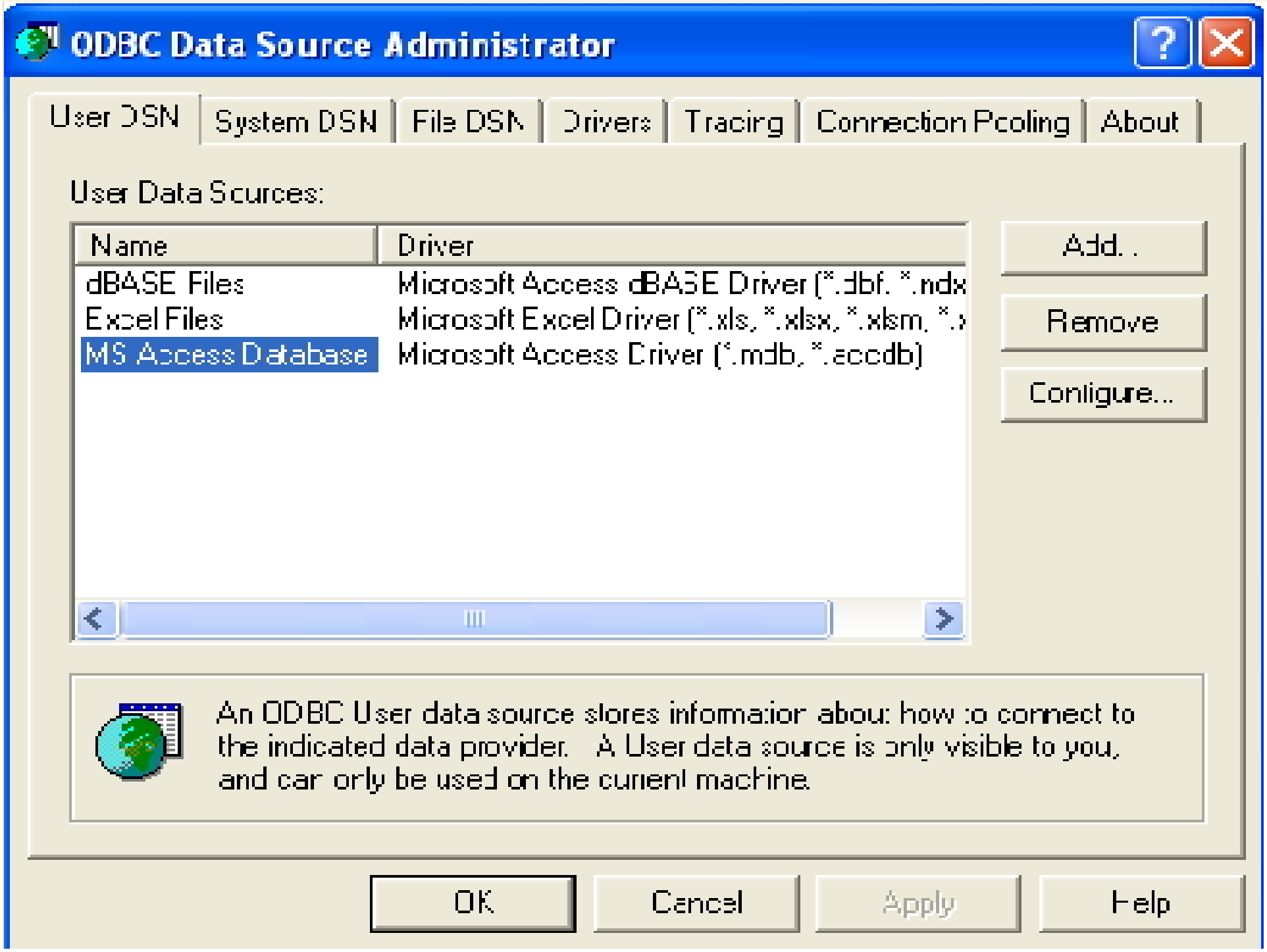
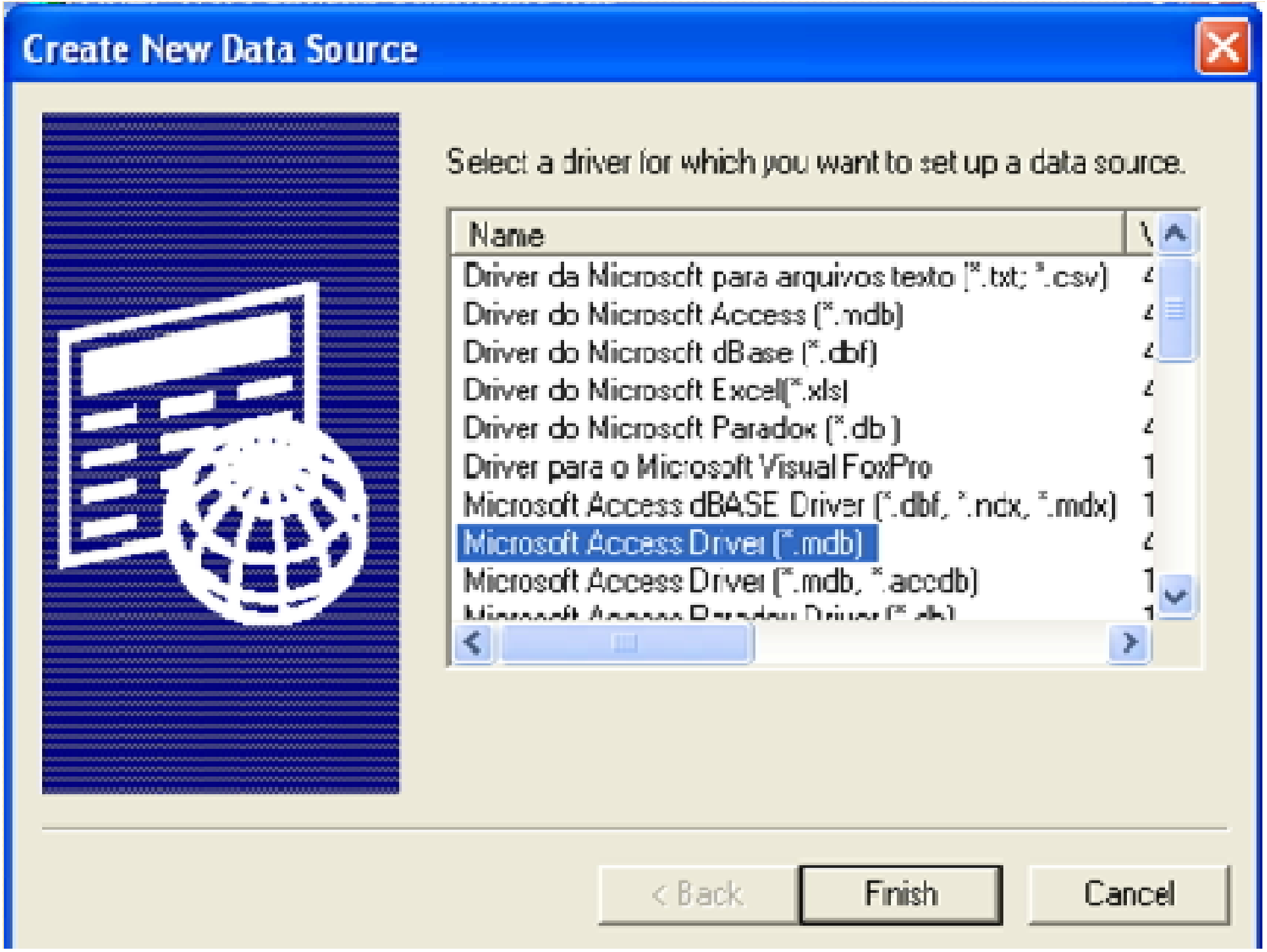
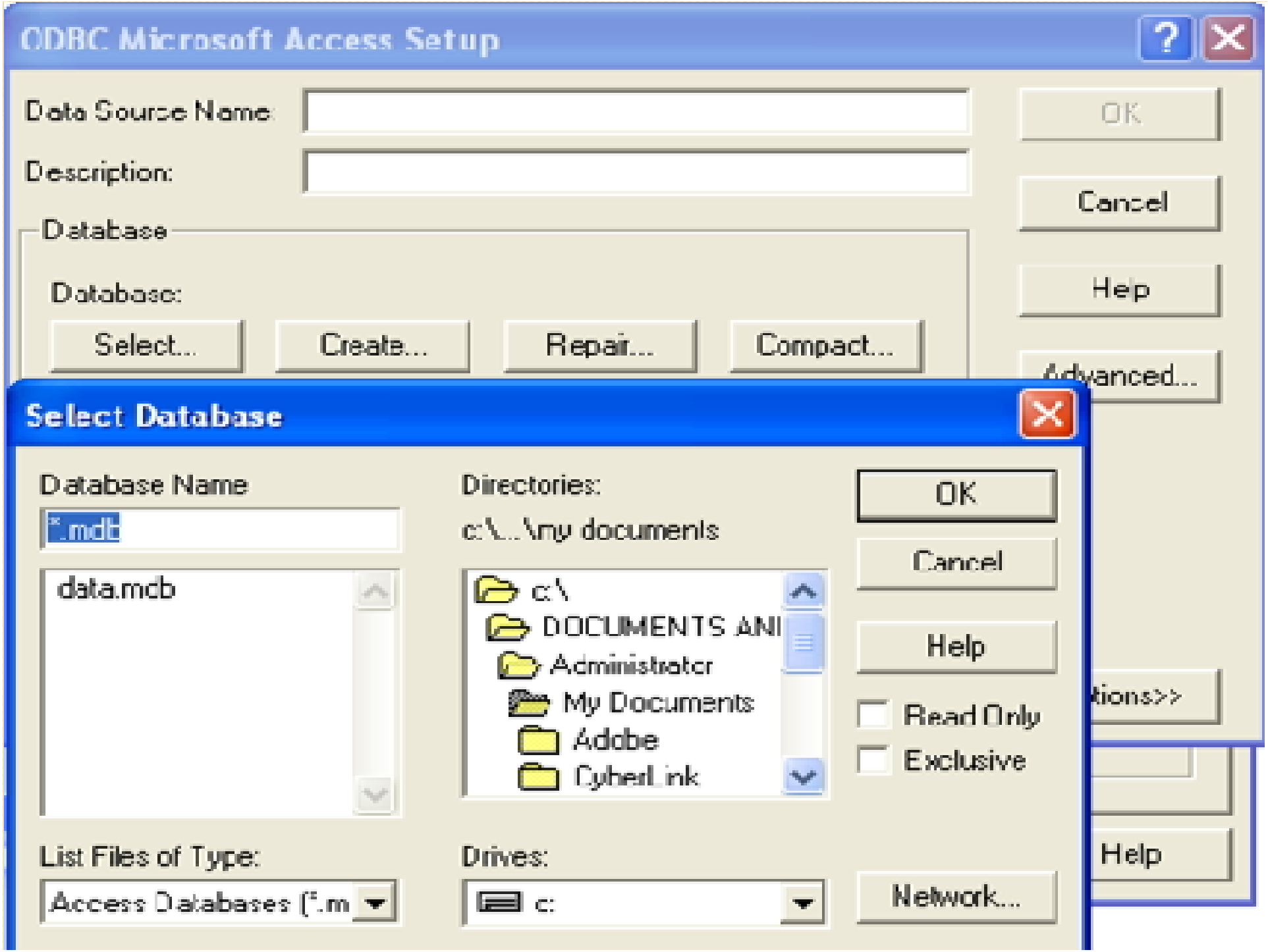
Course Name:
Advanced Java

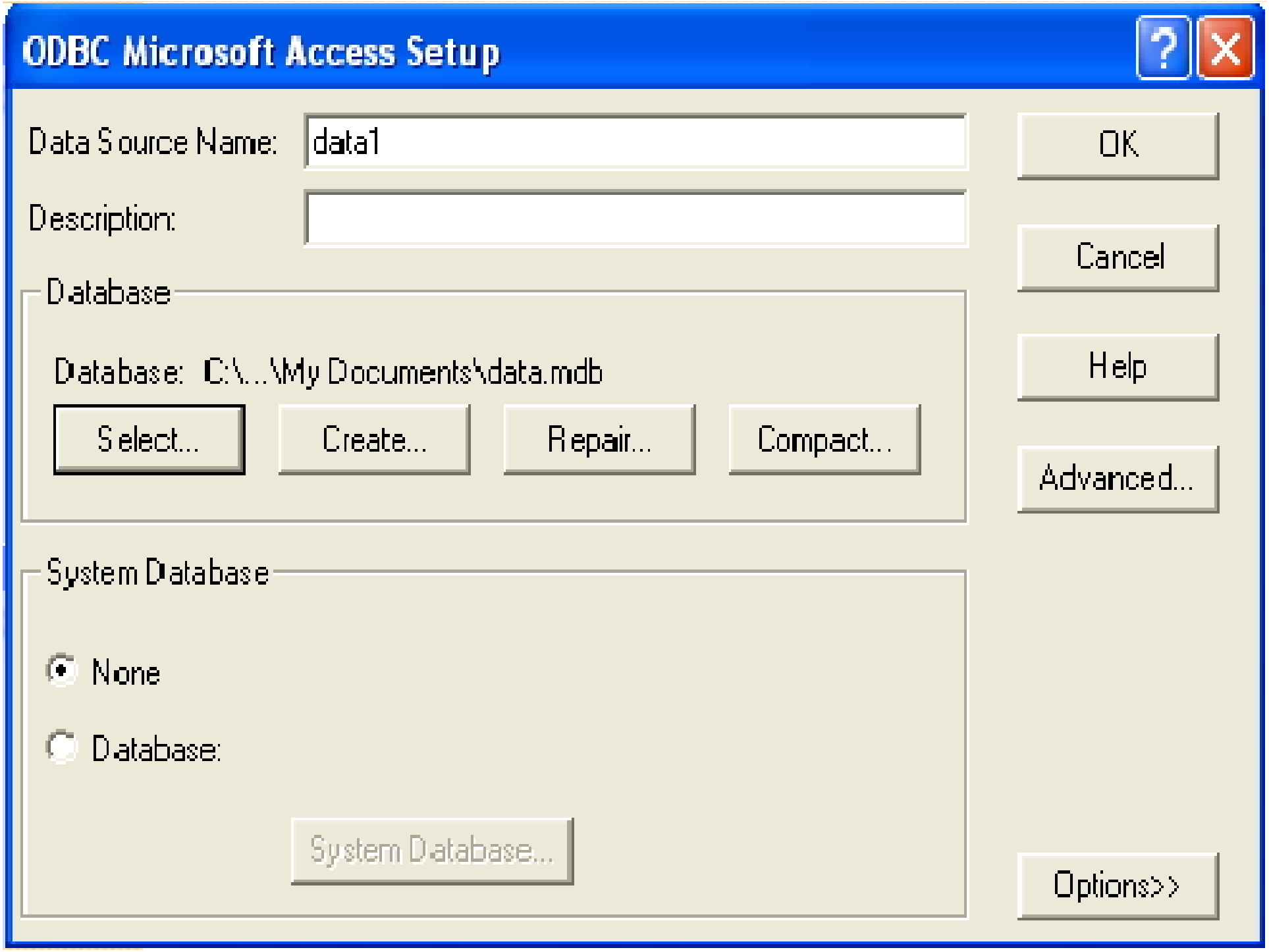
Lecture 14
Topics to be covered

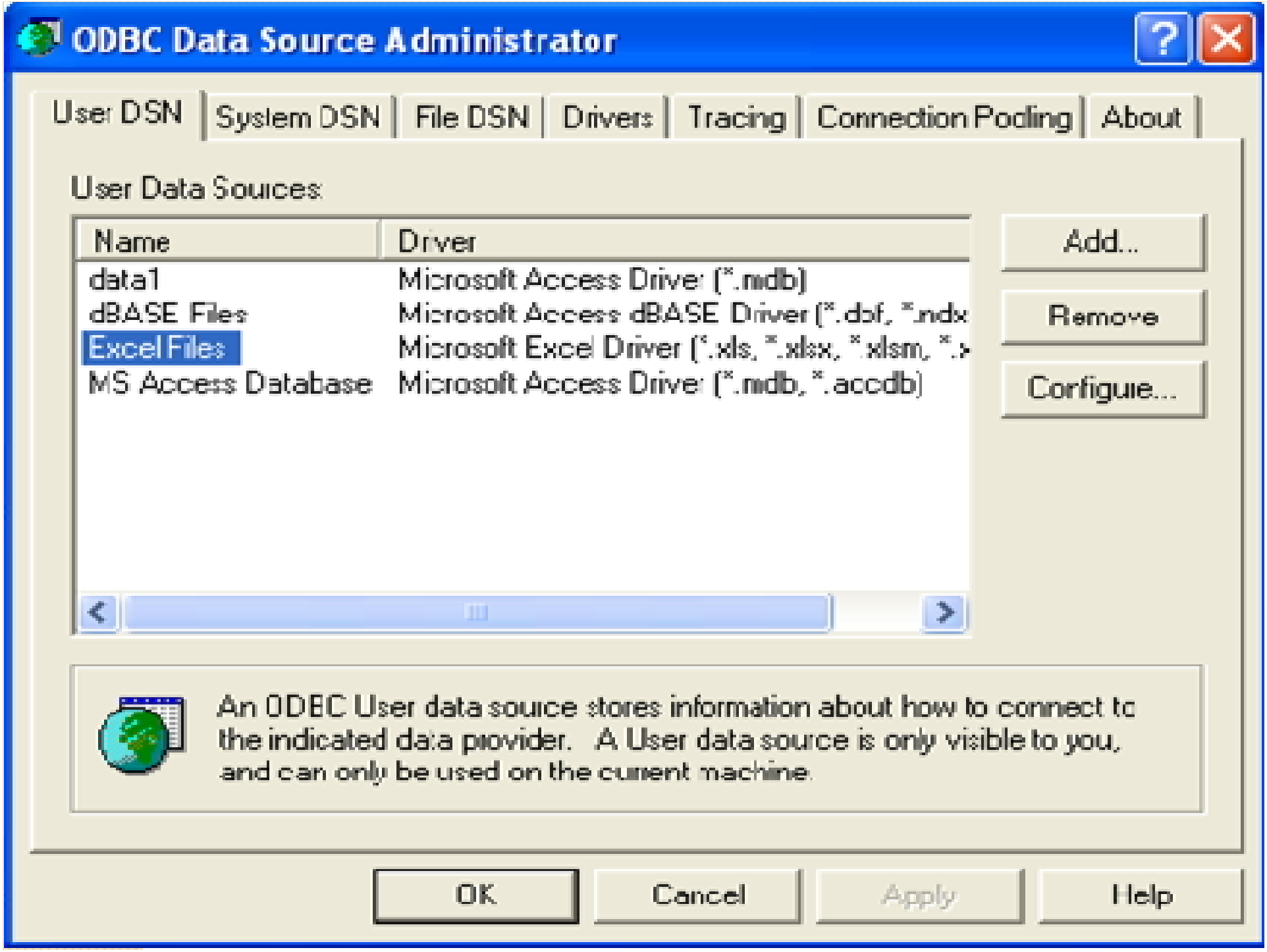
- JDBC Installation
- Transactions
- Metadata


ODBC Data Source Administrator
User DSN
System DSN
File DSN
Drivers
Tracing
Connection Pooling
About
User Data Sources:
Name
Driver
dBASE Files
Microsoft Access dBASE Driver (*.dbf, *.ndx
Excel Files
Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.
MS Access Database
Microsoft Access Driver (*.mdb, *.accdb)
Add...
Remove
Configure...
An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.
OK
Cancel
Apply
Help


Create New Data Source
Select a driver for which you want to set up a data source.
Name
Driver da Microsoft para arquivos texto (*.txt; *.csv)
Driver do Microsoft Access (*.mdb)
Driver do Microsoft dBase (*.dbf)
Driver do Microsoft Excel(*.xls)
Driver do Microsoft Paradox (*.db)
Driver para o Microsoft Visual FoxPro
Microsoft Access dBASE Driver (*.dbf, *.ndx, *.mdx)
Microsoft Access Driver (*.mdb)
Microsoft Access Driver (*.mdb, *.accdb)
< Back
Finish
Cancel


ODBC Microsoft Access Setup
Data Source Name:
Description:
Database
Database:
Select...
Create...
Repair...
Compact...
OK
Cancel
Help
Advanced...
Select Database
Database Name
*.mdb
data.mcb
Directories:
c:\...\my documents
c:\
DOCUMENTS AND
Administrator
My Documents
Adobe
CyberLink
OK
Cancel
Help
Read Only
Exclusive
List Files of Type:
Access Databases (*.m
Drives:
c:
Network...
Help


ODBC Microsoft Access Setup
Data Source Name:
data1
Description:
Database
Database: C:\...\My Documents\data.mdb
Select...
Create...
Repair...
Compact...
System Database
None
Database:
System Database...
OK
Cancel
Help
Advanced...
Options>>


ODBC Data Source Administrator
User DSN
System DSN
File DSN
Drivers
Tracing
Connection Pooling
About
User Data Sources:
Name
Driver
data1
Microsoft Access Drive (*.mdb)
dBASE Files
Microsoft Access dBASE Driver (*.dbf, *.ndx
Excel Files
Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.
MS Access Database
Microsoft Access Drive (*.mdb, *.accdb)
Add...
Remove
Configure...
An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.
OK
Cancel
Apply
Help

Steps in JDBC Connectivity:-

- Here are the **JDBC Steps** to be followed while writing JDBC program:
- Loading Driver
- Establishing Connection
- Executing Statements
- Getting Results
- Closing Database Connection

1. Loading Driver-

```
Connection con=null;
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

2. Establishing Connection –

```
con=DriverManager.getConnection("jdbc:odbc:table1"," "," ");
```

2. Executing Statements –

```
Statement statement=con.createStatement();
ResultSet rs=statement.executeQuery("select * from tab1");
```

3. Getting Results –

```
while(rs.next())
{
out.println("\n"+rs.getString("id")+"\t");
out.println("\n"+rs.getString("name")+"\t");
out.println("\n"+rs.getString("roll")+"\t");
out.println("\n"+rs.getString("price")+"/t");
}
```

4. Closing Database Connection -

```
rs.close();
```

JDBC Transactions

- A Transaction's ACID properties are:
  - **Atomic**: The entire set of actions must succeed or the set fails
  - **Consistent**: consistent state transfer from one state to the next
  - **Isolated**: A transaction is encapsulated and unmodifiable until the execution of the transaction set is *complete*
  - **Durable**: Changes committed through a transaction survive and tolerate system failures.
- Classic Example 1: Bank Transfer from one account to another
  - Step 1: withdrawal from Account A
  - Step 2: deposit into Account B

Using Transactions

- Step 1: turn off autocommit:
  - conn.setAutoCommit(false);
- Step 2: create and execute statements like normal
- Step 3: fish or cut bait: commit or rollback
  - if all succeeded:
    - conn.commit();
  - else, if one or more failed:
    - conn.rollback();
- Step 4 (Optional): turn autocommit back on:
  - conn.setAutoCommit(true);

Rolling Back Transactions

- When you get a SQLException, you are not told what part of the transaction succeeded and what part failed (this should be irrelevant)
- Best Practice:
  - *try* to rollback() (may throw new SQLException)
  - start over
- Example:
  - catch(SQLException e) {
    - try {
      - conn.rollback();
    - } catch (SQLException e) { checkPlease(); }
  - }

Transactions and Performance Implications

- Favor Transactions:
  - Disabling auto-commit means fewer commits over the wire (from driver to DBMS) which may cut down on IO overhead at the dataserver
- Favor Autocommit:
  - enabling autocommit may improve performance when multiple users are vying for database resources because locks are held for shorter periods of time
    - locks are only held per transaction. In autocommit mode, each statement is essentially a transaction
    - locks may be either page-level or row-level locks, the latter being more efficient (Oracle)

Transaction Isolation Modes

- TRANSACTION_NONE
  - Transactions are disabled or unsupported
- TRANSACTION_READ_UNCOMMITTED
  - Open policy that allows others to read uncommitted segments of a transaction, high potential for *dirty reads*
- TRANSACTION_READ_COMMITTED
  - Closed policy that disallows others' reading uncommitted segments. They must block until a commit is received, dirty reads are forbidden.
- TRANSACTION_REPEATABLE_READ
  - subsequent read transactions always get same set regardless of alteration until they call commit(), after which they get the changed data
- TRANSACTION_SERIALIZABLE
  - as above but also adds row insertion protection as well. If a transaction reads, and another transaction adds a row, and the first transaction reads again, it will get the original set without seeing the new row.

Conn.setTransactionIsolation(TRANSACTION_READ_COMMITTED)

Stored Procedures

- A Stored Procedure is written in a metalanguage defined by the DBMS vendor
- Used to batch or group multiple SQL statements that are stored in executable form at the database
- Written in some internal programming language of the DBMS:
  - Oracle's PL/SQL
  - Sybase's Transact-SQL
- THESE LANGUAGES ARE NON-PORTABLE from one DBMS to another (with the exception of the SQLJ standard, which allows you to write SQL in standard Java and have that understood by any DBMS that supports the SQLJ standard).

Incompatibilities

- Oracle Example:
  - CREATE PROCEDURE sp_select_min_bal
    @balance IN FLOAT,
    AS
    SELECT account_id
    WHERE balance > @balance
- Sybase Example:
  - create proc sp_select_min_bal
    (@balance real)
    as
    select account_id
    where balance > @balance
    return

Why Use Stored Procedures?

- Faster Execution of SQL (compiled and in-memory stored query plan)
- Reduced Network Traffic
- Modular Programming
- Automation of complex or sensitive transactions
- Syntax checking at time of creation of SP
- Syntax supports if, else, while loops, goto, local variables, etc., all of which dynamic SQL doesn't have

Using Stored Procedures

- Create a CallableStatement (using prepareCall which is similar to prepareStatement)
  - CallableStatement stmt =
    - conn.prepareCall("{call sp_setBalance(?,?)}"
    - stmt.registerOutParameter(2, Types.FLOAT);
    - stmt.setInt(1, custID);
    - stmt.setFloat(2, 213432.625);
    - stmt.execute();
    - Float newBalance = stmt.getFloat(2);
  - Always register OUT or INOUT parameters in stored procedures using registerOutParameter()

Using the JDBC MetaData Interface

- ResultSet: ResultSetMetaData getMetaData()
- ResultSetMetaData provides information about the types and properties of the DDL properties of a ResultSet object
- ResultSetMetaData provides various methods for finding out information about the structure of a ResultSet:
  - getColumnClassName(int col): gets fully-qualified Java class name to which a column value will be mapped; eg. Java.lang.Integer, etc.
  - getColumnCount(): gets the number of columns in the ResultSet
  - getColumnDisplaySize(int col): gets the normal maximum width in characters for column
  - getColumnName(int col): gets the name of column
  - int getColumnType(int col): gets the JDBC type (java.sql.Types) for the value stored in col; eg. Value 12 = JDBC VARCHAR, etc.
  - getPrecision(int col): for numbers, gets the mantissa length, for others, gets the number of bytes for column

JDBC – Metadata from RS

```
public static void printRS(ResultSet rs) throws SQLException
{
    ResultSetMetaData md = rs.getMetaData();
    // get number of columns
    int nCols = md.getColumnCount();
    // print column names
    for(int i=1; i < nCols; ++i)
        System.out.print( md.getColumnName( i)+",");
     / / output resultset
    while ( rs.next() )
    {   for(int i=1; i < nCols; ++i)
            System.out.print( rs.getString( i)+",");
        System.out.println( rs.getString(nCols) );
    }
}
```