BUSINESS INTELLIGENCE LABORATORY

Data Access: Relational Data Bases
RDBMS data access

- Protocols and API
  - ODBC, OLE DB, ADO, ADO.NET, JDBC

- JDBC Programming
  - Java classes java.sql

Business Intelligence Lab
Connecting to a RDBMS

Connection protocol
- locate the RDBMS server
- open a connection
- user authentication

Querying
- query SQL
  - SELECT
  - UPDATE/INSERT/CREATE
- stored procedures
- prepared query SQL

Scan Result set
- scan row by row
- access result meta-data

Client
- ConnectionString
- OK

Server
- SQL query
- Result set

Business Intelligence Lab
Connection Standards

- **ODBC** - Open Database Connectivity
  - Windows: odbc
  - Linux: unixodbc, iodbc
  - Tabular Data

- **JDBC** – Java Database Connectivity
  - See later on

- **OLE DB** (Microsoft) – Object Linking and Embedding
  - Tabular data, XML, multi-dimensional data

- **ADO** (Microsoft) – ActiveX Data Objects
  - Object-oriented API on top of OLE DB

- **ADO.NET**
  - Evolution of ADO in the .NET framework
ODBC Open DataBase Connectivity
ODBC Demo

- Registering an ODBC data source
  - pubs on access
  - pubs on SQL Server (driver SQL Server)

- Data access
  - copy Access table to Excel

- Linked tables
  - Linking SQL Server Table from Access
OLE DB Demo

- Creating .udl data links
- Data access
  - accessing Access data from Excel
- Linked tables
  - accessing Excel data from Access
- OLE DB Drivers
  - By Microsoft
RDBMS data access

- Protocols and API
  - ODBC, OLE DB, ADO, ADO.NET, JDBC

- JDBC Programming
  - Java classes java.sql
JDBC is a library of API’s for the uniform access to relational databases

- It includes classes for:
  - Connecting to a database
  - Submitting SQL queries
  - Scanning the results of queries
  - Accessing meta-data on tables

- Versions
  - Latest: JDBC 4.3 – Java 7
  - We use features from: JDBC 2.0 – Java 3

- Java Packages
  - java.sql
  - javax.sql
JDBC Architecture

- **Type 1 Drivers**
  - Translate JDBC in ODBC
    - JDBC-ODBC Bridge

- **Type 2 Drivers**
  - Part in Java and part in native code

- **Type 3 Drivers**
  - Pure Java,
  - Protocol independent from the RDBMS

- **Type 4 Drivers (the most portable)**
  - Pure Java,
  - Protocol specific/optimized for the RDBMS

since Java 8
JDBC: on-line resources

- **JDBC web site**

- **Java API**
  - [http://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html](http://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html)

- **Tutorial**
  - [http://docs.oracle.com/javase/tutorial/jdbc/](http://docs.oracle.com/javase/tutorial/jdbc/)
JDBC: Programming pattern

1. Register the JDBC driver
2. Connect to the RDBMS
3. Submit a SQL query
4. Process query results
5. Close the connection
import java.sql.*; // JDBC package

public int count(String[] args)
    throws ClassNotFoundException
{
    // MS Sql Server driver
    Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
    ...

    // alternative way (driver list separated by ':')
    System.setProperty("jdbc.drivers", "com.microsoft.sqlserver.jdbc.SQLServerDriver:
                          oracle.jdbc.OracleDriver");
JDBC: (1) Register the JDBC driver

- It consists of loading a Java class in the JVM

- Common drivers:
  - **SQL Server**
    - Microsoft `com.microsoft.sqlserver.jdbc.SQLServerDriver`
    - JTDS `net.sourceforge.jtds.jdbc.Driver`
  - **Oracle** `oracle.jdbc.OracleDriver`
  - **MySQL** `com.mysql.jdbc.Driver`
  - **Microsoft Access** `net.ucanaccess.jdbc.UcanaccessDriver`
  - **IBM DB2** `com.ibm.db2.jcc.DB2Driver`
JDBC: (1) Register the JDBC driver

- To locate the Java library with the driver:
  - running from commandline
    - `java -classpath <jar file>`
    - or, add `<jar file>` to the CLASSPATH
  - in Eclipse, add jar files in
    - Properties->Java Build Path -> Libraries

- Sample jar files:
  - SQL Server driver:
    - Microsoft: sqljdbcXX.jar
    - JTDS: jtdsXX.jar
  - Oracle driver:
    - ojdbcXX.jar
  - MySQL driver:
    - mysql-connector-java-XX-bin.jar
String url = “jdbc:sqlserver:”; + // driver
    “//apa.di.unipi.it/” + // host
    “;DatabaseName=pubs”; // database
String user = “foo”;  
String password = “hello”;  
Connection conn =
    DriverManager.getConnection(url, user, password);
...
JDBC: (2) Connect to the RDBMS

- The connection URL syntax depends on the driver
  - See driver documentation
  - SQL Server via JTDS
    - url = “jdbc:jtds:sqlserver://apa.di.unipi.it/pubs”
  - Oracle
    - url = “jdbc:oracle:thin:@apa.di.unipi.it:1521:PUBS”
  - MySQL
    - String url = “jdbc:mysql://apa.di.unipi.it/pubs”
  - MS Access
    - String url = “jdbc:ucanaccess://pubs.mdb”
// SQL statements processing object
Statement stmt = con.createStatement();

String query = "SELECT name, age FROM students";
// submit the SQL query, get result set
ResultSet rs = stmt.executeQuery(query);

String update = "UPDATE students SET age = age + 1";
// method for update/insert/create table
int affectedRows = stmt.executeUpdate(update);
JDBC: (4) Scan query results

// ResultSet is an iterator over the rows of the result
while ( rs.next() )
{
    // get field values by field name
    String nome = rs.getString("name");
    int age = rs.getInt("age");

    // get field values by field position (starts from 1!)
    int ageAgain = rs.getInt( 2 );

    System.out.println(nome + "," + age);
}

...
JDBC: (5) Close the connection

...  

// close the statement object
stmt.close();

// close connection to the database
conn.close();

...
The meta-data problem

The meta-data problem

How a value of a ‘numeric’ data type column in a MS Access DB is mapped into a Java variable?

Solution

(1) JDBC defines a set of common data types,

(2) MS Access driver maps ‘numeric’ column values to the common ‘REAL’ data type

(3) Java maps the common ‘REAL’ data type to ‘double’

(4) Java programmers call the getDouble() method to assign a program variable with the column value
The meta-data problem

<table>
<thead>
<tr>
<th>JDBC Type</th>
<th>Java Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>boolean</td>
</tr>
<tr>
<td>TINYINT</td>
<td>byte</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>short</td>
</tr>
<tr>
<td>INTEGER</td>
<td>int</td>
</tr>
<tr>
<td>BIGINT</td>
<td>long</td>
</tr>
<tr>
<td>REAL</td>
<td>float</td>
</tr>
<tr>
<td>FLOAT</td>
<td>double</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>double</td>
</tr>
<tr>
<td>BINARY</td>
<td>byte[]</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>String</td>
</tr>
<tr>
<td>LONGVARBINARY</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JDBC Type</th>
<th>Java Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMERIC</td>
<td>BigDecimal</td>
</tr>
<tr>
<td>DECIMAL</td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>java.sql.Date</td>
</tr>
<tr>
<td>TIME</td>
<td>java.sql.Timestamp</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>CLOB</td>
<td>Clob*</td>
</tr>
<tr>
<td>BLOB</td>
<td>Blob*</td>
</tr>
<tr>
<td>ARRAY</td>
<td>Array*</td>
</tr>
<tr>
<td>DISTINCT</td>
<td>mapping of underlying type</td>
</tr>
<tr>
<td>STRUCT</td>
<td>Struct*</td>
</tr>
<tr>
<td>REF</td>
<td>Ref*</td>
</tr>
<tr>
<td>JAVA_OBJECT</td>
<td>underlying Java class</td>
</tr>
</tbody>
</table>

*SQL3 data type supported in JDBC 2.0
Date, time, timestamp

- Package java.sql
  - Date, Time, Timestamp
    - Inherit from java.util.Date
    - Represent SQL data types:
      - DATE `getDay()`
      - TIME `getTime()`
      - TIMESTAMP `getTimeStamp()` it is DATE + TIME
- Date birth = resultSet.getDate(“BornDate”);
Date, time, timestamp: issues

- A plethora of formats for
  - Dates
  - Time ( 5:03:35.25 PM, 17:03:35.25 )
  - Timestamp (= Date + Time )

- Reasoning about
  - TimeZones
    - Fuse and summertime/standard time
  - Locale
    - Conventions on date format
  - Calendars
(Since Java 8) Package java.time

- Basic classes: LocalDate, LocalTime, LocalDateTime
- Utility classes:
  - DateTimeFormatter, Duration, Period, …

```java
date birth = resultSet.getDate("BornDate");
localDate birthdate = birth.toLocalDate();
datetimeformatter dform = datetimeformatter.ofPattern("dd/MMM/yyyy hh:mm:ss.S");
system.out.println(dform.format(birthdate));
```

- Tutorial on Java 8 time
public static void printRS(ResultSet rs) throws SQLException
{
   ResultSetMetaData md = rs.getMetaData();

   // output column names
   int nCols = md.getColumnCount();
   for(int i=1; i < nCols; ++i)
       System.out.print( md.getColumnName(i)+",");
   System.out.println( md.getColumnName(nCols));

   ...
....

// output resultset
while ( rs.next() )
{
    for(int i=1; i < nCols; ++i)
    {
        System.out.print( rs.getString(i)+" ",);
        System.out.println( rs.getString(nCols));
    }
}

Business Intelligence Lab
Meta-data on DB tables

...  
Connection con = .... ;  
DatabaseMetaData dbmd = con.getMetaData();  
String catalog = null;  // null = all  
String schema = null;  
String table = "sys%";  // table names starting with sys  
String[] types = null;  
ResultSet rs =  
    dbmd.getTables(catalog , schema , table , types );  
...

Business Intelligence Lab
**Problem**: read N rows from a CSV file, and insert each one into a database table

- **N SQL queries?**

  INSERT INTO names (id, name) VALUES (1, 'Luigi Rossi')
  INSERT INTO names (id, name) VALUES (2, 'Mario Bianchi')
  ...

  - Inefficiency: an execution plan has to be computed for every query, yet all of them share a common structure

- **Use ? as a placeholder for parameters**

  INSERT INTO names (id, name) VALUES (?, ?)
Prepared commands

```java
Connection con = .... ;
String query =
    "INSERT INTO names (id, name) VALUES (?, ?)";
PreparedStatement st = con.prepareStatement(query);

BufferedReader r = ... ;
int id = 1;
String name;
while( (name = r.readLine()) != null ) {
    st.setInt( 1, id++);
    st.setString( 2, name);
    st.executeUpdate();
}
```
JDBC: (4) Scrolling results

... 

```java
Statement stmt =
con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
```

String query = "SELECT name, age FROM students";

```java
ResultSet rs = stmt.executeQuery( query );
```

```java
rs.previous(); // previous row
rs.relative(-5); // 5 rows behind
rs.relative(7); // 7 rows ahead
rs.absolute(100); // 100th row
...
JDBC: (4) Updating results

...  

```
Statement stmt = con.createStatement(ResultSet.TYPE_FORWARD_ONLY,
                                        ResultSet.CONCUR_UPDATABLE);
```

String query = "SELECT name, age FROM students";

```
ResultSet rs = stmt.executeQuery(query);
```

```
while ( rs.next() )
{
  int age = rs.getInt("age");
  rs.updateInt("age", age+1);
  rs.updateRow();
}
```
Summary of java.sql classes
JDBC: Other features

- We will not discuss...
  - Transactions
  - Stored procedures
  - Batch UPDATE/INSERT
  - SQL embedded in Java (SQLJ)
  - Java Naming and Directory Interface (JNDI) URL repository
  - Java Persistent API (JPA)
    - Competitors: JDO, Hibernate, Toplink, …
  - Front ends for DB querying and administration
    - Competitors: Oracle SQL Developer, Squirrel, …
  - …
Exercise: Pivoting

- Pivoting a column \( A \) with distinct values \( V_1 \ldots V_k \) consists of yielding \( k \) binary columns \( A_1 \ldots A_k \) such that \( A_i = 1 \) iff \( A = V_i \)

<table>
<thead>
<tr>
<th>( A )</th>
<th>=&gt;</th>
<th>high</th>
<th>avg</th>
<th>low</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>low</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- Develop a Java program that given a database column yields in output an ARFF file obtained by pivoting the column.
Exercise: Stratified subsampling

- Let T be a database table (e.g., census), and A a column in T (e.g., sex)

- Develop a Java program that exports on a CSV file a subset of 30% of rows of T:
  - the subset is randomly chosen;
  - but it must preserve the proportion of distinct values of column A
    - e.g., if there are 65% of male students, the subset must contain 65% of males and 35% of females).
Intuition on the solution!

**Males**

- **Nrows** = 100
- **SelRow** = 30

1. **1° Rec** = 30/100
   - Selected
   - Not selected

2. **2° Rec** = 29/99
   - Selected
   - Not selected
   - Not selected

3. **3° Rec** = 28/98
   - Selected
   - Not selected
   - Not selected
   - …

4. **M° Rec** = 30/30
   - Not selected

All records selected!!!
How to generate an element with probability x/y?

- Generate a number n in the range [0 .... Y]
- The element is selected if n < x the record is selected
- For random selection of a number in the above range
  \((\text{int})(\text{Math.random()}*Y)\)