JDBC di base

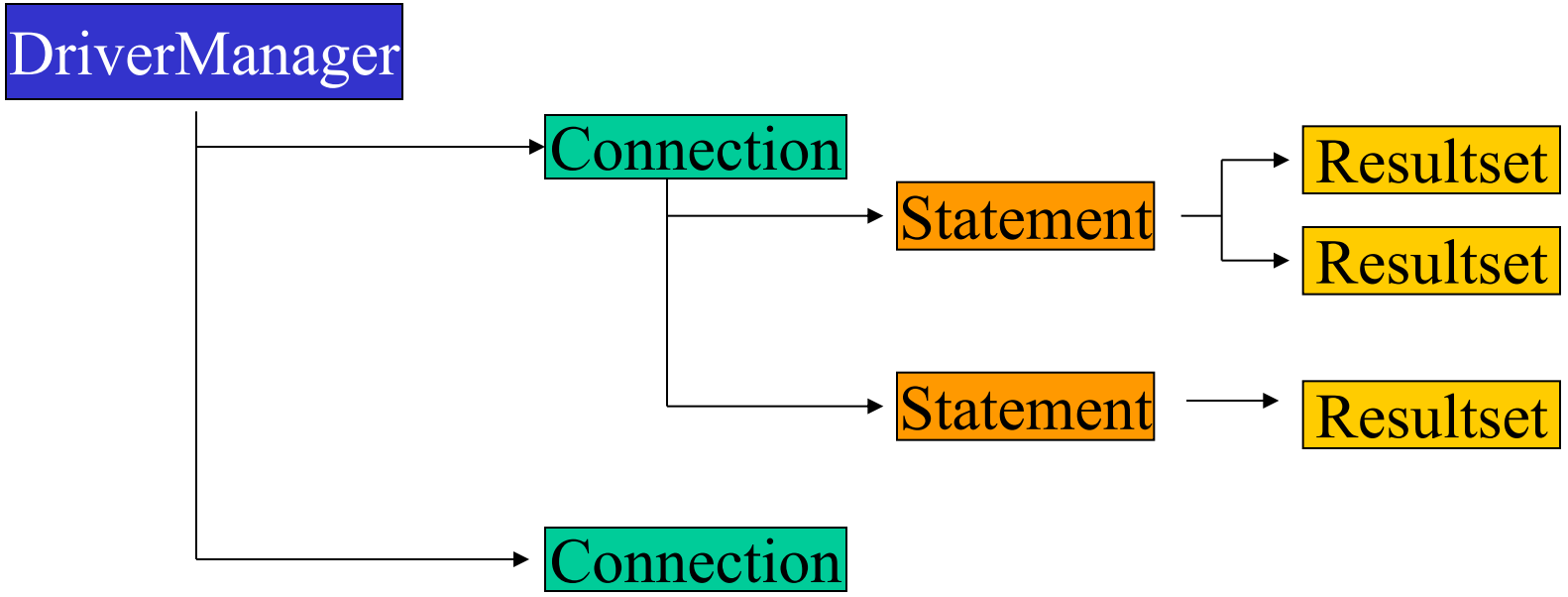
- Java Database Connectivity è il package Java per l'accesso a database relazionali
- il package contiene interfacce e classi astratte
- completa indipendenza del codice dal tipo di database o di DBMS a cui si intende accedere
  - ottenuto grazie al paradigma obiect-oriented (separazione dell'interfaccia dall'implementazione tramite il concetto di driver di accesso al database)
- JDBC, a differenza di ODBC (Open DataBase Connectivity), può essere usato solo con il linguaggio Java

Le classi/interfacce principali di JDBC

- DriverManager: fornisce l'interfaccia tra Java ed il driver vero e proprio del database
  - uno dei compiti è quello della creazione della connessione al db
- Connection: è la classe che si occupa di mantenere la connessione verso il database
- Statement: è la classe che si occupa di eseguire i comandi SQL
- ResultSet: è la classe che contiene il risultato di una query eseguita sul database

Schema dei legami tra le classi principali

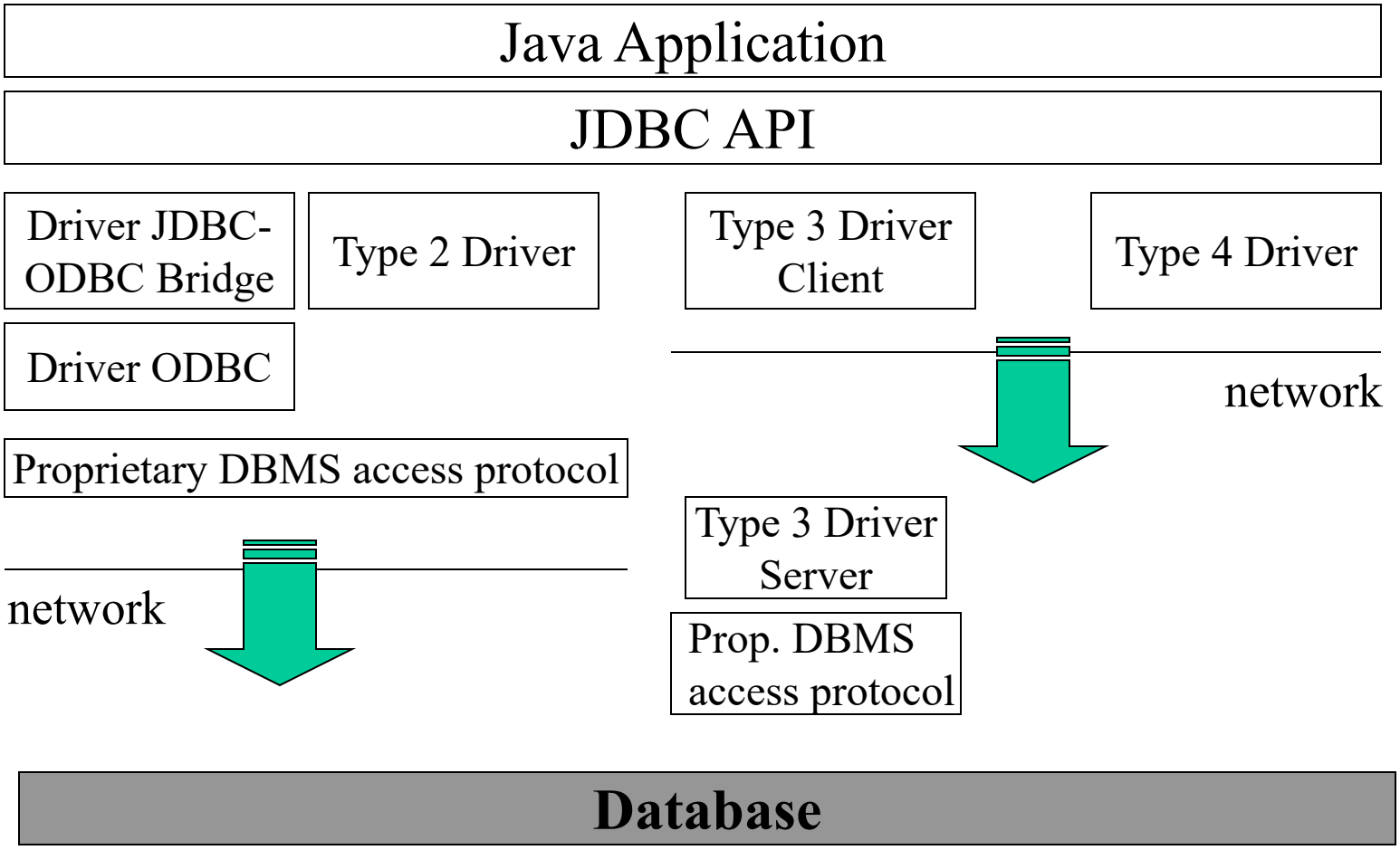
- Il DriverManager può creare una o più oggetti Connection per l'accesso ad uno stesso db o db distinti (remoti, eterogenei)
- un oggetto Connection può creare diversi oggetti Statement
- un oggetto Statement può dar luogo a più oggetti Resultset



Driver JDBC

- un driver JDBC è un componente software che **implementa** le interfacce e classi astratte del package JDBC
- driver JDBC di database/DBMS diversi implementano in modo diverso il package JDBC rispettandone però le specifiche
  - e.g. per accedere ad Oracle si usa un driver diverso da quello necessario per accedere a SQL Server o Informix
- Il driver generalmente è fornito dallo stesso produttore del database/DBMS
- Vedremo successivamente che, indipendentemente dai produttori, esistono 4 tipologie di driver

Architettura dei tipi di driver JDBC



Caricamento del driver

- il driver viene caricato a tempo di esecuzione, tramite il proprio nome nel modo seguente:
  - Class.forName("<nome del driver>");
  - è possibile caricare più driver;
- questa istruzione può produrre un'eccezione di classe ClassNotFoundException nel caso in cui il driver (la classe) non sia rintracciabile
  - generalmente questo capita quando la classe non è presente nel classpath dell'applicazione.
- ovviamente Class.forName(..) permette di caricare classi in generale e non solo driver JDBC

Exception nel caricamento del driver

- L'eccezione: ClassNotFoundException deve necessariamente essere gestita, pena la non compilazione dell'applicazione
- Il codice sarà quindi, in realtà, come il seguente:

```
try {
        Class.forName("<nome del driver>");
} catch (ClassNotFoundException cnfe) {
        // codice di gestione dell'eccezione
}
```

JDBC-ODBC Bridge (i)

- è un tipo di driver adatto agli ambienti Microsoft (ma non solo)
  - è un adattatore fra JDBC ed ODBC (Open Database Connectivity)
- Il nome del driver (della classe) di JDBC-ODBC Bridge è sun.jdbc.odbc.JdbcOdbcDriver
  - esso riceve le richieste dall'applicazione Java e le inoltra al driver ODBC traducendole opportunamente
  - il driver ODBC inoltra le richieste al database/DBMS
  - il driver ODBC restituisce i risultati ottenuti al driver JDBC che li ritrasmette all'applicazione Java

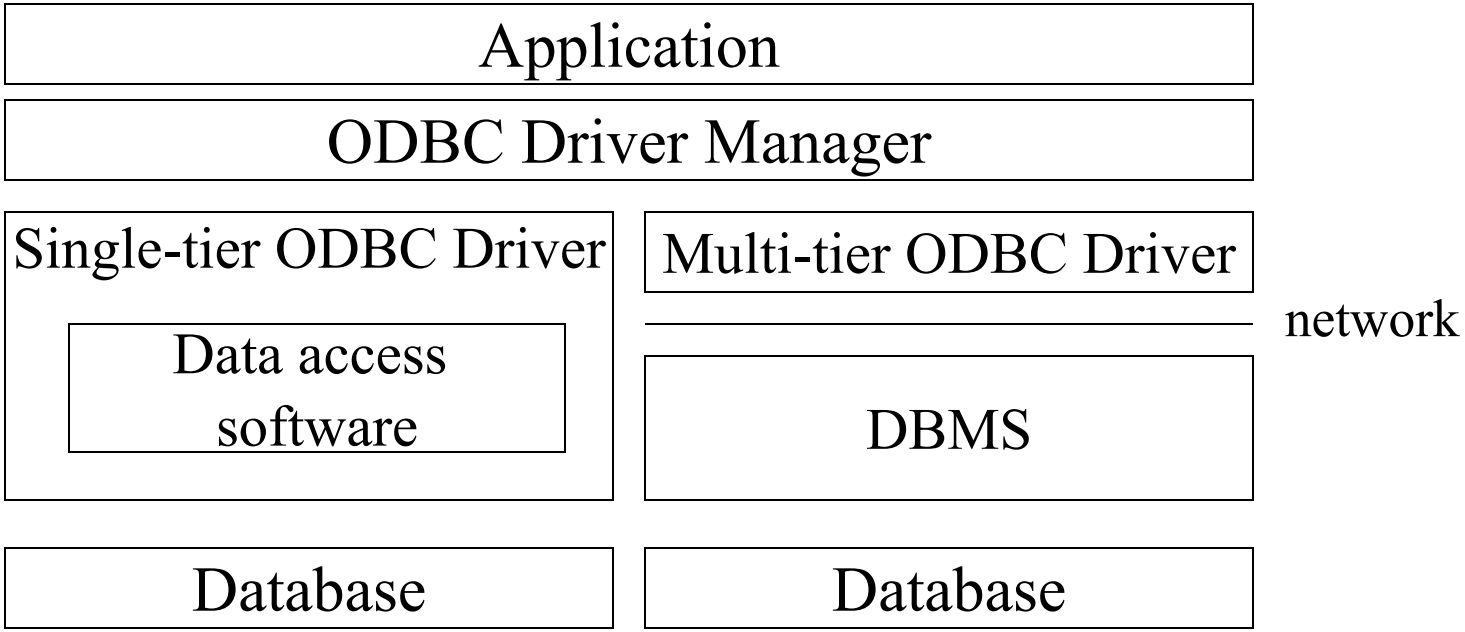
JDBC-ODBC Bridge (ii)

- questo driver è principalmente utilizzato per sviluppo e test poiché ha dei limiti:
  - non garantisce la gestione della concorrenza sulla stessa connessione
  - incompleta gestione di tutti i possibili tipi di dato
  - meno efficiente degli altri tipi di driver
  - controllo limitato sulle transazioni
- Il codice per caricare questo driver è il seguente: Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);

ODBC

- Per impiegare il driver JDBC-ODBC Bridge è necessario utilizzare anche ODBC
- ODBC, analogamente a JDBC, è uno standard per l'accesso a database relazionali che offre un insieme di *funzioni/procedure* astratte invocabili da diversi linguaggi di programmazione
- In generale ogni produttore di database file o DBMS offre anche il driver ODBC che implementa le relative funzioni astratte:
  - Ms Access, DBIV, Excel, Text ... (database files)
  - Oracle, SQL Server, Informix, DB2, …. (DBMS)

Tipi di driver ODBC: Single-tier e Multi-tier



- Esegue sia le funzioni ODBC che gli statement SQL => include il software di accesso al db
- Esempi: Dbase Files,Ms Access

- Esegue le funzioni ODBC, ma invia gli statement SQL al DBMS => non include il software di accesso
- Esempi: Oracle, Informix, SQL Server

Datasource ODBC

- Il driver ODBC accede al db attraverso un data source name (DSN)
- Un data source ODBC è un nome che definisce un riferimento ad un database file o a un database gestito con DBMS
- In entrambi i casi il DSN richiede che almeno venga specificato:
  - quale driver ODBC di quale db/DBMS deve essere usato
  - dove si trova il database (e.g. in locale, o in rete per i DBMS)

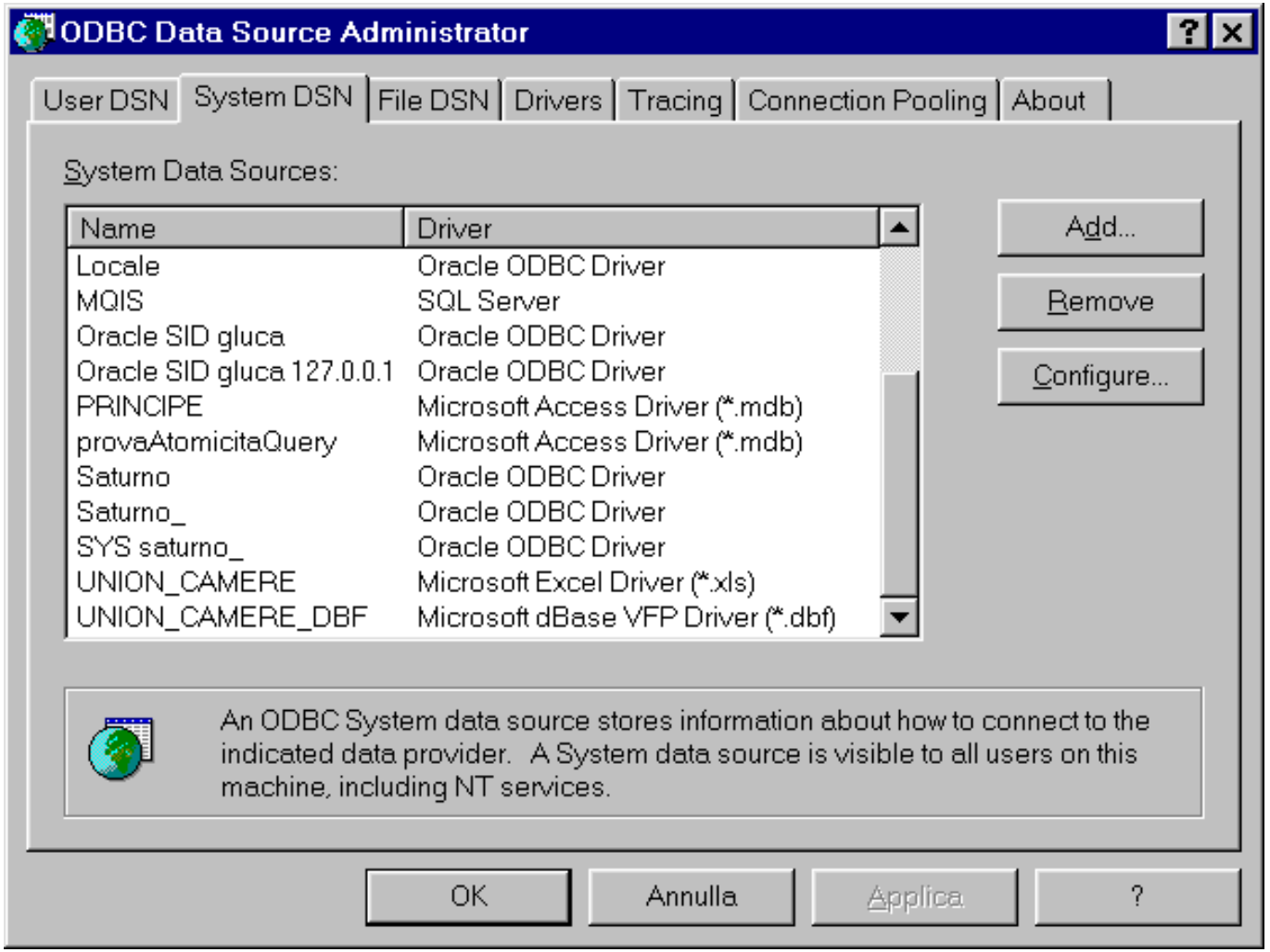
Tipi di data source names

- Esistono tre tipi di DSN:
  - DSN Utente: è un DSN utilizzabile dal solo utente che lo ha creato; eseguendo il login come utente diverso il DSN è inutilizzabile;
  - DSN di Sistema: è un DSN accessibile da tutti gli utenti della macchina su cui è stato creato
  - DSN su File: è un DSN le cui informazioni vengono custodite su un file e non nel registry; è la scelta migliore per applicazioni con installazione automatica

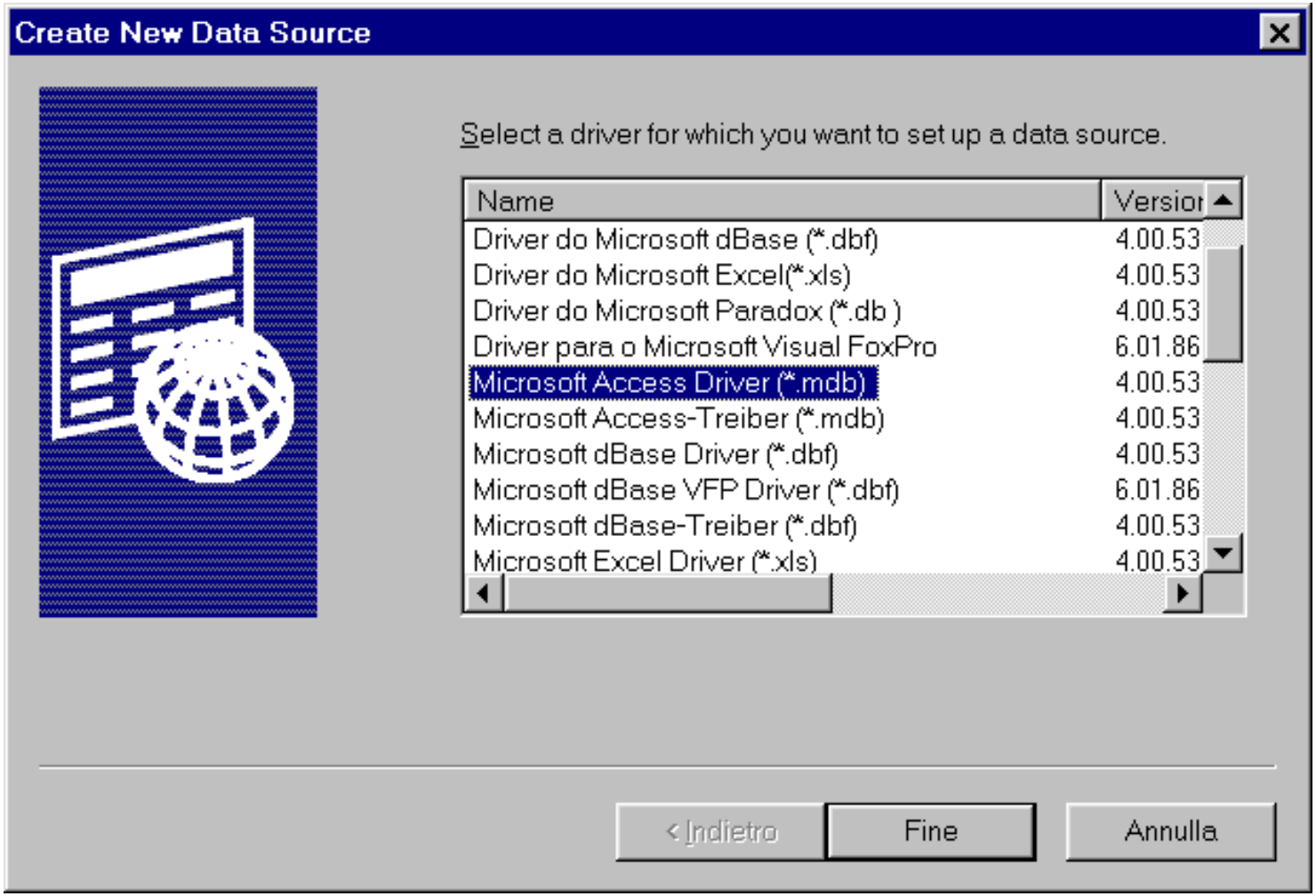
Configurazione di un DSN ODBC

- nel seguito, verranno mostrati i passi necessari per la configurazione di un DSN di sistema per l'accesso ad un database Ms Access nella stessa macchina del DSN
- il wizard che permette di configurare il DSN è strettamente dipendente dal driver utilizzato
  - quindi esso cambia a seconda della versione del driver, del tipo e versione di DBMS, nonché del produttore del driver

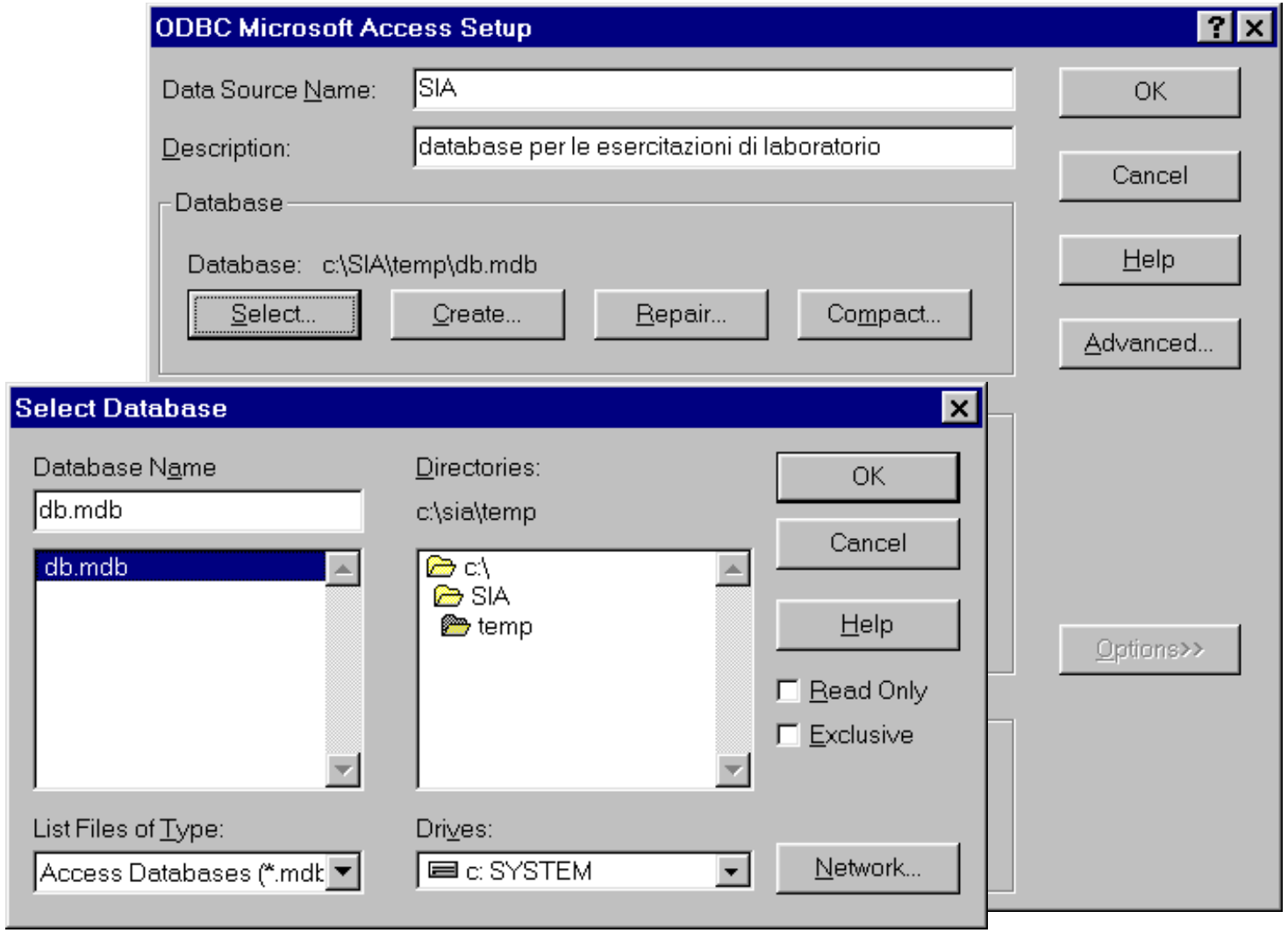
Aggiunta di un nuovo DSN



Selezione del driver ODBC



Definizione del DSN



Connessione (i)

- Dopo aver caricato il driver l'applicazione Java può effettuare la connessione gestita tramite la classe Connection
- L'apertura (i.e. istanziazione) di una connessione richiede la specifica del riferimento URL del database
  - cioè della sua posizione/indirizzo e nome (e.g. DSN)
  - nonché dello username e della password (se necessari)
- Ogni connessione (oggetto Connection) viene creata dal DriverManager

Connessione (ii)

- il codice necessario per creare una connessione è il seguente:

  Connection con =

  DriverManager.getConnection(“<db URL>”, “<user>”, “<password>”);

- anche questo metodo può produrre un’eccezione che deve essere obbligatoriamente gestita
  - l’eccezione da gestire nel solito blocco try-catch è di tipo SQLException

Connessione: esempio con JDBC-ODBC

- Il codice seguente crea una connessione via JDBC-ODBC al DSN di nome SIA

  Connection con = DriverManager.getConnection("jdbc:odbc:SIA");

  – username e password non stati specificati perché non necessari in questo caso
- la sintassi dell'URL dipende dal driver utilizzato:
  – in questo caso, non essendo possibile specificare una macchina remota, non appare alcun riferimento ad indirizzi IP o a nomi di macchine, come invece accade per i driver JDBC per i DBMS

Classe Statement (i)

- La classe Statement istanzia oggetti in grado di inoltrare istruzioni SQL al database cui l'applicazione è collegata
- In linea di principio è possibile eseguire tutte le istruzioni di SQL:
  - per la definizione dei dati (Data Definition Language)
  - per la manipolazione dei dati (DML), query, aggiornamenti, cancellazioni, inserimenti di tuple
  - per il controllo (DCL)

Classe Statement (ii)

- gli oggetti Statement vengono creati dall'oggetto connection come segue:

  Statement stmt = con.createStatement();

- L'esecuzione di questo metodo può produrre un'eccezione:
  - si tratta di un'eccezione di tipo SQLException da gestire con un usuale blocco try-catch

Metodi della Classe Statement

- I due metodi più importanti definiti nella classe Statement sono:
  - ResultSet executeQuery(String sql):
    esegue una query SQL restituendo un oggetto ResultSet;
  - int executeUpdate(String sql):
    esegue uno statement SQL di modifica dati (INSERT, UPDATE, DELETE) restituendo il numero di record coinvolti
- entrambi i metodi possono produrre eccezioni di tipo SQLException da gestire con try-catch
  - causata ad esempio da errori nello statement SQL

Esempi d'uso di executeUpdate

- stmt.executeUpdate("insert into persone(nome, cognome)
  values ('Giuseppe', 'Verdi')");

- stmt.executeUpdate("update persone
  set nome='Giuseppe'
  where cognome='Verdi'");

- stmt.executeUpdate("delete
  from persone
  where cognome='Verdi'");

Classe ResultSet

- Gli oggetti di questa classe contengono il risultato di query (quindi tuple) accessibile con i metodi:
  - boolean next(): muove il cursore sulla tupla successiva, restituendo true se essa esiste o false altrimenti; la posizione iniziale del cursore è prima del primo record;
  - <tipo> get<tipo>(int colonna): recupera il valore dell'attributo di posizione colonna della tupla corrente; <tipo> può essere String, Int, Date, BigDecimal etc. (in generale i tipi di dati previsti nelle specifiche JDBC);
  - <tipo> get<tipo>(String attributo): recupera il valore di un attributo della tupla corrente specificandone il nome
- l'invocazione di questi metodi deve essere inserita all'interno del blocco try-catch perché possono produrre eccezioni SQL Exception

ResultSet: Esempio

Esegue una query e visualizza il suo risultato:

```
import java. sql.*; // importazione package JDBC
class Esempio {
    public static void main( String argv[]) {
      try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection("jdbc:odbc:SIA");
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM studenti");
            while (rs.next()) {
            for (int i=1; i<=10; i++)
                  System.out.print(rs.getString(i) + " ");
            System.out.println;
      } // try
          catch( Exception e ) {e. printStackTrace();}
    } // main
} // classe
```

Metodi close()

- Le classi precedenti forniscono il metodo close() per rilasciare le risorse impegnate
- Ad esempio per ogni oggetto Statement creato viene allocata della memoria sia nello spazio di indirizzamento dell'applicazione che del database/DBMS
  - e.g. Oracle restituisce errore al 50-esimo Statement *aperto* indipendentemente dall'inoltro di query
- E' buona norma limitare il numero di oggetti Statement riusando quelli già esistenti e chiudendoli opportunamente:
  - rs.close(); stmt.close(); con.close();
- SQL Exception => usare try-catch

Driver JDBC per DBMS: l'Esempio di Oracle

- Tra i vari driver Oracle fornisce un driver JDBC di tipo 2 ed uno di tipo 4 (vedi architettura prec.):
  - JDBC OCI client-side: driver JDBC basato sulla libreria nativa Oracle OCI, necessita dell'installazione client di Oracle (**Tipo 2**);
  - JDBC THIN client-side: driver JDBC che non utilizza librerie native, non necessita dell'installazione client di Oracle **(Tipo 4)**;
- Oracle fornisce anche driver ODBC, pertanto è possibile l'accesso mediante il tipo 1

Riferimenti

- http://www.javasoft.com/jdbc
  - non solo la documentazione ufficiale su JDBC ma anche tutorial gratuiti
- http://industry.java.sun.com/products/jdbc/drivers
  - per cercare driver jdbc
- Giuseppe Naccarato, Java database e programmazione client/server, Apogeo, ISBN 88-7303-773-9, Febbraio 2001
  - contiene anche un'introduzione su basi di dati relazionali e SQL
- White et al, JDBC™ API Tutorial and Reference: Universal Data Access for the Java™ 2 Platform, 2/e, Addison Wesley, ISBN 0-201-43328-1, 1999
- George Reese, Database Programming with JDBC and Java, Second Edition, O'Reilly, ISBN 1-56592-616-1, 2000