'area di comunicazione
  - in altri casi è necessario usare l'istruzione

    EXEC SQL INCLUDE SQLCA
- È necessario disporre di variabili apposite per conoscere lo stato dell'ultima istruzione SQL eseguita
  - variabile SQLCA.SQLCODE
    - definita in automatico

Esecuzione di istruzioni SQL

- L'embedded SQL permette di eseguire tutte le tipologie di istruzioni SQL
  - DML
  - DDL
- Esecuzione di un'istruzione SQL

  EXEC SQL *IstruzioneSQL*;

Esecuzione di istruzioni SQL

- Dopo l'esecuzione è possibile verificare lo stato dell'istruzione eseguita mediante la variabile SQLCA.SQLCODE
  - comando eseguito correttamente

    SQLCODE=0
  - comando non eseguito a causa di un errore

    SQLCODE≠0
    - il valore di SQLCODE specifica il tipo di errore

Istruzioni di aggiornamento e DDL

- Istruzioni che non restituiscono un insieme di tuple
  - è necessario verificare se l'operazione è andata a buon fine

    SQLCODE=0
  - non ci sono risultati da analizzare
    - non è necessario l'uso di cursori

Interrogazioni

- Si opera in modo diverso in funzione del numero di tuple restituite dall'interrogazione
  - una sola tupla
    - esecuzione dell'istruzione SELECT
    - indicazione delle variabili in cui memorizzare il risultato direttamente nell'istruzione SELECT
      - non è necessario l'uso di cursori
  - un insieme di tuple
    - definizione e uso di un *cursore* associato all'istruzione SELECT
    - indicazione delle variabili in cui memorizzare le singole tuple lette nell'istruzione FETCH

Esempio: selezione di una sola tupla

- Selezionare il numero di soci e la sede del fornitore il cui valore del codice è contenuto nella variabile ospite *VarCodF*

```
EXEC SQL SELECT NSoci, Sede INTO :NumSoci, :Sede
         FROM F
         WHERE CodF = :VarCodF;
```

- Nell'interrogazione SQL si indicano dopo la parola chiave INTO le variabili in cui memorizzare il risultato

Esempio: selezione di una sola tupla

- Selezionare il numero di soci e la sede del fornitore il cui valore del codice è contenuto nella variabile ospite *VarCodF*

```
EXEC SQL SELECT NSoci, Sede INTO :NumSoci, :Sede
         FROM F
         WHERE CodF = :VarCodF;
```

- Nella clausola WHERE è possibile utilizzare variabili del linguaggio ospite al posto di costanti

Esempio: selezione di una sola tupla

- Occorre verificare sempre lo stato dell'operazione dopo l'esecuzione
  - SQLCODE = 0
    - interrogazione eseguita correttamente
    - i valori selezionati sono stati memorizzati nelle variabili indicate nell'interrogazione (NumSoci e Sede)
  - SQLCODE = 100
    - nessuna tupla soddisfa il predicato
  - SQLCODE < 0
    - errore di esecuzione
      - più record soddisfano il predicato
      - tabella non disponibile
      - ...

Cursore

- Il cursore permette di leggere singolarmente le tuple che fanno parte del risultato di un'interrogazione
  - deve essere associato a un'interrogazione specifica
- Ogni interrogazione SQL che può restituire un insieme di tuple *deve essere associata* a un cursore

Operazioni sui cursori

- Operazioni di base sui cursori
  - dichiarazione
  - apertura
  - lettura (tipicamente all'interno di un ciclo)
  - chiusura
- Simili alle modalità di gestione di un file

Dichiarazione

- Istruzione DECLARE
  - dichiarazione della struttura del cursore
    - assegnazione di un nome al cursore
    - definizione dell'interrogazione associata al cursore

Istruzione DECLARE

EXEC SQL DECLARE *NomeCursore* [SCROLL] CURSOR
FOR *InterrogazioneSQL*
[FOR <READ ONLY| UPDATE [OF *ElencoAttributi*]>];

- Opzione READ ONLY
  - il cursore può essere usato solo per leggere i risultati
  - opzione di default

Istruzione DECLARE

EXEC SQL DECLARE *NomeCursore* [SCROLL] CURSOR
FOR *InterrogazioneSQL*
[FOR <READ ONLY| UPDATE [OF *ElencoAttributi*]>];

- Opzione SCROLL
  - l'applicazione può muoversi liberamente sul risultato
    - lettura in avanti e indietro

Istruzione DECLARE

EXEC SQL DECLARE *NomeCursore* [SCROLL] CURSOR
FOR *InterrogazioneSQL*
[FOR <READ ONLY| UPDATE [OF *ElencoAttributi*]>];

- Opzione UPDATE
  - il cursore può essere usato in un'istruzione di aggiornamento
    - è possibile specificare quali attributi saranno aggiornati

Apertura

- Istruzione OPEN
  - apertura del cursore
    - esecuzione dell'interrogazione sulla base di dati
    - memorizzazione del risultato in un'area temporanea

```
EXEC SQL OPEN NomeCursore;
```

- Dopo l'apertura il cursore si trova prima della prima tupla del risultato

Lettura

- Istruzione FETCH
  - lettura della prossima tupla disponibile
    - memorizzazione della tupla in una variabile del programma ospite
  - aggiornamento della posizione del cursore
    - spostamento del cursore in avanti di una riga
      - spostamento alla tupla successiva
- È necessario definire un ciclo per leggere tutte le tuple del risultato
  - si utilizza il linguaggio ospite
  - ogni chiamata dell'istruzione FETCH all'interno del ciclo seleziona una sola tupla

Istruzione FETCH

EXEC SQL FETCH [*Posizione* FROM] *NomeCursore*
INTO *ElencoVariabili*;

- Se nella definizione del cursore è presente l'opzione SCROLL il parametro *Posizione* può assumere i valori
  - next, prior, first, last, absolute, relative
- Altrimenti può assumere solo il valore next
  - valore di default

Posizione del cursore (1/2)

- Valori della *Posizione*
  - next
    - lettura della riga successiva alla riga corrente
  - prior
    - lettura della riga precedente alla riga corrente
  - first
    - lettura della prima riga del risultato
  - last
    - lettura dell'ultima riga del risultato

Posizione del cursore (2/2)

- absolute *espressioneIntera*
  - lettura della riga *i*-esima del risultato
    - *la posizione i* è il risultato dell'espressione intera
- relative *espressioneIntera*
  - come absolute ma il punto di riferimento è la posizione corrente

Chiusura

- Istruzione CLOSE
  - chiusura del cursore
    - liberazione dell'area temporanea contenente il risultato dell'interrogazione
      - il risultato dell'interrogazione non è più accessibile
    - aggiornamento della base di dati nel caso di cursori associati a interrogazioni aggiornabili

EXEC SQL CLOSE *NomeCursore*;

Osservazioni

- All'interno di un'applicazione, un cursore
  - è definito una sola volta
  - può essere usato più volte
    - deve essere aperto e chiuso ogni volta
- Si possono definire più cursori nella stessa applicazione

Esempio: selezione fornitori

- Presentare a video il codice e il numero di soci dei fornitori la cui sede è contenuta nella variabile ospite *VarSede*
  - il valore di *VarSede* è fornito dall'utente come parametro dell'applicazione

Esempio: selezione fornitori

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/************************** Gestione errori ***********************************/
void sql_error(char *msg)
{
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    fprintf(stderr,"\n%s\n", msg);
    fprintf(stderr,"codice interno errore: %ld\n", sqlca.sqlcode);
    fprintf(stderr,"%s\n",sqlca.sqlerrm.sqlerrmc);
    EXEC SQL ROLLBACK;
    exit(EXIT_FAILURE);
}
```

Esempio: selezione fornitori

```
/*********************************  MAIN  ***********************************/
int main(int argc,char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
        char username[20]="bdati";
        char password[20]="passbdati";
        char VarSede[16];
        char CodF[6];
        int  NSoci;
    EXEC SQL END DECLARE SECTION;

     /* Gestione diretta degli errori */
    EXEC SQL WHENEVER SQLERROR CONTINUE;

    /* Apertura connessione */
    EXEC SQL CONNECT TO forniture@127.0.0.1 USER :username IDENTIFIED BY :password;

    if (sqlca.sqlcode!=0)
        sql_error("Errore in fase di connessione");
```

Esempio: selezione fornitori

```
/* Dichiarazione cursore */
EXEC SQL DECLARE fornitoriSelezionati CURSOR FOR
SELECT CodF,NSoci FROM F WHERE Sede = :VarSede;

/* Impostazione valore VarSede */
strcpy(VarSede,argv[1]);

/* Apertura del cursore */
EXEC SQL OPEN fornitoriSelezionati;

if (sqlca.sqlcode!=0)
    sql_error("Errore in fase di apertura cursore");

/* Stampa dei dati selezionati */
printf("Elenco fornitori\n");
```

Esempio: selezione fornitori

```
do {
        EXEC SQL FETCH fornitoriSelezionati INTO :CodF, :NSoci;
        /* Verifica stato ultima operazione di fetch */
        switch(sqlca.sqlcode)   {
                case 0: /* Letta correttamente una nuova tupla */
                { /* Stampa a video della tupla  */
                    printf("%s,%d",CodF, NSoci);
                }
                break;

                case 100: /* Sono finiti i dati */
                break;

                default: /* Si e' verificato un errore */
                sql_error("Errore in fase di lettura dei dati");
                break;
        }
}
while (sqlca.sqlcode==0);
```

Esempio: selezione fornitori

```
    /* Chiusura cursore */
    EXEC SQL CLOSE fornitoriSelezionati;
}
```

Esempio: selezione fornitori

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/************************** Gestione errori ***********************************/
void sql_error(char *msg)
{
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    fprintf(stderr,"\n%s\n", msg);
    fprintf(stderr,"codice interno errore: %ld\n", sqlca.sqlcode);
    fprintf(stderr,"%s\n",sqlca.sqlerrm.sqlerrmc);
    EXEC SQL ROLLBACK;
    exit(EXIT_FAILURE);
}
```

Gestione errori

Esempio: selezione fornitori

```
/********************************  MAIN  ***********************************/
int main(int argc,char **argv)
{
  EXEC SQL BEGIN DECLARE SECTION;
     char username[20]="bdati";
     char password[20]="passbdati";
     char VarSede[16];
     char CodF[6];
     int  NSoci;
  EXEC SQL END DECLARE SECTION;

   /* Gestione diretta degli errori */
  EXEC SQL WHENEVER SQLERROR CONTINUE;

  /* Apertura connessione */
  EXEC SQL CONNECT TO forniture@127.0.0.1 USER :username IDENTIFIED BY :password;

  if (sqlca.sqlcode!=0)
     sql_error("Errore in fase di connessione");
```

Definizione variabili

Esempio: selezione fornitori

```
/*********************************  MAIN  ************************************/
int main(int argc,char **argv)
{
   EXEC SQL BEGIN DECLARE SECTION;
      char username[20]="bdati";
      char password[20]="passbdati";
      char VarSede[16];
      char CodF[6];
      int  NSoci;
   EXEC SQL END DECLARE SECTION;

    /* Gestione diretta degli errori */
   EXEC SQL WHENEVER SQLERROR CONTINUE;

   /* Apertura connessione */
   EXEC SQL CONNECT TO forniture@127.0.0.1 USER :username IDENTIFIED BY :password;

   if (sqlca.sqlcode!=0)
      sql_error("Errore in fase di connessione");
```

Connessione con il DBMS

Esempio: selezione fornitori

```
/* Dichiarazione cursore */
EXEC SQL DECLARE fornitoriSelezionati CURSOR FOR
SELECT CodF,NSoci FROM F WHERE Sede = :VarSede;

/* Impostazione valore VarSede */
strcpy(VarSede,argv[1]);

/* Apertura del cursore */
EXEC SQL OPEN fornitoriSelezionati;

if (sqlca.sqlcode!=0)
    sql_error("Errore in fase di apertura cursore");

/* Stampa dei dati selezionati */
printf("Elenco fornitori\n");
```

Definizione cursore

Esempio: selezione fornitori

```
/* Dichiarazione cursore */
EXEC SQL DECLARE fornitoriSelezionati CURSOR FOR
SELECT CodF,NSoci FROM F WHERE Sede = :VarSede;

/* Impostazione valore VarSede */
strcpy(VarSede,argv[1]);

/* Apertura del cursore */
EXEC SQL OPEN fornitoriSelezionati;

if (sqlca.sqlcode!=0)
    sql_error("Errore in fase di apertura cursore");

/* Stampa dei dati selezionati */
printf("Elenco fornitori\n");
```

Apertura cursore

Esempio: selezione fornitori

```
do {
        EXEC SQL FETCH fornitoriSelezionati INTO :CodF, :NSoci;
        /* Verifica stato ultima operazione di fetch */
        switch(sqlca.sqlcode)   {
                case 0: /* Letta correttamente una nuova tupla */
                { /* Stampa a video della tupla  */
                    printf("%s,%d",CodF, NSoci);
                }
                break;

                case 100: /* Sono finiti i dati */
                break;

                default: /* Si e' verificato un errore */
                sql_error("Errore in fase di lettura dei dati");
                break;
        }
}
while (sqlca.sqlcode==0);
```

Ciclo di lettura delle tuple

Esempio: selezione fornitori

```
do {
        EXEC SQL FETCH fornitoriSelezionati INTO :CodF, :NSoci;
        /* Verifica stato ultima operazione di fetch */
        switch(sqlca.sqlcode)   {
                case 0: /* Letta correttamente una nuova tupla */
                { /* Stampa a video della tupla  */
                    printf("%s,%d",CodF, NSoci);
                }
                break;

                case 100: /* Sono finiti i dati */
                break;

                default: /* Si e' verificato un errore */
                sql_error("Errore in fase di lettura dei dati");
                break;
        }
}
while (sqlca.sqlcode==0);
```

Lettura di una tupla

Esempio: selezione fornitori

```
do {
        EXEC SQL FETCH fornitoriSelezionati INTO :CodF, :NSoci;
        /* Verifica stato ultima operazione di fetch */
        switch(sqlca.sqlcode)   {
                case 0: /* Letta correttamente una nuova tupla */
                { /* Stampa a video della tupla  */
                    printf("%s,%d",CodF, NSoci);
                }
                break;

                case 100: /* Sono finiti i dati */
                break;

                default: /* Si e' verificato un errore */
                sql_error("Errore in fase di lettura dei dati");
                break;
        }
}
while (sqlca.sqlcode==0);
```

Analisi dell'esito della lettura

Esempio: selezione fornitori

```
    /* Chiusura cursore */
    EXEC SQL CLOSE fornitoriSelezionati;
}
```

Chiusura cursore

Aggiornamento mediante cursori

- È possibile aggiornare o cancellare la tupla puntata da un cursore
  - aggiornamento

```
EXEC SQL UPDATE NomeTabella
         SET NomeColonna = Espressione
            {, NomeColonna = Espressione}
         WHERE CURRENT OF NomeCursore;
```

  - cancellazione

```
EXEC SQL DELETE FROM NomeTabella
         WHERE CURRENT OF NomeCursore;
```

Aggiornamento mediante cursori

- L'aggiornamento e la cancellazione sono possibili se e solo se
  - il cursore è stato definito in modo appropriato
    - opzione FOR UPDATE nell'istruzione DECLARE
  - esiste una corrispondenza uno a uno tra le tuple del risultato e le tuple presenti nel DBMS

Gestione delle transazioni

- In un programma embededd SQL è possibile definire i limiti di una transazione
  - inizio di una transazione

    EXEC SQL BEGIN TRANSACTION;
  - termine di una transazione con successo

    EXEC SQL COMMIT;
  - fallimento di una transazione

    EXEC SQL ROLLBACK;

Gestione delle transazioni

- Fino a quando non si invocano in modo esplicito le istruzioni
  - COMMIT o ROLLBACK
- Le operazioni SQL di aggiornamento devono essere considerate "tentativi di aggiornamento"

SQL per le applicazioni

Call Level Interface (CLI)

Call Level Interface

- Le richieste sono inviate al DBMS per mezzo di funzioni del linguaggio ospite
  - soluzione basata su interfacce predefinite
    - API, Application Programming Interface
  - le istruzioni SQL sono passate come parametri alle funzioni del linguaggio ospite
  - non esiste il concetto di precompilatore
- Il programma ospite contiene direttamente le chiamate alle funzioni messe a disposizione dall'API

Call Level Interface

- Esistono diverse soluzioni di tipo Call Level Interface (CLI)
  - standard SQL/CLI
  - ODBC (Open DataBase Connectivity)
    - soluzione proprietaria Microsoft di SQL/CLI
  - JDBC (Java Database Connectivity)
    - soluzione per il mondo Java
  - OLE DB
  - ADO
  - ADO.NET

Modalità d'uso

- Indipendentemente dalla soluzione CLI adottata, esiste una strutturazione comune dell'interazione con il DBMS
  - apertura della connessione con il DBMS
  - esecuzione di istruzioni SQL
  - chiusura della connessione

Interazione con il DBMS

1. Chiamata a una primitiva delle API per creare una connessione con il DBMS
2. Invio sulla connessione di un'istruzione SQL
3. Ricezione di un risultato in risposta all'istruzione inviata
   - nel caso di SELECT, di un insieme di tuple
4. Elaborazione del risultato ottenuto
   - esistono apposite primitive per leggere il risultato
5. Chiusura della connessione al termine della sessione di lavoro

JDBC (Java Database Connectivity)

- Soluzione CLI per il mondo JAVA
- L'architettura prevede
  - un insieme di classi e interfacce standard
    - utilizzate dal programmatore Java
    - indipendenti dal DBMS
  - un insieme di classi "proprietarie" (driver)
    - implementano le interfacce e le classi standard per fornire la comunicazione con un DBMS specifico
    - dipendono dal DBMS utilizzato
    - sono invocate a runtime
      - in fase di compilazione dell'applicazione non sono necessarie

JDBC: interazione con il DBMS

- Caricamento del driver specifico per il DBMS utilizzato
- Creazione di una connessione
- Esecuzione delle istruzioni SQL
  - creazione di uno statement
  - richiesta di esecuzione dell'istruzione
  - elaborazione del risultato nel caso di interrogazioni
- Chiusura dello statement
- Chiusura della connessione

Caricamento del DBMS driver

- Il driver è specifico per il DBMS utilizzato
- Il caricamento avviene tramite l'istanziazione dinamica della classe associata al driver

  Object Class.forName(String nomeDriver)

  - nomeDriver contiene il nome della classe da istanziare
    - esempio: "oracle.jdbc.driver.OracleDriver"

Caricamento del DBMS driver

- È la prima operazione da effettuare
- Non è necessario conoscere in fase di compilazione del codice quale DBMS sarà usato
  - la lettura del nome del driver può avvenire a runtime da un file di configurazione

Creazione di una connessione

- Invocazione del metodo getConnection della classe DriverManager

  Connection DriverManager.getConnection(String url, String user, String password)

  - url
    - contiene l'informazione necessaria per identificare il DBMS a cui ci si vuole collegare
    - formato legato al driver utilizzato
  - user e password
    - credenziali di autenticazione

Esecuzione di istruzioni SQL

- L'esecuzione di un'istruzione SQL richiede l'uso di un'interfaccia specifica
  - denominata Statement
- Ogni oggetto Statement
  - è associato a una connessione
  - è creato tramite il metodo createStatement della classe Connection

```
Statement createStatement()
```

Istruzioni di aggiornamento e DDL

- L'esecuzione dell'istruzione richiede l'invocazione su un oggetto Statement del metodo

  int executeUpdate(String istruzioneSQL)

  - istruzioneSQL
    - è l'istruzione SQL da eseguire
  - il metodo restituisce
    - il numero di tuple elaborate (inserite, modificate, cancellate)
    - il valore 0 per i comandi DDL

Interrogazioni

- Esecuzione immediata dell'interrogazione
  - il server compila ed esegue immediatamente l'istruzione SQL ricevuta
- Esecuzione "preparata" dell'interrogazione
  - utile quando si deve eseguire la stessa istruzione SQL più volte nella stessa sessione di lavoro
    - varia solo il valore di alcuni parametri
  - l'istruzione SQL
    - è compilata (preparata) una volta sola e il suo piano di esecuzione è memorizzato dal DBMS
    - è eseguita molte volte durante la sessione

Esecuzione immediata

- È richiesta dall'invocazione su un oggetto Statement del seguente metodo

  ResultSet executeQuery(String istruzioneSQL)

  - istruzioneSQL
    - è l'interrogazione SQL da eseguire
  - il metodo restituisce sempre una collezione di tuple
    - oggetto di tipo ResultSet
  - gestione uguale per interrogazioni che
    - restituiscono al massimo una tupla
    - possono restituire più tuple

Lettura del risultato

- L'oggetto ResultSet è analogo a un cursore
  - dispone di metodi per
    - spostarsi sulle righe del risultato
      - next()
      - first()
      - ...
    - estrarre i valori di interesse dalla tupla corrente
      - getInt(String nomeAttributo)
      - getString(String nomeAttributo)
      - ....

Statement preparato

- L'istruzione SQL "preparata" è
  - compilata una sola volta
    - all'inizio dell'esecuzione dell'applicazione
  - eseguita più volte
    - prima di ogni esecuzione è necessario specificare il valore corrente dei parametri
- Modalità utile quando è necessario ripetere più volte l'esecuzione della stessa istruzione SQL
  - permette di ridurre il tempo di esecuzione
    - la compilazione è effettuata una volta sola

Preparazione dello Statement

- Si utilizza un oggetto di tipo PreparedStatement
  - creato con il metodo

    PreparedStatement preparStatement(String istruzioneSQL)
    - istruzioneSQL
      - contiene il comando SQL da eseguire
      - dove si vuole specificare la presenza di un parametro è presente il simbolo "?"
- Esempio

```
PreparedStatement pstmt;
pstmt=conn.preparStatement("SELECT CodF, NSoci
   FROM F WHERE Sede=?");
```

Impostazione dei parametri

- Sostituzione dei simboli ? per l'esecuzione corrente
- Si evoca su un oggetto PreparedStatement uno dei seguenti metodi
  - void setInt(int numeroParametro, int valore)
  - void setString(int numeroParametro, String valore)
  - ...
    - numeroParametro indica la posizione del parametro da assegnare
      - possono essere presenti più parametri nella stessa istruzione SQL
      - il primo parametro è associato al numero 1
    - valore indica il valore da assegnare al parametro

Esecuzione dell'istruzione preparata

- Si invoca su un oggetto PreparedStatement il metodo appropriato
  - interrogazione SQL

    ResultSet executeQuery()
  - aggiornamento

    ResultSet executeUpdate()
- I due metodi non hanno nessun parametro di ingresso
  - sono già stati definiti in precedenza
    - l'istruzione SQL da eseguire
    - i suoi parametri di esecuzione

Esempio: statement preparati

```
.....
PreparedStatement pstmt=conn.prepareStatement("UPDATE P
    SET Colore=? WHERE CodP=?");

/* Assegnazione del colore RossoElettrico al prodotto P1 */
pstmt.setString(1,"RossoElettrico");
pstmt.setString(2,"P1");
pstmt.executeUpdate();

/* Assegnazione del colore BluNotte al prodotto P5 */
pstmt.setString(1,"BluNotte");
pstmt.setString(2,"P5");
pstmt.executeUpdate();
```

Esempio: statement preparati

```
.....
PreparedStatement pstmt=conn.prepareStatement("UPDATE P
    SET Colore=? WHERE CodP=?");

/* Assegnazione del colore RossoElettrico al prodotto P1 */
pstmt.setString(1,"RossoElettrico");
pstmt.setString(2,"P1");
pstmt.executeUpdate();

/* Assegnazione del colore BluNotte al prodotto P5 */
pstmt.setString(1,"BluNotte");
pstmt.setString(2,"P5");
pstmt.executeUpdate();
```

Chiusura di statement e connessione

- Quando uno statement o una connessione non servono più
  - devono essere immediatamente chiusi
- Sono rilasciate le risorse
  - dell'applicazione
  - del DBMS

  che non sono più utilizzate

Chiusura di uno statement

- La chiusura di uno statement
  - è eseguita invocando il metodo close sull'oggetto Statement
    - void close()
- Sono rilasciate le risorse associate all'istruzione SQL corrispondente

Chiusura di una connessione

- La chiusura di una connessione
  - deve essere eseguita quando non è più necessario interagire con il DBMS
  - chiude il collegamento con il DBMS e rilascia le relative risorse
    - chiude anche gli statement associati alla connessione
  - è eseguita invocando il metodo close sull'oggetto Connection
    - void close()

Gestione delle eccezioni

- Gli errori sono gestiti mediante eccezioni di tipo SQLException
- L'eccezione SQLException contiene
  - una stringa che descrive l'errore
  - una stringa che identifica l'eccezione
    - in modo conforme a Open Group SQL Specification
  - un codice d'errore specifico per il DBMS utilizzato

Esempio: selezione fornitori

- Presentare a video il codice e il numero di soci dei fornitori la cui sede è contenuta nella variabile ospite *VarSede*
  - il valore di *VarSede* è fornito come parametro dell'applicazione dall'utente

Esempio: selezione fornitori

```
import java.io.*;
import java.sql.*;

class FornitoriSede {

  static public void main(String argv[])  {
      Connection conn;
      Statement stmt;
      ResultSet rs;
      String query;
      String VarSede;

      /* Registrazione driver */
      try  {
          Class.forName("oracle.jdbc.driver.OracleDriver");
      }
      catch(Exception e)  {
          System.err.println("Driver non disponibile: "+e);
      }
```

Esempio: selezione fornitori

```
try {
    /* Connessione alla base di dati */
   conn=DriverManager.getConnection("jdbc:oracle:thin:@127.0.0.1:1521:xe",
            “bdati”,“passbdati");

   /* Creazione statement per comandi immediati */
   stmt = conn.createStatement();

   /* Composizione interrogazione */
   VarSede =argv[0];
   query="SELECT CodF, NSoci FROM F WHERE Sede = '"+VarSede+"'";

   /* Esecuzione interrogazione */
   rs=stmt.executeQuery(query);
```

Esempio: selezione fornitori

```
        System.out.println("Elenco fornitori di "+VarSede);
        /* Analisi tuple restituite */
        while (rs.next()) {
           /* Stampa a video della tupla corrente */
            System.out.println(rs.getString("CodF")+","+rs.getInt("NSoci"));
        }
        /* Chiusura resultset,  statement e connessione */
        rs.close();
        stmt.close();
        conn.close();
    }
    catch(Exception e) {
         System.err.println("Errore: "+e);
    }
  }
}
```

Esempio: selezione fornitori

```
import java.io.*;
import java.sql.*;

class FornitoriSede {

  static public void main(String argv[])  {
      Connection conn;
      Statement stmt;
      ResultSet rs;
      String query;
      String VarSede;

      /* Registrazione driver */
      try  {
          Class.forName("oracle.jdbc.driver.OracleDriver");
      }
      catch(Exception e)  {
          System.err.println("Driver non disponibile: "+e);
      }
```

Caricamento driver

Esempio: selezione fornitori

```
try {
    /* Connessione alla base di dati */
    conn=DriverManager.getConnection("jdbc:oracle:thin:@127.0.0.1:1521:xe",
            "bdati","passbdati");

    /* Creazione statement per comandi immediati */
    stmt = conn.createStatement();

    /* Composizione interrogazione */
    VarSede =argv[0];
    query="SELECT CodF, NSoci FROM F WHERE Sede = '"+VarSede+"'";

    /* Esecuzione interrogazione */
    rs=stmt.executeQuery(query);
```

Connessione al DBMS

Esempio: selezione fornitori

```
try {
    /* Connessione alla base di dati */
   conn=DriverManager.getConnection("jdbc:oracle:thin:@127.0.0.1:1521:xe",
            "bdati","passbdati");

   /* Creazione statement per comandi immediati */
   stmt = conn.createStatement();

   /* Composizione interrogazione */
   VarSede =argv[0];
   query="SELECT CodF, NSoci FROM F WHERE Sede = '"+VarSede+"'";

   /* Esecuzione interrogazione */
   rs=stmt.executeQuery(query);
```

Creazione statement

Esempio: selezione fornitori

```
try {
    /* Connessione alla base di dati */
   conn=DriverManager.getConnection("jdbc:oracle:thin:@127.0.0.1:1521:xe",
            "bdati","passbdati");

   /* Creazione statement per comandi immediati */
   stmt = conn.createStatement();

   /* Composizione interrogazione */
   VarSede =argv[0];
   query="SELECT CodF, NSoci FROM F WHERE Sede = '"+VarSede+"'";

   /* Esecuzione interrogazione */
   rs=stmt.executeQuery(query);
```

Composizione interrogazione SQL

Esempio: selezione fornitori

```
try {
    /* Connessione alla base di dati */
   conn=DriverManager.getConnection("jdbc:oracle:thin:@127.0.0.1:1521:xe",
            "bdati","passbdati");

   /* Creazione statement per comandi immediati */
   stmt = conn.createStatement();

   /* Composizione interrogazione */
   VarSede =argv[0];
   query="SELECT CodF, NSoci FROM F WHERE Sede = '"+VarSede+"'";

   /* Esecuzione interrogazione */
   rs=stmt.executeQuery(query);
```

Esecuzione immediata dell'interrogazione

Esempio: selezione fornitori

```
        System.out.println("Elenco fornitori di "+VarSede);
        /* Analisi tuple restituite */
        while (rs.next()) {
           /* Stampa a video della tupla corrente */
            System.out.println(rs.getString("CodF")+","+rs.getInt("NSoci"));
        }
        /* Chiusura resultset,  statement e connessione */
        rs.close();
        stmt.close();
        conn.close();
    }
    catch(Exception e) {
        System.err.println("Errore: "+e);
    }
  }
}
```

Ciclo di lettura delle tuple

Esempio: selezione fornitori

```
        System.out.println("Elenco fornitori di "+VarSede);
        /* Analisi tuple restituite */
        while (rs.next()) {
           /* Stampa a video della tupla corrente */
            System.out.println(rs.getString("CodF")+","+rs.getInt("NSoci"));
        }
        /* Chiusura resultset,  statement e connessione */
        rs.close();
        stmt.close();
        conn.close();
      }
      catch(Exception e) {
          System.err.println("Errore: "+e);
      }
   }
}
```

Chiusura resultset,
statement e connessione

ResultSet aggiornabile

- È possibile creare un ResultSet di tipo aggiornabile
  - l'esecuzione di aggiornamenti della base di dati è più efficiente
  - è simile a un cursore aggiornabile
    - è necessario che esista una corrispondenza uno a uno tra tuple del risultato e tuple delle tabelle presenti nel DBMS

Definizione di transazione

- Le connessioni avvengono implicitamente in modalità *auto-commit mode*
  - dopo l'esecuzione con successo di ogni istruzione SQL, è eseguito automaticamente commit

Definizione di transazione

- Le connessioni avvengono implicitamente in modalità *auto-commit mode*
  - dopo l'esecuzione con successo di ogni istruzione SQL, è eseguito automaticamente commit
- Quando è necessario eseguire commit solo dopo aver eseguito con successo una *sequenza* di istruzioni SQL
  - si esegue *un solo* commit alla fine dell'esecuzione di tutte le istruzioni
  - il commit deve essere gestito in modo *non automatico*

Gestione delle transazioni

- Gestione della modalità di commit invocando il metodo setAutoCommit() sulla connessione

  void setAutoCommit(boolean autoCommit);

  - parametro autoCommit
    - true se si vuole abilitare l'autocommit (default)
    - false se si vuole disabilitare l'autocommit

Gestione delle transazioni

- Se si disabilita l'autocommit
  - le operazioni di commit e rollback devono essere richieste *esplicitamente*
    - commit

      ```
      void commit();
      ```
    - rollback

      ```
      void rollback();
      ```
  - i metodi sono invocati sulla connessione interessata

SQL per le applicazioni

Stored Procedure

Stored procedure

- La stored procedure è una funzione o una procedura definita all'interno del DBMS
  - è memorizzata nel dizionario dati
    - fa parte dello schema della base di dati
- È utilizzabile come se fosse un'istruzione SQL predefinita
  - può avere parametri di esecuzione
- Contiene codice applicativo e istruzioni SQL
  - il codice applicativo e le istruzioni SQL sono fortemente integrati tra loro

Stored procedure: linguaggio

- Il linguaggio utilizzato per definire una stored procedure
  - è un'estensione procedurale del linguaggio SQL
  - è dipendente dal DBMS
    - prodotti diversi offrono linguaggi diversi
    - l'espressività del linguaggio dipende dal prodotto prescelto

Stored procedure: esecuzione

- Le stored procedure sono integrate nel DBMS
  - approccio server side
- Le prestazioni sono migliori rispetto a embedded SQL e CLI
  - ogni stored procedure è compilata e ottimizzata *una sola volta*
    - subito dopo la definizione
    - oppure la prima volta che è invocata

Linguaggi per le stored procedure

- Esistono diversi linguaggi per definire stored procedure
  - PL/SQL
    - Oracle
  - SQL/PL
    - DB2
  - Transact-SQL
    - Microsoft SQL Server
  - PL/pgSQL
    - PostgreSQL

Connessione al DBMS

- Non occorre effettuare la connessione al DBMS all'interno di una stored procedure
  - il DBMS che esegue le istruzioni SQL è lo stesso in cui è memorizzata la stored procedure

Gestione delle istruzioni SQL

- Nelle istruzioni SQL presenti nella stored procedure è possibile far riferimento a variabili o parametri
  - il formalismo dipende dal linguaggio utilizzato
- Per leggere il risultato di un'interrogazione che restituisce un insieme di tuple
  - è necessario definire un cursore
    - simile all'embedded SQL

Stored procedure in Oracle

- Creazione di una stored procedure in Oracle

  CREATE [OR REPLACE] PROCEDURE *nomeStoredProcedure*
  [(*elencoParametri*)]
  IS (*istruzioneSQL*|*codicePL/SQL*);

- La stored procedure può essere associata a
  - una singola istruzione SQL
  - un blocco di codice scritto in PL/SQL

Parametri

- Ogni parametro nell'elenco *elencoParametri* è specificato nella forma

  nomeParametro [IN|OUT|IN OUT] [NOCOPY] *tipoDato*

  - *nomeParametro*
    - nome associato al parametro
  - *tipoDato*
    - tipo del parametro
    - sono utilizzati i tipi di SQL
  - le parole chiave IN, OUT, IN OUT e NOCOPY specificano le operazioni che si possono eseguire sul parametro
    - default IN

Parametri

- Parola chiave IN
  - il parametro è utilizzabile solo in lettura
- Parola chiave OUT
  - il parametro è utilizzabile solo in scrittura
- Parola chiave IN OUT
  - il parametro può essere sia letto, sia scritto all'interno della stored procedure
- Per i parametri di tipo OUT e IN OUT il valore finale è assegnato solo quando la procedura termina in modo corretto
  - la parola chiave NOCOPY permette di scrivere direttamente il parametro durante l'esecuzione della stored procedure

Struttura base di una procedura PL/SQL

- Ogni blocco PL/SQL presente nel corpo di una stored procedure deve avere la seguente struttura

```
[dichiarazioneVariabilieCursori]
BEGIN
codiceDaEseguire
[EXCEPTION codiceGestioneEccezioni]
END;
```

Linguaggio PL/SQL

- Il linguaggio PL/SQL è un linguaggio procedurale
  - dispone delle istruzioni classiche dei linguaggi procedurali
    - strutture di controllo IF-THEN-ELSE
    - cicli
  - dispone di strumenti per
    - l'esecuzione di istruzioni SQL
    - la scansione dei risultati
      - cursori
- Le istruzioni SQL
  - sono normali istruzioni del linguaggio PL/SQL
    - non sono precedute da parole chiave
    - non sono parametri di funzioni o procedure

Esempio: istruzione di aggiornamento

- Aggiornamento della sede del fornitore identificato dal valore presente nel parametro *codiceFornitore* con il valore presente in *nuovaSede*

```
CREATE PROCEDURE aggiornaSede(codiceFornitore
VARCHAR(5), nuovaSede VARCHAR(15))
IS
BEGIN
        UPDATE F SET Sede=nuovaSede
        WHERE codF=CodiceFornitore;
END;
```

Cursori in PL/SQL

- Dichiarazione

  CURSOR *nomeCursore* IS *interrogazioneSQL*
  [FOR UPDATE];

- Apertura

  OPEN *nomeCursore*;

- Lettura tupla successiva

  FETCH *nomeCursore* INTO *elencoVariabili*;

- Chiusura cursore

  CLOSE *nomeCursore*;

Esempio: selezione fornitori

- Presentare a video il codice e il numero di soci dei fornitori la cui sede è contenuta nel parametro *VarSede*

Esempio: selezione fornitori

```
CREATE PROCEDURE fornitoriSede(VarSede IN F.Sede%Type) IS
/*
      Definizione variabili e cursori
*/
codiceF    F.CodF%Type;
numSoci    F.NSoci%Type;

CURSOR fornitoriSelezionati IS
SELECT CodF,NSoci FROM F WHERE Sede = VarSede;

BEGIN
  DBMS_OUTPUT.PUT_LINE('Elenco fornitori di '||VarSede);
  /*
     Apertura cursore
  */
  OPEN fornitoriSelezionati;
```

Esempio: selezione fornitori

```
/*
   Analisi dati selezionati dall'interrogazione
*/

LOOP
    FETCH fornitoriSelezionati INTO codiceF, numSoci;
    /*
        Uscita dal ciclo quando non ci sono più tuple da analizzare
    */
    EXIT WHEN fornitoriSelezionati%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE(codiceF||','||numSoci);
END LOOP;

/*
   Chiusura cursore
 */
 CLOSE fornitoriSelezionati;
END;
```

Esempio: selezione fornitori

```
CREATE PROCEDURE fornitoriSede(VarSede IN F.Sede%Type) IS
/*
      Definizione variabili e cursori
*/
codiceF    F.CodF%Type;
numSoci    F.NSoci%Type;

CURSOR fornitoriSelezionati IS
SELECT CodF,NSoci FROM F WHERE Sede = VarSede;

BEGIN
  DBMS_OUTPUT.PUT_LINE('Elenco fornitori di '||VarSede);
  /*
     Apertura cursore
  */
  OPEN fornitoriSelezionati;
```

Definizione parametri

Esempio: selezione fornitori

```
CREATE PROCEDURE fornitoriSede(VarSede IN F.Sede%Type) IS
/*
      Definizione variabili e cursori
*/
codiceF    F.CodF%Type;
numSoci    F.NSoci%Type;

CURSOR fornitoriSelezionati IS
SELECT CodF,NSoci FROM F WHERE Sede = VarSede;

BEGIN
  DBMS_OUTPUT.PUT_LINE('Elenco fornitori di '||VarSede);
  /*
     Apertura cursore
  */
  OPEN fornitoriSelezionati;
```

Assegna a VarSede il tipo di F.Sede

Esempio: selezione fornitori

```
CREATE PROCEDURE fornitoriSede(VarSede IN F.Sede%Type) IS
/*
      Definizione variabili e cursori
*/
codiceF    F.CodF%Type;
numSoci    F.NSoci%Type;

CURSOR fornitoriSelezionati IS
SELECT CodF,NSoci FROM F WHERE Sede = VarSede;

BEGIN
  DBMS_OUTPUT.PUT_LINE('Elenco fornitori di '||VarSede);
  /*
     Apertura cursore
  */
  OPEN fornitoriSelezionati;
```

Definizione variabili e cursori

Esempio: selezione fornitori

```
CREATE PROCEDURE fornitoriSede(VarSede IN F.Sede%Type) IS
/*
      Definizione variabili e cursori
*/
codiceF    F.CodF%Type;
numSoci    F.NSoci%Type;

CURSOR fornitoriSelezionati IS
SELECT CodF,NSoci FROM F WHERE Sede = VarSede;

BEGIN
  DBMS_OUTPUT.PUT_LINE('Elenco fornitori di '||VarSede);
  /*
     Apertura cursore
  */
  OPEN fornitoriSelezionati;
```

Apertura cursore

Esempio: selezione fornitori

```
/*
   Analisi dati selezionati dall'interrogazione
*/

LOOP
    FETCH fornitoriSelezionati INTO codiceF, numSoci;
    /*
        Uscita dal ciclo quando non ci sono più tuple da analizzare
    */
    EXIT WHEN fornitoriSelezionati%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE(codiceF||','||numSoci);
END LOOP;

/*
   Chiusura cursore
 */
 CLOSE fornitoriSelezionati;
END;
```

Ciclo di lettura dei dati

Esempio: selezione fornitori

```
/*
   Analisi dati selezionati dall'interrogazione
*/

LOOP
    FETCH fornitoriSelezionati INTO codiceF, numSoci;
    /*
        Uscita dal ciclo quando non ci sono più tuple da analizzare
    */
    EXIT WHEN fornitoriSelezionati%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE(codiceF||','||numSoci);
END LOOP;

/*
   Chiusura cursore
 */
  CLOSE fornitoriSelezionati;
END;
```

Chiusura cursore

SQL per le applicazioni

Confronto tra le alternative

Embedded SQL, CLI e Stored procedure

- Le tecniche proposte per l'integrazione del linguaggio SQL nelle applicazioni hanno caratteristiche diverse
- Non esiste un approccio sempre migliore degli altri
  - dipende dal tipo di applicazione da realizzare
  - dipende dalle caratteristiche delle basi di dati
    - distribuite, eterogenee
- È possibile utilizzare soluzioni miste
  - invocazione di stored procedure tramite CLI o embedded SQL

Embedded SQL vs Call Level Interface

- Embedded SQL
  - (+) precompila le interrogazioni SQL statiche
    - più efficiente
  - (-) dipendente dal DBMS e dal sistema operativo usato
    - a causa della presenza del precompilatore
  - (-) generalmente non permette di accedere contemporaneamente a più basi di dati diverse
    - in ogni caso, è un'operazione complessa

Embedded SQL vs Call Level Interface

- Call Level Interface
  - (+) indipendente dal DBMS utilizzato
    - solo in fase di compilazione
      - la libreria di comunicazione (driver) implementa un'interfaccia standard
      - il funzionamento interno dipende dal DBMS
    - il driver è caricato e invocato dinamicamente a runtime
  - (+) non necessita di un precompilatore

Embedded SQL vs Call Level Interface

- Call Level Interface
  - (+) permette di accedere dalla stessa applicazione a più basi di dati
    - anche eterogenee
  - (-) usa SQL dinamico
    - minore efficienza
  - (-) solitamente supporta un sottoinsieme di SQL

Stored procedure vs approcci client side

- Stored procedure
  - (+) maggiore efficienza
    - sfrutta la forte integrazione con il DBMS
    - riduce la quantità di dati inviati in rete
    - le procedure sono precompilate

Stored procedure vs approcci client side

- Stored procedure
  - (-) dipendente dal DBMS utilizzato
    - usa un linguaggio ad hoc del DBMS
    - solitamente non portabile da un DBMS a un altro
  - (-) i linguaggio utilizzati offrono meno funzionalità dei linguaggi tradizionali
    - assenza di funzioni per la visualizzazione complessa dei risultati
      - grafici e report
    - meno funzionalità per la gestione dell'input

Stored procedure vs approcci client side

- Approcci client side
  - (+) basati su linguaggi di programmazione tradizionali
    - più noti ai programmatori
    - compilatori più efficienti
    - maggiori funzionalità per la gestione di input e output
  - (+) in fase di scrittura del codice, maggiore indipendenza dal DBMS utilizzato
    - solo per gli approcci basati su CLI
  - (+) possibilità di accedere a basi di dati eterogenee

Stored procedure vs approcci client side

- Approcci client side
  - (-) minore efficienza
    - minore integrazione con il DBMS
    - compilazione delle istruzioni SQL a tempo di esecuzione
      - soprattutto per approcci basati su CLI