😷

Capitolo 10 (Sviluppo applicazioni per basi di dati)

SQL e applicazioni

In applicazioni complesse, l’utente non vuole eseguire comandi SQL, ma programmi, con poche scelte

SQL non basta, sono necessarie funzionalità da gestire:

- Input (scelte dell’utente e parametri)
- Output (con dati che non sono relazioni o se si vuole una presentazione complessa)
- Per gestire il controllo

Le applicazioni sono scritte in linguaggi di programmazione tradizionali

Conflitto di impedenza

“Disaccoppiamento di impedenza” fra base di dati e linguaggio

- Linguaggi: operazioni su singole variabili o oggetti
- SQL: operazioni su relazioni (insiemi di ennuple)

Altre differenze:

- Accesso ai dati e correlazione

    - Linguaggi: dipende dal paradigma e dai tipi disponibili; ad esempio scansione di liste o “navigazione” tra oggetti
    - SQL: Join (ottimizzabile)
- Tipi di base:
    - Linguaggi: numeri, stringhe, booleani, …
    - SQL: CHAR, VARCHAR, DATE, …
- Costruttori di tipo:
    - Linguaggi: dipende dal paradigma
    - SQL: relazioni e ennuple

SQL e linguaggi di programmazione

Tecniche principali:

- SQL immerso (”Embedded SQL”)
    - “SQL statico”
- SQL dinamico
- Call Level Interface (CLI)
    - SQL/CLI, JDBC

SQL immerso

Le istruzioni SQL sono “immerse” nel programma scritto in linguaggio ospite

Un precompilatore del DBMS analizza il programma e lo traduce in un sorgente nel linguaggio ospite, sostituendo istruzioni SQL con chiamate a funzioni di un’API del DBMS

Esempio:

```c
#include <stdlib.h>

int main(void) {
    exec sql begin declare section;

    char *NomeDip = "Manutenzione";
    char *CittaDip = "Pisa";
    int NumeroDip = 20;

    exec sql end declare section;

    exec sql connect to utente@librobd;

    if (sqlca.sqlcode != 0)
        printf("Connessione al DB non riuscita\n");
    else {
        exec sql insert into Dipartimento
            values(:NomeDip, :CittaDip, :NumeroDip);
        exec sql disconnect all;
    }

    return 0;
}
```

`EXEC SQL` denota le porzioni di interesse del precompilatore:

- Definizioni dei dati
- Istruzioni SQL

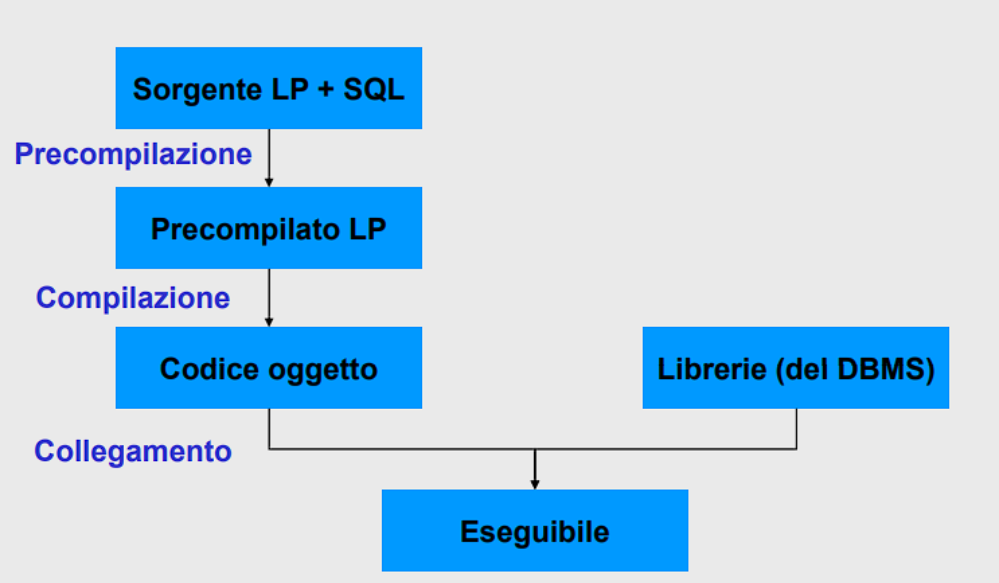
Le variabili del programma possono essere usate come “parametri” nelle istruzioni SQL (precedute da ‘:’) dove sintatticamente sono ammesse costanti

`sqlc` è una struttura dati per la comunicazione fra programma e DBMS

`sqlcode` è un campo di `sqlca` che mantiene il codice di errore dell’ultimo comando SQL eseguito:

- 0: successo
- Altro valore: errore o anomalia

Fasi:



Altro esempio:


```
#include <stdlib.h>

int main(void) {
    exec sql connect to universita
        user pguser identified by pguser;
    exec sql create table studente
        (matricola integer primary key,
         nome varchar(20), annodicorso integer);
    exec sql disconnect;

    return 0;
}
```

L’esempio precompilato:

```
/* These include files are added by the preprocessor */
#include <ecpgtype.h>
#include <ecpglib.h>
#include <ecpgerrno.h>
#include <sqlca.h>
int main() {
  ECPGconnect(__LINE__, "universita" , "pguser" ,
    "pguser" , NULL, 0);
  ECPGdo(__LINE__, NULL, "create table studente (
  matricola integer primary key , nome varchar ( 20 ) ,
  annodicorso integer )", ECPGt_EOIT, ECPGt_EORT);
  ECPGdisconnect(__LINE__, "CURRENT");
  return 0;
}
```

Il precompilatore è specifico della combinazione Linguaggio-DBMS-Sistema operativo

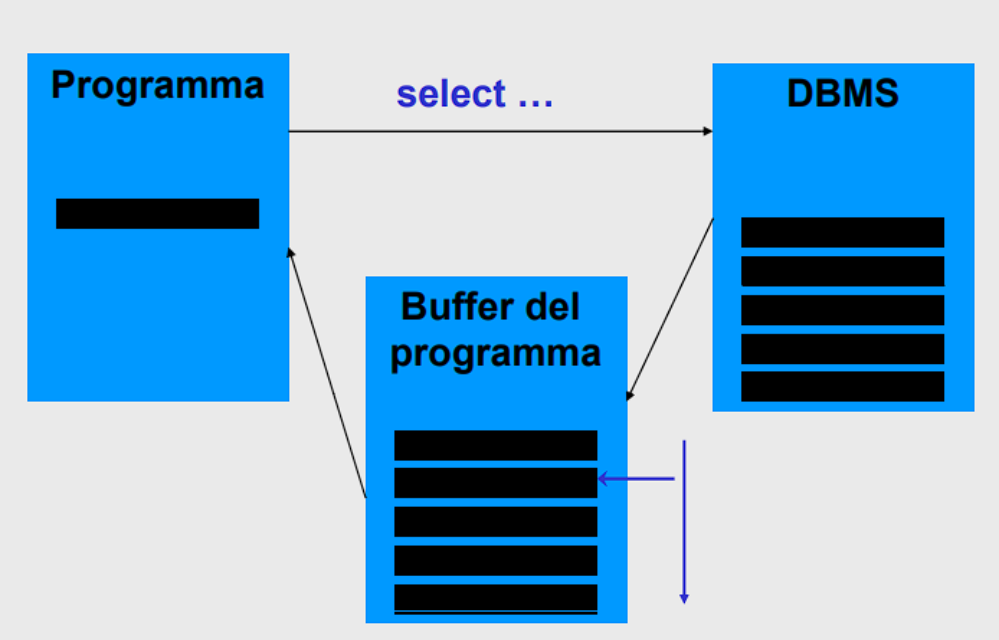
Conflitto d'impedenza in SQL immerso

Il risultato di una `select` è costituito da zero o più ennuple:

- Zero o una: ok - l'eventuale risultato può essere gestito in un record
- Più ennuple: l'insieme (in effetti, la lista) non è gestibile facilmente in molti linguaggi

Cursore: tecnica per trasmettere al programma una ennupla alla volta

Cursore:



Il cursore accede a tutte le ennuple di una interrogazione in modo globale (tutte insieme o a blocchi - è il DBMS che sceglie la strategia efficiente)

Trasmette le ennuple al programma una alla volta

Operazioni sui cursori

Definizione del cursore:

- `declare NomeCursore [scroll] cursor for Select`

Esecuzione dell'interrogazione:

- `open NomeCursore`

Utilizzo dei risultati (una ennupla alla volta):

- `fetch NomeCursore into ListaVariabili`

Disabilitazione del cursore:

- `close cursor NomeCursore`

Accesso alla ennupla corrente (di un cursore su singola relazione ai fini di aggiornamento):

- `current of NomeCursore` nella clausola `where`

Esempio:

```
char citta[20], nome[20];
long reddito, aumento;

printf("Nome della città? ");
scanf("%s", citta);

EXEC SQL DECLARE P CURSOR FOR
    SELECT NOME, REDDITO
    FROM PERSONE
    WHERE CITTA = :citta;
EXEC SQL OPEN P;
EXEC SQL FETCH P INTO :nome, :reddito;

while (sqlcode == 0) {
    printf("Nome della persona: %s, aumento? ", nome);
    scanf("%l", &aumento);

    EXEC SQL UPDATE PERSONE
        SET REDDITO = REDDITO + :aumento
        WHERE CURRENT OF P

    EXEC SQL FETCH P INTO :nome, :reddito
}

EXEC SQL CLOSE CURSOR P
```


Esempio 2:

```
void VisualizzaStipendiDipart(char NomeDip[]) {
    char Nome[20], Cognome[20];
    long int Stipendio;
```

```
    EXEC SQL DECLARE ImpDip CURSOR FOR
        SELECT Nome, Cognome, Stipendio
        FROM Impiegato
        WHERE Dipart = :NomeDip;

    EXEC SQL OPEN ImpDip;
    EXEC SQL FETCH ImpDip INTO :Nome, :Cognome, :Stipendio;

    printf("Dipartimento %s\n", NomeDip);

    while (sqlcode == 0) {
        printf("Nome e cognome dell'impiegato: %s %s\n", Nome, Cognome);
        printf("Attuale stipendio: %d\n", Stipendio);

        EXEC SQL FETCH ImpDip INTO :Nome, :Cognome, :Stipendio;
    }

    EXEC SQL CLOSE CURSOR ImpDip;
}
```

Per aggiornamenti e **query scalari** (interrogazioni che restituiscono sempre una sola riga (ennupla)) il cursore non serve:

```
SELECT Nome, Cognome INTO :nomeDip, :cognomeDip
FROM Dipendente
WHERE Matricola = :matrDip;
```


I cursori possono ricondurre la programmazione ad un livello troppo basso, pregiudicando la capacità dei DBMS di ottimizzare le interrogazioni:

SQL dinamico

Non sempre le istruzioni SQL sono note quando si scrive il programma

Allo scopo, è stata definita una tecnica completamente diversa, chiamata **Dynamic SQL** che permette di eseguire istruzioni SQL costruite dal programma (o addirittura ricevute dal programma attraverso parametri o da input)

Non è banale gestire i parametri e la struttura dei risultati (non noti a priori)

Le operazioni SQL possono essere:

- Eseguite immediatamente
  - `execute immediate SQLStatemenet`
- Prima “preparate” (analizza un’istruzione SQL e la traduce nel linguaggio interno del DBMS, associando alla traduzione dell’istruzione un nome):
  - `prepare CommandName from SQLStatement`
  - E poi eseguite (anche più volte):
  - `execute CommandName [into TargetList] [using ParameterList]`

Call Level Interface

Indica genericamente interfacce che permettono di inviare richieste al DBMS per mezzo di parametri trasmessi a funzioni

Il programmatore ha già a disposizione un insieme di primitive che permettono di interagire con il DBMS, si tratta di uno strumento più flessibile e meglio integrato col linguaggio di programmazione

- Standard SQL/CLI
- ODBC: implementazione proprietaria di SQL/CLI
- JDBC: una CLI per il mondo java

SQL immerso vs CLI

SQL immerso permette:

- Precompilazione (e quindi efficienza)
- Uso di SQL completo

CLI:

- Indipendente dal DBMS
- Permette di accedere a più basi di dati, anche eterogenee

JDBC

Il modulo Java Database Connectivity (JDBC) è una API (Application Programming Interface) di Java (intuitivamente: una libreria) per l'accesso a basi di dati, in modo indipendente dalla spcifica tecnologia

Ciascun DBMS fornisce un driver specifico che:

- Viene caricato a run-time
- Traduce le chiamate alle funzioni di JDBC a quelle del database

Modalità di accesso al database

JDBC è generalmente usato in due architetture di sistema:

- L'applicazione Java "parla" direttamente col database
- Un livello intermedio invia i comandi SQL al database, eseguendo controlli sugli accessi ed effettuando aggiornamenti
    - Tutto il controllo sull'accesso ai dati è affidato allo strato centrale (middle-tier), che può avvalersi di un middleware o un application server

    - Può avere vantaggi in termini di scalabilità

Architettura di JDBC

Un sistema che usa JDBC ha quattro componenti principali:

- **Applicazione:** inizia e termina la connessione, imposta le transazioni, invia i comandi SQL, recepisce risultati. Tutto avviene tramite l'API JDBC (nei sistemi three tiers se ne occupa lo stato intermedio)
- **Gestore di driver:** carica i driver, passa le chiamate al driver corrente, esegue controlli sugli errori
- **Driver:** stabilisce la connessione, inoltra le richieste e restituisce i risultati, trasforma dati e formati di errore dalla forma dello specifico DBMS allo standard JDBC
- **Sorgente di dati:** elabora i comandi provenienti dal driver e restituisce i risultati

Caratteristiche di JDBC

Esecuzione di comandi SQL

- DDL (Data Definition Language)
- DML (Data Manipulation Language)
- Manipolazione dei risultati tramite result set (una forma di cursore)
- Reperimento dei metadati
- Gestione delle transazioni
- Gestione di errori ed eccezioni
- Definizione di stored procedure scritte in Java (supportate da alcuni DBMS)

Software: cosa occorre

- Piattaforma Java
  - Il package `java.sql`: funzionalità di base di JDBC
  - Il package `javax.sql`: funzionalità più avanzata (ad es. utili per la gestione di pooling di connessioni o per la programmazione a componenti)
- Driver JDBC per il DBMS a cui ci si vuole connettere
- DBMS

Driver JDBC

Il driver JDBC per un certo DBMS viene distribuito in genere dalla casa produttrice del DBMS

È di fatto una libreria Java che va collegata in fase di esecuzione

Questo va fatto:

- Settando opportunatamente le variabili d'ambiente
- Configurando l'ambiente di sviluppo Java che si sta usando (caricare il .jar del driver nel progetto)

Funzionamento di JDBC, in breve:

- Caricamento del driver
- Apertura della connessione alla base di dati
- Richiesta di esecuzione di istruzioni SQL
- Elaborazione dei risultati delle istruzioni SQL

Esempio di programma:

```
import java.sql.*;

public class PrimoJDBC {
    public static void main(String[] args) {
        Connection con = null;

        try {
            Class.forName("com.mysql.jdbc.Driver");
            // Corsi nome schema
            String url = "jdbc:mysql://localhost:3306/corsi";
            String username = "<username>"; String pwd = "<pwd>";
            con = DriverManager.getConnection(url, username, pwd);
        }
        catch (Exception e) {
            System.out.println("Connessione fallita");
        }

        try {
            Statement query = con.createStatement();
            ResultSet result = query.executeQuery("SELECT * FROM Corsi");
            while (result.next()) {
                String nomeCorso = result.getString("NomeCorso");
                System.out.println(nomeCorso);
            }
        }
        catch (Exception e) {
            System.out.println("Errore nell'interrogazione");
        }
    }
}
```


L’interfaccia JDBC è contenuta nel package `java.sql`

`Class.forName("com.mysql.jdbc.Driver");` serve a caricare il driver

Connessione: oggetto di tipo `Connection` che costituisce un collegamento attivo fra programma Java e base di dati; viene creato da:

```
String url = "jdbc:odbc:Corsi";
con = DriverManager.getConnection(url);
```

Statement

Interfaccia i cui oggetti consentono di inviare, tramite una connessione, istruzioni SQL e di ricevere risultati

Un oggetto di tipo `Statement` viene creato con il metodo `createStatement` di `Connection`

I metodi dell'interfaccia `Statement`:

- `executeUpdate` per specificare aggiornamenti o istruzioni DDL
- `executeQuery` per specificare interrogazioni e ottenere un risultato
- `execute` per specificare istruzioni non note a priori

ResultSet

I risultati delle interrogazioni sono forniti in oggetti di tipo `ResultSet` (interfaccia definita in `java.sql`)

In sostanza, un result set è una sequenza di ennuple su cui si può "navigare" (in avanti, indietro e anche con accesso diretto) e dalla cui ennupla "corrente" si possono estrarre i valori degli attributi

Metodi principali:

- `next()`
- `getXXX(posizione)`
  - Es. `getString(3); getInt(2);`
- `getXXX(nomeAttributo)`
  - Es. `getString("Cognome"); getInt("Codice");`

Specializzazioni di Statement

`PreparedStatement` permette di utilizzare codice SQL già compilato, eventualmente parametrizzato rispetto alle costanti

- In generale più efficiente di `Statement`
- Permette di distinguere più facilmente istruzioni e costanti (e apici nelle costanti)
- I metodi `setXXX(,)` permettono di definire i parametri

`CallableStatement` permette di utilizzare “stored procedure”

Esempio 1:

```
import java.sql.*;

public class PrimoJDBC {
    public static void main(String[] args) {
        Connection con = null;

        try {
            Class.forName("com.mysql.jdbc.Driver");
            // Corsi nome schema
            String url = "jdbc:mysql://localhost:3306/corsi";
            String username = "<username>"; String pwd = "<pwd>";
            con = DriverManager.getConnection(url, username, pwd);
        }
        catch (Exception e) {
            System.out.println("Connessione fallita");
        }

        try {
            PreparedStatement pquery = con.prepareStatement(
                "SELECT * FROM Corsi WHERE NomeCorso LIKE ?"
            );
            String param = JOptionPane.showInputDialog("Nome corso (anche parziale)? ");
            param = "%" + param + "%";
            pquery.setString(1, param);
            ResultSet result = pquery.executeQuery();
            while (result.next()) {
                String nomeCorso = result.getString("NomeCorso");
                System.out.println(nomeCorso);
            }
        }
        catch (Exception e) {
            System.out.println("Errore nell'interrogazione");
        }
    }
}
```

Esempio 2:

```
import java.sql.*;

public class PrimoJDBC {
    public static void main(String[] args) {
        Connection con = null;

        try {
            Class.forName("com.mysql.jdbc.Driver");
            // Corsi nome schema
            String url = "jdbc:mysql://localhost:3306/corsi";
            String username = "<username>"; String pwd = "<pwd>";
            con = DriverManager.getConnection(url, username, pwd);
        }
        catch (Exception e) {
            System.out.println("Connessione fallita");
        }

        try {
            CallableStatement pquery = con.prepareCall("{ call queryCorso(?) }");
            String param = JOptionPane.showInputDialog("Nome corso (anche parziale)? ");
            param = "*" + param + "*";
            pquery.setString(1, param);
            ResultSet result = pquery.executeQuery();
            while (result.next()) {
                String nomeCorso = result.getString("NomeCorso");
                System.out.println(nomeCorso);
            }
        }
        catch (Exception e) {
            System.out.println("Errore nell'interrogazione");
        }
    }
}
```


Altre funzionalità

Molte fra cui:

- Username e password
- Aggiornamento dei ResultSet

- Richiesta di metadati
- Gestione di transazioni

Transazioni in JDBC

Scelta della modalità delle transazioni: un metodo definito nell'interfaccia `Connection`:

- `setAutoCommit(boolean autoCommit)`
  - `con.setAutoCommit(true);`: (default) "autocommit": ogni operazione è una transazione
  - `con.setAutoCommit(false);`: gestione delle transazioni da programma
    - `con.commit();`
    - `con.rollback();`
    - Non c'è `begin transaction`

Procedure

SQL: 1999 (come già SQL-2) permette la definizione di procedure e funzioni (chiamate genericamente **stored procedures**)

Le stored procedures sono parte dello schema

Esempio:

```
PROCEDURE AssignCity(:Dep CHAR(20), :City CHAR(20))
    UPDATE Department
    SET City = :City
    WHERE Name = :Dep
```

Lo standard prevede funzionalità limitate e non è molto recepito

Procedure in Oracle PL/SQL:

```
PROCEDURE Debit(ClientAccount CHAR(5), Withdrawal INTEGER) IS
    OldAmount INTEGER;
    NewAmount INTEGER;
    Threshold INTEGER;
BEGIN
    SELECT Amount, Overdraft INTO OldAmount, Threshold
    FROM BankAccount
    WHERE AccountNo = ClientAccount
    FOR UPDATE OF Amount;

    NewAmount := OldAmount - WithDrawal;

    IF NewAmount > Threshold
        THEN UPDATE BankAccount
        SET Amount = NewAmount
        WHERE AccountNo = ClientAccount;
    ELSE
        INSERT INTO OverDraftExceeded
        VALUES (ClientAccount, Withdrawal, sysdate);
    END IF;
END Debit;
```