7

parte quarta

Applicazioni Web

# Accesso ai database con JDBC

## OBIETTIVI DI APPRENDIMENTO

In questo capitolo imparerai ad usare gli strumenti Java per l'accesso ai database.

Sarai in grado di effettuare la connessione al database e le operazioni di manipolazione e interrogazione.

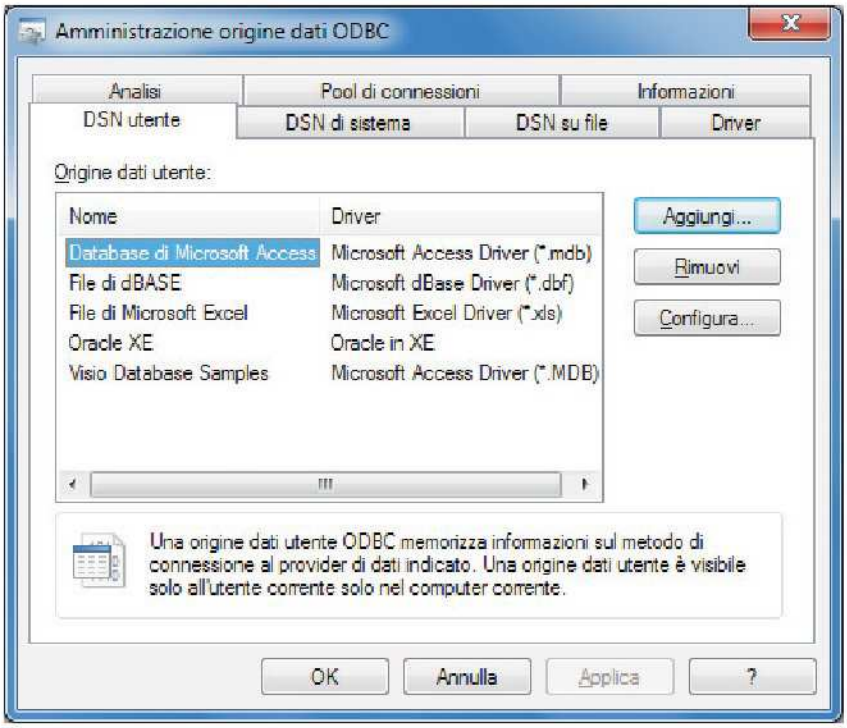
# 1 I driver per la connessione al database

In informatica il termine **driver** indica, in generale, un programma che consente a una periferica hardware di comunicare con l'unità centrale di un computer: il programma driver ritrova i dati richiesti e li trasferisce al programma richiedente. Analogamente, il **driver per database** è un programma standard che consente di trasferire i dati presenti in un database: quando un programma vuole accedere al database creato con un altro applicativo, deve utilizzare il driver per quel database.
**ODBC** (*Open Database Connectivity*) è l'interfaccia software standard in ambiente Windows che consente ai programmatori di scrivere in modo semplice le applicazioni per connettersi ai database e ritrovare i dati in essi contenuti. I database possono essere creati con prodotti DBMS diversi: ODBC possiede i driver software per database ed evita quindi di dover scrivere un programma diverso per ciascun tipo di database.

Per visualizzare le proprietà di ODBC in Windows e per fissarne le impostazioni, occorre fare le seguenti operazioni:

1. Nel *Pannello di controllo* scegliere *Sistema e sicurezza*, *Strumenti di amministrazione* e infine fare doppio clic sull'icona *Origine dati (ODBC)*.

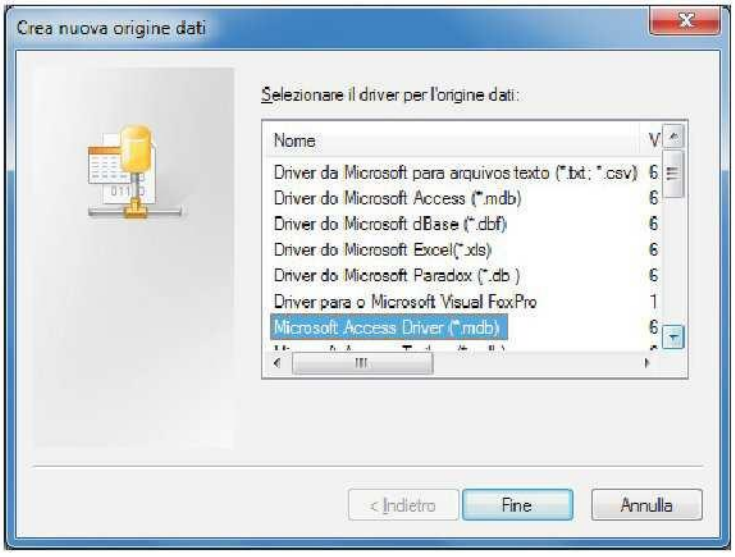
2. Fare clic sulla scheda *DSN utente*, *DSN di sistema* o *DSN su file*, a seconda del tipo di origine dati.




3. Per modificare la definizione di un'origine dati esistente:
   - selezionare l'origine dati dall'elenco,
   - scegliere il pulsante *Configura*,
   - completare le finestre di dialogo.

**4.** Per definire una nuova origine dati per un driver installato, nella finestra iniziale, scegliere il pulsante *Aggiungi*; nella nuova finestra che si apre (*Crea nuova origine dati*) scegliere il driver adatto per il database e poi fare clic sul pulsante *Fine*.

La sigla **DSN** (*Data Source Name*) indica il nome della sorgente di dati; è il nome che identifica il database quando viene utilizzato da un'applicazione Web in Internet o in una Intranet aziendale.




L'origine dei dati può essere:

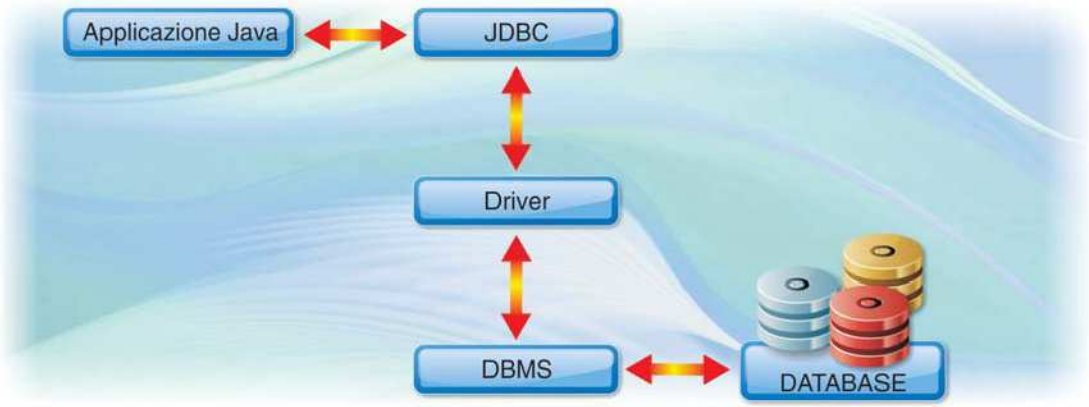
- DSN utente (*User DSN*) per fissare una sorgente di dati per uno specifico utente;
- DSN di sistema (*System DSN*) per le sorgenti di dati disponibili per molti utenti che accedono allo stesso computer di rete;
- DSN su file (*File DSN*) per configurare un DSN come file con estensione *.dsn* che può essere condiviso da più utenti che dispongono dello stesso driver installato.


MATERIALI ON LINE

## 1. Richiami sui database e il linguaggio SQL

## 2 La tecnologia JDBC

**JDBC** (*Java DataBase Connectivity*) mette a disposizione una libreria di classi Java per interfacciare l'accesso ai database che usano il linguaggio standard SQL, con l'obiettivo di fornire un'uniformità di accesso ai prodotti DBMS relazionali.
Attraverso JDBC è possibile richiamare comandi SQL direttamente, fornendo la base per costruire applicazioni per l'accesso ai database.




Secondo la terminologia di Java il software JDBC è classificato come una **API** (*Application Programming Interface*), perché fornisce un'interfaccia analoga a quella offerta dall'API per i sistemi operativi moderni: in questo caso fornisce i driver ai database per la connessione, l'esecuzione dei comandi SQL e le transazioni verso i database gestiti con i più diffusi prodotti DBMS.
L'API JDBC è contenuta nel JDK di Oracle.
JDBC comprende anche il **JDBC-ODBC Bridge**, un ponte verso i driver standard ODBC, che rende possibile la connessione ai database creati con prodotti DBMS molto diffusi, tra i quali *Microsoft Access*.

Le operazioni di manipolazione e di interrogazione possono essere eseguite su un database locale usando **applicazioni Java** oppure sul database di un server di rete, usando le tecnologie del Web.
I programmi che usano JDBC devono specificare l'uso del package **java.sql** del linguaggio Java e quindi devono contenere l'istruzione

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

La classe **DriverManager** tiene traccia dei driver disponibili e li gestisce stabilendo la connessione tra un driver e un particolare database.
Il programmatore carica il driver esplicitandolo con il comando

**Class.forName(**"*nome del driver*"**)**

indicando il nome della specifica classe del driver JDBC, cioè la classe che consente a Java di accedere al database.
Per esempio:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

I comandi e i nomi delle classi sono *case-sensitive*, si faccia quindi attenzione all'uso delle lettere minuscole e maiuscole.

La classe indicata come parametro contiene un driver fornito con JDK per accedere ai database che utilizzano lo standard ODBC (tra cui Access) e quindi è in grado di convertire le richieste al DBMS dal formato JDBC a quello ODBC.
È buona norma inserire la **gestione delle eccezioni** nella fase di caricamento del driver utilizzando la seguente struttura:

```
try
{
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
}
catch(ClassNotFoundException e)
{
  System.err.print("ClassNotFoundException: ");
  System.err.println(e.getMessage());
}
```

L'eccezione **ClassNotFoundException** viene rilevata se non è stata trovata la classe del driver. Usando il metodo **getMessage()** si ottiene una descrizione completa dell'eccezione che si è verificata.
Il programma deve poi richiedere alla classe **DriverManager** di creare la connessione a un particolare database specificando il nome del database, ed eventualmente il nome dell'utente (*username*) e la *password* per l'autorizzazione nell'accesso.
Il nome del database è specificato all'interno di una stringa che contiene tre informazioni:
- protocollo
- sottoprotocollo
- nome del database secondo il formato previsto dal driver.

Per esempio:

```
String url = "jdbc:odbc:db1";
```

In questa assegnazione viene indicato il protocollo *jdbc*, il sottoprotocollo *odbc* (driver ODBC) e il nome *db1* definito come DSN nel caso si utilizzi ODBC.
La stringa viene in genere chiamata **URL** di JDBC perché richiama il concetto di indirizzo Internet: in effetti identifica il database attraverso il protocollo *jdbc*, così come un browser distingue un sito Internet tramite i protocolli *http* o *ftp*.
Il *DriverManager* controlla che ci sia un driver disponibile in grado di gestire la richiesta di connessione; quando lo trova può effettuare la connessione, con l'istruzione:

```
Connection con = DriverManager.getConnection(url, "username", "password");
```

L'oggetto **Connection** è l'oggetto restituito dal metodo **getConnection** della classe *DriverManager*.
Se non sono stati fissati username e password, l'istruzione viene usata nel seguente formato:

```
Connection con = DriverManager.getConnection(url, "", "");
```

La connessione rappresenta una sessione di lavoro con uno specifico database: all'interno di una connessione vengono eseguiti i comandi SQL e si ottengono i risultati con i dati richiesti.

**AUTOVERIFICA**

Domande da 1 a 3 pag. 408
Problemi da 1 a 2 pag. 409

# 3 Manipolazione dei dati

Per gli esempi presentati successivamente si farà riferimento a un database di nome *Utenti*, contenente la tabella *Anag* con i dati delle persone. Il tracciato del record anagrafico è il seguente:

| Nome | Tipo | Dimensione |
|---|---|---|
| ID | Contatore | Intero lungo |
| Cognome | Testo | 40 |
| Nome | Testo | 40 |
| DataNascita | Data/ora | 8 |
| Luogo | Testo | 40 |
| Sesso | Testo | 1 |
| Via | Testo | 50 |
| CAP | Testo | 5 |
| Città | Testo | 30 |
| Provincia | Testo | 2 |
| Telefono | Testo | 12 |
| DataAggio | Data/ora | 8 |
| CodiceFiscale | Testo | 16 |
| Banca | Testo | 30 |
| Agenzia | Testo | 30 |
| NumeroConto | Testo | 15 |
| CAB | Testo | 6 |
| ABI | Testo | 6 |

Il database, implementato in Access, è contenuto nel file *Utenti.mdb* ed è associato tramite ODBC al nome **db1**, specificato come nome della sorgente di dati (DSN).

Dopo aver stabilito la connessione, come visto nel precedente paragrafo, si possono inviare al database i comandi SQL.
Per far questo, prima si predispone l'istruzione SQL in una stringa, per esempio:

```
String query = "SELECT * FROM Anag";
```

Poi si predispone il comando JDBC (*statement*) con il metodo **createStatement** che crea un'istanza della classe **Statement**.
Per esempio, l'istruzione seguente usa la connessione (oggetto *con*) per costruire un comando SQL da inviare successivamente al database:

```
Statement stmt = con.createStatement();
```

I comandi SQL sono poi attivati con i due principali metodi applicati allo *statement*:

- metodo **executeUpdate** per i comandi SQL di inserimento (*Insert*), aggiornamento (*Update*) e cancellazione (*Delete*); si usa anche per i comandi SQL relativi alla creazione (*Create Table*), cancellazione (*Drop Table*), modifica della struttura (*Alter Table*) delle tabelle nel database;
- metodo **executeQuery** per le interrogazioni.

Il primo metodo restituisce il numero di righe che sono state sottoposte all'operazione di manipolazione della tabella, il secondo metodo restituisce un insieme di risultati (*ResultSet*) contenente le righe della tabella che soddisfano la condizione specificata nell'interrogazione SQL.

La **gestione delle eccezioni** per l'accesso al database tramite i comandi SQL viene gestita con una struttura come la seguente:

```
try
{
  ..............
  ..............
}
catch(SQLException ex)
{
  System.err.print("SQLException: ");
  System.err.println(ex.getMessage());
}
```

L'esempio seguente rappresenta il programma per l'inserimento di un nuovo record nella tabella del database, utilizzando il comando *Insert* di SQL.

## PROGETTO 1

**Inserire i dati di una nuova persona nella tabella delle anagrafiche.**

**PROGRAMMA JAVA** (*Inserimento.java*)

```
/* Inserimento di un nuovo record */
import java.sql.*;

public class Inserimento
{
  public static void main(String args[])
  {
    // crea un URL specificando il DSN (data source name) di ODBC
    String url = "jdbc:odbc:db1";
    Connection con;
    String cmdSQL = "INSERT INTO Anag " +
      " (ID, Cognome, Nome, DataNascita, Luogo, Sesso, " +
      " Via, CAP, Città, Provincia, Telefono, DataAggio, " +
      " CodiceFiscale, Banca, Agenzia, NumeroConto, CAB, ABI) " +
      " VALUES (005, 'Rossi', 'Angelo', '12/02/65', 'Bologna', " +
      " 'M', 'Via Lunga 7', '20102', 'Milano', 'MI', '02/2828217', " +
      " '13/05/01', 'RSSNGL65B12R342Z', 'BIPOP', 'BO2', '23456', " +
      " '078977', '023411')";
    Statement stmt;

    // connessione al database
    try
    {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
    catch(ClassNotFoundException e)
    {
      System.err.print("ClassNotFoundException: ");
      System.err.println(e.getMessage());
    }
```

```
    try
    {
      con = DriverManager.getConnection(url,"", "");
      stmt = con.createStatement();
      // esegue il comando SQL
      stmt.executeUpdate(cmdSQL);
      System.out.print("Record registrato");
      stmt.close();
      con.close();
    }
    catch(SQLException ex)
    {
      System.err.print("SQLException: ");
      System.err.println(ex.getMessage());
    }
  }
}
```

Per comodità di lettura il comando SQL è stato costruito con una stringa indicata con il nome *cmdSQL* e formata da diverse parti concatenate con l'operatore +. Si faccia attenzione all'uso corretto dei doppi apici per delimitare le parti di stringa e degli apici per delimitare i valori alfanumerici dei campi della tabella.

Al termine del programma, viene utilizzato il metodo **close** sia con l'oggetto *stmt*, per rilasciare le risorse usate dal comando SQL, che con l'oggetto *con* per interrompere la connessione al database.

Nel progetto seguente viene presentata un'applicazione Java per eliminare una persona dall'archivio anagrafico utilizzano il comando *Delete* di SQL.
In essa viene utilizzato il valore di ritorno del metodo *executeUpdate* per controllare se il record è stato effettivamente eliminato.

## PROGETTO 2

**Cancellare dall'archivio anagrafico i dati della persona avente il codice = 5.**

**PROGRAMMA JAVA** (*Cancellazione.java*)

```
/* Cancellazione di un record */
import java.sql.*;

public class Cancellazione
{
  public static void main(String args[])
  {
    String url = "jdbc:odbc:db1";
    Connection con;
    String cmdSQL = "DELETE FROM Anag WHERE ID = 5";
    Statement stmt;
    int righe;
```

```
    try
    {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
    catch(java.lang.ClassNotFoundException e)
    {
      System.err.print("ClassNotFoundException: ");
      System.err.println(e.getMessage());
    }
    try
    {
      con = DriverManager.getConnection(url,"", "");
      stmt = con.createStatement();
      righe = stmt.executeUpdate(cmdSQL);
      if (righe > 0)
      {
        System.out.print("Record eliminato");
      }
      else
      {
        System.out.print("Record da eliminare non trovato");
      }
      stmt.close();
      con.close();
    }
    catch(SQLException ex)
    {
      System.err.print("SQLException: ");
      System.err.println(ex.getMessage());
    }
  }
}
```

Il metodo *createStatement*, utilizzato negli esempi precedenti, in genere serve per attivare semplici comandi SQL che non contengono alcun parametro.
JDBC mette poi a disposizione anche il metodo **prepareStatement**, il quale crea un'istanza della classe **PreparedStatement**, che è una sottoclasse di **Statement** e che quindi ne eredita tutte le funzionalità.
Il metodo *prepareStatement* risulta particolarmente utile quando si devono eseguire istruzioni SQL con uno o più parametri, oppure istruzioni SQL semplici che però devono essere eseguite più volte.
I parametri sono indicati nel comando SQL con un punto di domanda.
Per esempio per indicare come parametri il codice della persona da aggiornare e il nuovo numero telefonico, si scrive il comando SQL in questo modo:

```
PreparedStatement stmt = con.prepareStatement(
    "UPDATE Anag SET Telefono = ? WHERE ID = ?");
```

Per assegnare poi i valori ai parametri si usano i metodi **set*XXX***, sostituendo a *XXX* uno dei tipi di dato previsti nel linguaggio Java, con l'indicazione del numero del parametro da sostituire (i parametri indicati con il punto di domanda nel comando SQL sono numerati da sinistra verso destra a partire da 1).

Per esempio:

```
stmt.setString(1, "02/8883917");
stmt.setInt(2, 365);
```

essendo *365* il valore del codice del record da aggiornare e *02/8883917* il nuovo numero telefonico.
Il telefono è il parametro 1 e il codice della persona è il parametro 2 nel comando SQL.
Il metodo *setXXX* converte i parametri nei tipi appropriati del linguaggio SQL.
A questo punto si può eseguire il comando SQL con il metodo **executeUpdate**, seguito dalle parentesi tonde vuote:

```
stmt.executeUpdate();
```

I dati utilizzati dai metodi *setXXX*, anziché essere indicati come costanti nel programma, possono essere anche acquisiti da tastiera, come mostrato nel progetto seguente.

## PROGETTO 3

**Aggiornare il numero di telefono nell'archivio anagrafico per una persona della quale viene fornito da tastiera il codice. Anche il nuovo numero di telefono deve essere inserito dall'utente.**

**PROGRAMMA JAVA** (*Aggiornamento.java*)

```
/* Aggiorna un record con i dati inseriti da tastiera */
import java.io.*;
import java.sql.*;

class Aggiornamento
{
  public static void main(String argv[])
  {
    int codice;
    String telef;

    // impostazione dello standard input
    InputStreamReader input = new InputStreamReader(System.in);
    BufferedReader tastiera = new BufferedReader(input);

    try
    {
      // crea un URL specificando il DSN (data source name) di ODBC
      String url = "jdbc:odbc:db1";

      // connessione al database
      try
      {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      }
      catch(ClassNotFoundException e)
      {
        System.err.print("ClassNotFoundException: ");
        System.err.println(e.getMessage());
      }
```

```
      Connection con = DriverManager.getConnection(url, "", "");
      // statement di preparazione all'aggiornamento con due parametri
      PreparedStatement stmt = con.prepareStatement(
          "UPDATE Anag SET Telefono = ? WHERE ID = ?");

      // inserimento da tastiera dei dati
      try
      {
        System.out.println("Codice da aggiornare ");
        String numeroLetto = tastiera.readLine();
        codice = Integer.valueOf(numeroLetto).intValue();
      }
      catch(Exception e)
      {
        System.out.println("\nNumero di codice non corretto!");
        return;
      }
      try
      {
        System.out.println("Nuovo numero telefonico ");
        telef = tastiera.readLine();
      }
      catch(Exception e)
      {
        System.out.println("\nInserimento errato");
        return;
      }

      // esegue il comando SQL
      stmt.setString(1, telef);
      stmt.setInt(2, codice);
      stmt.executeUpdate();
      System.out.println("Aggiornamento record: OK");
      stmt.close();
      con.close();
    }
    catch(SQLException ex)
    {
      System.err.print("SQLException: ");
      System.err.println(ex.getMessage());
    }
  }
}
```

Anche in questo programma si potrebbe usare il valore di ritorno del metodo *executeUpdate* per controllare che l'aggiornamento abbia avuto esito positivo, come visto nel Progetto 2 per la cancellazione.

**AUTOVERIFICA**

Domande da 4 a 5 pag. 408
Problemi da 3 a 7 pag. 409

# 4 Interrogazioni

Le interrogazioni al database vengono eseguite tramite il metodo **executeQuery**.
Per esempio, per effettuare una proiezione della *Tabella1* sulle colonne *a*, *b*, *c* occorre scrivere la seguente istruzione:

```
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Tabella1");
```

oppure, si può passare al metodo *executeQuery*, come parametro, una stringa *cmdSQL* che contiene il comando SQL.

```
String cmdSQL = "SELECT a, b,c FROM Tabella1";
...........
ResultSet rs = stmt.executeQuery(cmdSQL);
```

Il valore di ritorno del metodo *executeQuery* è un singolo **ResultSet**, cioè un insieme di risultati, contenente le righe della tabella che soddisfano la condizione specificata nell'interrogazione SQL. Il risultato di una *Select* su una tabella nei database relazionali infatti è a sua volta una tabella formata dalle righe selezionate e dalle colonne scelte.
Per scorrere questa tabella si deve far riferimento a un **cursore** che indica la riga della tabella. Dopo l'esecuzione del comando *executeQuery*, il cursore si trova automaticamente posizionato sulla prima riga del *ResultSet*: ci si può muovere sulle righe successive con il metodo **next()**.
Si supponga che i campi *a*, *b*, *c* siano rispettivamente di tipo *int*, *float*, *String*; la visualizzazione dei risultati contenuti nel *ResultSet* si ottiene con il seguente ciclo di lettura sequenziale che utilizza i metodi **get*XXX***, dove *XXX* va sostituito con uno dei tipi di dato di Java.

```
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Tabella1");
while (rs.next())
{
  int i = rs.getInt("a");
  float f = rs.getFloat("b");
  String s = rs.getString("c");
  System.out.println(i);
  System.out.println(f);
  System.out.println(s);
}
```

In alternativa, anziché il nome del campo (etichetta della colonna), si può indicare, come argomento dei metodi *getXXX* il numero della colonna contenente il dato, come nell'esempio seguente.

```
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Tabella1");
while (rs.next())
{
  int i = rs.getInt(1);
  float f = rs.getFloat(2);
  String s = rs.getString(3);
  System.out.println(i);
  System.out.println(f);
  System.out.println(s);
}
```

Il progetto seguente rappresenta un'applicazione Java che esegue un'interrogazione al database, utilizzando un comando Select di SQL.

## PROGETTO 4

**Visualizzare il codice, il cognome, il nome e il codice fiscale delle persone che sono residenti nella provincia di Milano.**

**PROGRAMMA JAVA** (*Selezione.java*)

```
/* Visualizza i dati ottenuti da un'interrogazione */
import java.sql.*;

public class Selezione
{
  public static void main(String args[])
  {
    String url = "jdbc:odbc:db1";
    Connection con;
    String query = "SELECT ID, Cognome, Nome, CodiceFiscale " +
                   "FROM Anag " +
                   "WHERE Provincia = 'MI'";
    Statement stmt;

    try
    {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
    catch(ClassNotFoundException e)
    {
      System.err.print("ClassNotFoundException: ");
      System.err.println(e.getMessage());
    }
    try
    {
      con = DriverManager.getConnection(url,"", "");
      stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(query);
      while (rs.next())
      {
        int cod = rs.getInt(1);
        String cogn = rs.getString(2);
        String nom = rs.getString(3);
        String cf = rs.getString(4);
        // visualizza i campi
        System.out.println("Codice = " + cod);
        System.out.println("Cognome = " + cogn);
        System.out.println("Nome = " + nom);
        System.out.println("Codice Fiscale = "+ cf);
        System.out.print("\n");
      }
      stmt.close();
      con.close();
    }
    catch(SQLException ex)
    {
      System.err.print("SQLException: ");
      System.err.println(ex.getMessage());
    }
  }
}
```

Il progetto seguente esegue la stessa interrogazione, selezionando però le persone sulla base della provincia fornita dall'utente del programma tramite tastiera. In questo caso occorre usare il comando **prepareStatement** per indicare un comando *Select* di SQL con il parametro.

## PROGETTO 5

**Visualizzare il codice, il cognome, il nome e il codice fiscale delle persone che sono residenti in una provincia fornita da tastiera.**

**PROGRAMMA JAVA** (*SelezParam.java*)

```
/* Interrogazione con parametro */
import java.io.*;
import java.sql.*;

public class SelezParam
{
  public static void main(String args[])
  {
    String prov;
    // impostazione dello standard input
    InputStreamReader input = new InputStreamReader(System.in);
    BufferedReader tastiera = new BufferedReader(input);
    String url = "jdbc:odbc:db1";
    Connection con;

    try
    {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
    catch(ClassNotFoundException e)
    {
      System.err.print("ClassNotFoundException: ");
      System.err.println(e.getMessage());
    }

    try
    {
      con = DriverManager.getConnection(url,"", "");
      // statement di preparazione all'aggiornamento con un parametro
      PreparedStatement stmt = con.prepareStatement(
          "SELECT ID, Cognome, Nome, CodiceFiscale " +
          "FROM Anag " +
          "WHERE Provincia = ? ");

      // inserimento da tastiera della provincia
      try
      {
        System.out.println("Provincia richiesta ");
        prov = tastiera.readLine();
      }
```

```
      catch(Exception e)
      {
        System.out.println("\nInserimento errato");
        return;
      }

      // esegue il comando SQL
      stmt.setString(1, prov);
      ResultSet rs = stmt.executeQuery();
      while (rs.next())
      {
        int cod = rs.getInt(1);
        String cogn = rs.getString(2);
        String nom = rs.getString(3);
        String cf = rs.getString(4);
        // visualizza i campi
        System.out.println("Codice = " + cod);
        System.out.println("Cognome = " + cogn);
        System.out.println("Nome = " + nom);
        System.out.println("Codice Fiscale = " + cf);
        System.out.print("\n");
      }

      stmt.close();
      con.close();
    }
    catch(SQLException ex)
    {
      System.err.print("SQLException: ");
      System.err.println(ex.getMessage());
    }
  }
}
```

Il seguente schema riassume le classi e i metodi utilizzati per la connessione al database e per le operazioni di manipolazione e interrogazione.

## METADATI

JDBC mette a disposizione del programmatore un'importante classe, chiamata **DatabaseMetaData**, che fornisce i metodi per conoscere tutte le informazioni sulla struttura del database (cioè i **metadati**) come un insieme unico, restituendo un oggetto **ResultSetMetaData** per ogni *ResultSet* ottenuto con una query.

Il comando **ResultSet.getMetaData** restituisce un oggetto *ResultSetMetaData* che fornisce numero, tipo e proprietà delle colonne ottenute con l'oggetto *ResultSet*.
Il seguente frammento di programma Java visualizza, per ogni riga dell'insieme *ResultSet* ottenuto con una query, il numero della colonna e il suo contenuto.

```
ResultSet rs = stmt.executeQuery(query);
ResultSetMetaData rsmd = rs.getMetaData();
int numberOfColumns = rsmd.getColumnCount();
int rowCount = 1;
while (rs.next())
{
  System.out.println("Riga " + rowCount + ": ");
  for (int i = 1; i <= numberOfColumns; i++)
  {
    System.out.print(" Colonna " + i + ": ");
    System.out.println(rs.getString(i));
  }
  System.out.println("");
  rowCount++;
}
```

I metodi forniti dalla classe *DatabaseMetaData*, applicati all'oggetto *ResultSetMetaData*, possono essere opportunamente utilizzati per visualizzare la struttura di una tabella del database, ottenendo l'elenco dei campi, con numero, nome e tipo di ciascuna colonna della tabella.

## PROGETTO 6

**Visualizzare il nome e il tipo per ciascuna colonna di una tabella del database.**

**PROGRAMMA JAVA** (*Struttura.java*)

```
/* Visualizza la struttura di una tabella */
import java.sql.*;

public class Struttura
{
  public static void main(String args[])
  {
    String url = "jdbc:odbc:db1";
    Connection con;
    String query = "SELECT * FROM Anag ";
    Statement stmt;
```

```
    try
    {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
    catch(ClassNotFoundException e)
    {
      System.err.print("ClassNotFoundException: ");
      System.err.println(e.getMessage());
    }

    try
    {
      con = DriverManager.getConnection(url,"", "");
      stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(query);
      ResultSetMetaData rsmd = rs.getMetaData();
      int numberOfColumns = rsmd.getColumnCount();
      for (int i = 1; i <= numberOfColumns; i++)
      {
        String n = rsmd.getColumnName(i) ;
        String t = rsmd.getColumnTypeName(i) ;
        System.out.print(" Colonna " + i + ": \t" + n);
        System.out.println("      Tipo: " + t);
      }
      stmt.close();
      con.close();
    }
    catch(SQLException ex)
    {
      System.err.print("SQLException: ");
      System.err.println(ex.getMessage());
    }
  }
}
```

**AUTOVERIFICA**

Domande da 6 a 7 pag. 408
Problemi da 8 a 12 pag. 409
Problemi di riepilogo da 13 a 15 pag. 409

MATERIALI ON LINE

**2. Dati storici sulle vendite per reparto**
**3. Forum per internet**

AUTOVERIFICA

# DOMANDE

## Connessione ai database

**1** Completa le frasi seguenti utilizzando una tra le sigle elencate alla fine della domanda.

a) La sigla ..... indica i driver per database che evitano di dover scrivere un programma diverso per ciascun tipo di database.

b) La sigla ..... indica il nome che identifica il database quando viene utilizzato da un'applicazione Web.

c) La sigla ..... indica una libreria di classi per interfacciare l'accesso ai database che usano lo standard SQL.

***DSN, DBJC, DBOC, JDBC, ODBC, DNS***

**2** Quali delle seguenti affermazioni sono vere (V) e quali false (F)?

a) Il software JDBC è classificato come una API V F

b) I programmi che usano JDBC devono necessariamente specificare l'uso del package *java.awt* V F

c) Per essere utilizzato in un'applicazione il database deve essere associato a un'origine dati ODBC V F

d) L'URL di JDBC indica il nome del driver per il database V F

e) La connessione rappresenta una sessione di lavoro con uno specifico database V F

**3** Quale delle seguenti frasi dichiara correttamente un URL di JDBC?

a) String url ="jdbcodbc:db1";
b) String url ="odbc:jdbc:db1";
c) String url ="jdbc:odbc:db1";
d) String url ="db1:odbc:jdbc";

## Operazioni di manipolazione

**4** Quale dei seguenti metodi esegue un comando di manipolazione?

a) updateExecute
b) executeSQL
c) executeQuery
d) executeUpdate

**5** Quale comando risulta particolarmente utile quando si devono eseguire istruzioni SQL con uno o più parametri?

a) prepareStatement
b) statementPrepare
c) createStatement
d) parameterStatement

## Interrogazioni

**6** Quale delle seguenti frasi rappresenta il risultato ottenuto con un comando *executeQuery*?

a) Il valore di ritorno è un ResultSet, contenente le righe della tabella ottenute con l'interrogazione SQL.
b) Il valore di ritorno è il numero di righe del ResultSet ottenuto con l'interrogazione SQL.
c) Il valore di ritorno è il numero di colonne del ResultSet ottenuto con l'interrogazione SQL.
d) Il valore di ritorno è il nome delle colonne del ResultSet ottenuto con l'interrogazione SQL.

**7** Quale delle seguenti frasi corrisponde alla descrizione della classe *DatabaseMetaData*?

a) La classe che fornisce i metodi per effettuare le operazioni di manipolazione
b) La classe che fornisce i metodi per effettuare le operazioni di interrogazione
c) La classe che fornisce i metodi per conoscere tutte le informazioni sulla struttura del database
d) La classe che fornisce i metodi per conoscere i valori contenuti nelle colonne di una tabella

# PROBLEMI

## Connessione ai database

1 Creare un database di Access, formato da due tabelle per registrare i dati sulle *automobili* e sui *motocicli*.

2 Usando il database del problema precedente, stabilire una connessione ODBC nel Pannello di Controllo di Windows, scegliendo il driver opportuno e indicando un nome DSN utente.

## Operazioni di manipolazione

3 Creare una nuova tabella in un database di Access con i campi: Codice, Descrizione e Importo. Scrivere l'applicazione Java per inserire alcuni dati di prova da programma.

4 Scrivere l'applicazione Java per inserire una nuova riga nella tabella, acquisendo i dati da tastiera.

5 Modificare l'importo, diminuendolo del 10%, di una riga della tabella della quale viene fornito da tastiera il codice.

6 Cancellare dalla tabella tutte le righe per le quali l'importo risulta inferiore a 100.

7 Cancellare dalla tabella tutte le righe per le quali l'importo risulta inferiore a una cifra prefissata e fornita da tastiera.

## Interrogazioni

8 Costruire una query che estrae dalla tabella i record aventi nel campo Importo valori superiori a 150.

9 Costruire una query parametrica che estrae dalla tabella i record aventi nel campo Importo valori superiori a una cifra prefissata e fornita da tastiera.

10 Costruire un'applicazione Java che restituisce la somma totale degli importi di tutte le righe della tabella.

11 Costruire una query parametrica che estrae dalla tabella i record aventi nel campo Codice valori inferiori a un codice prefissato e fornito da tastiera.

12 Scrivere l'applicazione Java per visualizzare la struttura della tabella, con il nome e il tipo dei campi.

# PROBLEMI DI RIEPILOGO

*Per ciascuno dei seguenti esercizi, dopo aver creato il database contenente la tabella indicata e aver definito una connessione ODBC, scrivere l'applicazione Java per eseguire l'interrogazione in SQL.*

13 Data la tabella con i dati dei *contribuenti*, contenente quattro colonne con CodiceFiscale, Cognome, Nome, Reddito, produrre l'elenco dei contribuenti aventi il codice fiscale che inizia con 'RSS'.

14 Data la tabella con i *motivi musicali*, contenente per ogni riga il Nome dell'artista, il Titolo e l'Anno di produzione, fornire l'elenco dei motivi musicali in ordine decrescente di anno, solo con il nome dell'artista e l'anno di produzione.

15 Data la tabella dei *voli aerei* con Numero del volo, Descrizione e Compagnia, dopo aver fornito da tastiera i nomi di due compagnie, si vuole calcolare quanti sono i voli appartenenti a ciascuna di esse.

Focus notes

## DATABASE

A database is any collection of related information about a subject, organised in a way that enables rapid access to selected data. Computer databases are organised as collections of data files, a data file is a set of data records organised as a group of data fields. The smallest unit of information is the data field. A data field that uniquely identifies a specific record is called a primary key.
Usually, some level of quality in terms of accuracy, availability, usability is required which implies the use of a *database management system* (DBMS). Typically, a DBMS is a software system that meets a number of requirements. The use of databases is now spread to such a wide degree that virtually every technology and product relies on databases. Simple databases consist of a single data file and one or more related index files. The data file contains the data with the information of interest while the index files, which are usually hidden from the user, provide high-speed access to specific records in the data file using a system of pointers: like when using the index of a book to reach the pages with the requested information.

## RELATIONAL DATABASE

A relational database is based on the relational data model first proposed by E. F. Codd in 1970. According to the relational model, a database is a set of tables called relations. Each table is made up of named attributes or columns of data. A row of data in the table contains as many attributes as the number of columns in the table. A great strength of the relational model is the simple logical structure, but behind this simple structure lies a strong theoretical foundation. In the terminology of the relational model a table has a degree and a cardinality. The degree of a relation is the number of attributes it contains, the cardinality of a table is the number of rows it contains and a relational database is a collection of tables with different names.

## SQL

SQL stands for *Structured Query Language*. SQL is a language used to work with databases. You can send an SQL Select command to the database to retrieve data from it, or you can insert new records in a table with the Insert command, modify one or more attribute values with the Update command and delete one or more records from a table with the Delete command. The SQL commands that modify the data in the database are identified as the DML (*Data Manipulation Language*) subset of SQL. There are also commands for creating, declaring and modifying the structure of a table or other objects of the database. These commands represent the DDL (*Data Definition Language*) part of the SQL language.

## JDBC

The JDBC is a Java API that can access any kind of tabular data, especially data stored in a relational database. JDBC helps to create tables, insert values into them, query the tables, retrieve the results of the queries, and update the tables.
JDBC helps the programmer to write Java applications that manage these programming activities:

- connect to a database
- create tables and insert values into them
- update the tables or delete rows from a table
- query the tables and process the results received from the database in answer to a query.

Focus notes

## Glossary

*API*
A software to be used as an interface by software components to communicate with each other.

*driver*
A software program designed to allow a device to communicate with a computer.
A JDBC driver is a software component enabling a Java application to interact with a database.

*field*
The element of a database having settings that determine the type of data they can store and how the data is displayed. One important setting for fields is the data type, including number, text, currency (money), and date/time. The data type limits and describes the kind of information in the field. The data type also determines how much memory the data uses.

*manipulation*
Command or action to insert, update or delete database information.

*metadata JDBC*
A collection of information about the database's structure (table, columns and column properties).

*query*
A request for information from a database that meets a set of criteria.

*query language*
A language for querying, or retrieving data from, a database or other file system. The de-facto standard query language to query a database is SQL (*Structured Query Language*).

## Acronyms

| | |
|---|---|
| **API** | Application Programming Interface |
| **DB** | DataBase |
| **DBMS** | DataBase Management System |
| **DSN** | Data Source Name |
| **JDBC** | Java DataBase Connectivity |
| **JDK** | Java Development Kit |
| **ODBC** | Open Database Connectivity |
| **RDBMS** | Relational DataBase Management System |
| **SQL** | Structured Query Language |
| **URL** | Uniform Resource Locator |

# SCHEDA DI AUTOVALUTAZIONE

## CONOSCENZE

- ☐ Driver per la connessione ai database
- ☐ Caratteristiche di JDBC
- ☐ Applicazioni Java per operazioni di manipolazione
- ☐ Interrogazioni in SQL
- ☐ Metadati del database

## ABILITÀ

- ☐ Fissare le impostazioni di ODBC
- ☐ Codificare in Java le operazioni di manipolazione usando JDBC
- ☐ Utilizzare i parametri nelle operazioni di manipolazione
- ☐ Gestire le eccezioni durante le operazioni di connessione e di accesso al database
- ☐ Rappresentare le interrogazioni SQL
- ☐ Utilizzare i parametri nelle query
- ☐ Visualizzare le informazioni sulle tabelle e sui campi del database

**SOLUZIONI AI QUESITI DI AUTOVERIFICA p. 539**