WORKING WITH DATABASES IN JAVA

Asst.Prof.Eng. Ioan-Gheorghe Ratiu, PhD
Asst.Prof. Nicoleta David

“George Barțiu” University, Brașov

Abstract

A database represents a modality of storing some information (data) on external support, with the possibility of recapturing them. A database is stored into one or more folders, which are manipulated using the database management systems (DBMS). The most prevalent database model is the relational one, in which data is stored into tables. Besides tables, a relational database could contain: indexes, stored procedures, triggers, users and user groups, types of data, security and administration mechanisms for transactions, etc. Other database models are: the hierarchic model, the object-oriented model and the XML model.

1. General facts on databases

Applications that utilize databases are generally, complex applications used for the administration of larger-size information in a secure and efficient manner.

A database represents a method of storing certain information (data) on an external support, with the possibility of retrieving it. Usually, a database is stored in one or more folders that are being manipulated using database management systems (DBMS). The classic base-model of data is the relational one, in which data is stored into tables. Besides tables, a database can contain: indexes, stored procedures, triggers, users and user groups, data types, security and administration mechanisms for transactions etc.

Other database models can be hierarchic model, object-oriented model and XML model.

XML (eXtensible Markup Language) is a markup meta-language recommended by the Web Consortium for creating other markup languages (markup or tag - explicit interpretation action of a portion of text), like XHTML, RDF, RSS, MathML, SVG, OWL etc.

Those languages form the XML language family. XML Meta-language is a simplification of the SGML language (from which HTML also derives) and it was projected to support data transfers between applications on Internet. XML is now also a model of storing a data unstructured and semi-structured within XML native databases.

The most important database producers are: Oracle, Sybase, IBM,
Informix, Microsoft etc.

Creating a database is done by using specialized applications offered by the producer of a specific type of database.

Accessing a database is done by using a driver, specific to a certain type of database. This is responsible for the effective access to stored data while being the link between application and database.

2. The JDBC driver

JDBC (Java Database Connectivity) is a standard SQL interface for database access. JDBC consists of a set of classes and interfaces written in Java, which provide standard mechanisms for database applications designers.

The package that offers support for working with databases is called java.sql.

By using JDBC, it is easy to transmit SQL sequences to relational databases. In other words, it is not necessary to write a program in order to access an Oracle database, another program to access a Sybase database and so forth. It is enough to write a single program by using JDBC API and this will be capable to send SQL sequences to the desired database.

Of course, by writing the source-code in Java, the program portability is assured. Therefore, these are two powerful reasons that make the Java – JDBC combination worthy of being taken into account.

By being robust, secure, easy to use and easy to understand, Java is an excellent language for developing database applications. Everything it misses is only the means through which Java applications can communicate with databases. However, JDBC does provide this mechanism.

On the whole, JDBC accomplishes three things:
– establishes a connection with a database;
– sends SQL sequences;
– processes the results.

3. Connecting to a database

The process of connecting to a database implies two operations:
– loading into memory a suitable driver;
– realizing a proper connection.

A connection (session) to a database represents a context through which SQL sequences are sent and results are received. Within an application, multiple simultaneous connections to different databases or to the same database can exist.

The classes and interfaces for realizing a connection are:
- **DriverManager** class, which deals with recording the drivers that will be used in application;
- **Driver** interface, which has to be implemented by any class describing a driver;
- **DriverPropertyInfo** class;
- **Connection** interface, which describes the objects that model a proper connection to database.

*Loading a driver into memory*

The first thing that must be done by an application in the connection process to a database is to load into memory the class that implements a driver necessary in communicating with respective database.

This can be realized by several means, as below:

- `DriverManager.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver());`
- `Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");`
- `System.setProperty("jdbc.drivers","sun.jdbc.odbc.JdbcOdbcDriver");`

*Specifying a database*

As soon as a JDBC driver has been loaded into memory with DriverManager, this can be used in establishing a connection to database.

Considering the fact that multiple drivers can be stored in memory, we must have the possibility to specify - besides database identifier - the driver that needs to be used, as well.

This can be accomplished by using a specific address, also named JDBC URL, having the following format:

`jdbc:sub-protocol:identificator_baza_de_date`

The field *sub-protocol* names the driver type that must be used for realizing connection and it can be: *odbc, oracle, sybase, db2*, and so on.

The *database identifier* is an indicator specific to each driver that specifies the database with which the application would interact.

According to driver’s type, this identifier can include the name of a host machine, a port number, name of a file or a directory, as follows:

- `jdbc:odbc:testdb;`
- `jdbc:oracle:thin@persistentjava.com:1521:testdb;`
- `jdbc:sybase:testdb;`
- `jdbc:db2:testdb;`

When receiving a JDBC URL, the DriverManager will go through a list of drivers stored in memory, until one of them recognizes the respective URL.
If none suitable exists, then an exception of SQLException type will be launched, bearing the message: “no suitable driver”.

**Realizing a connection**

The method used for realizing a connection is getConnection from the DriverManager class, and it can have multiple forms:

– Connection c = DriverManager.getConnection(url);
– Connection c = DriverManager.getConnection(url, username, password);
– Connection c = DriverManager.getConnection(url, dbproperies).

A connection will be used for:

– creating SQL sequences that will be used for querying or actualizing the database;
– finding of information attached to database (meta-data).

The Connection class assures support for controlling transactions from memory to database, through methods such as: commit, rollback, setAutoCommit.

4. **Effecting SQL sequence**

Once a connection is made with DriverManager.getConnection(), it is used a Connection object resulted for creating a Statements-type object, by which SQL sequences can be sent to database. The most common SQL commands are the ones used for:

– querying the database (SELECT);
– actualizing the database (INSERT, UPDATE, DELETE):
  
  Connection c = DriverManager.getConnection(url);
  Statement s = c.createStatement();
  ResultSet r = s.executeQuery("SELECT * FROM one_table ORDER BY one_column");
  s.executeUpdate("DELETE * FROM one_table").

The executeQuery method sends SQL queries to the database and receives answer into a ResultSet-type object.

5. **Obtaining and processing the results**

*The ResultSet interface:*

String query = "SELECT cod, name FROM cities ORDER BY name";
ResultSet r = s.executeQuery( query );
while (r.next()) {
    System.out.println (r.getString ("code") + "," + r.getString ("name");
}
The ResultSetMetaData interface:
```
ResultSet r = s.executeQuery("SELECT*FROM cities");
ResultSetMetaData rsmd = r.getMetaData();
System.out.println("Column: " + rsmd.getColumnCount());
```

**Conclusions**

Accessing a database is done by using a driver specific to the respective database type. This is responsible with the effective access at stored data, being the link between application and database. The process of connecting to a database implies two operations:

– loading into memory of a suitable driver;
– realizing a proper connection.

Accessing the database is done by using *JDBC (Java Database Connectivity)* - a standard SQL interface constituted of a set of classes and interfaces written in Java - which supplies standard mechanisms for database applications designers.

After connecting with `DriverManager.getConnection()`, it can be used a `Connection` object resulted to create a `Statements`-type object, by which SQL sequences can be sent to database.

**BIBLIOGRAPHY**