CHAPTER 4

ResultSet Metadata

The most important motive for work in school and in life is pleasure in work, pleasure in its result, and the knowledge of the value of the result to the community.

Albert Einstein

The main purpose of this chapter is to introduce you to the use of JDBC's result set metadata (the `java.sql.ResultSetMetaData` interface). `ResultSetMetaData` provides metadata for the `ResultSet` object. `ResultSetMetaData` enables us to get information about the types and properties of the columns in a `ResultSet` object. Typically, IDEs and GUI tools often use the JDBC `ResultSetMetaData` class to get information about data in a particular table. In order to help you understand this interface, I will list some important questions about metadata for `ResultSet` objects and then answer each one in detail. Before delving into `ResultSetMetaData`, we will look at the meaning of *metadata*. Note that `ResultSetMetaData` is an important interface in JDBC since RowSetMetaData extends this interface.

To answer questions about result set metadata, I'll provide a Java class called `jcb.meta.ResultSetMetaDataTool`. This class gives you ready-to-use `public static` methods for answering the questions listed in this chapter. For answering questions about result set metadata, I list portions of these classes at some sections, but the complete class definitions (including JavaDoc-style comments) is given in the Source Code section of the Apress website. All of the methods in this chapter are `static`, and each method is written to be as independent as possible. Whenever possible, the methods will return the result as XML (serialized as a `String` object, which can be easily converted to an XML document—`org.w3c.dom.Document`). Also, I'll provide a utility class, `DocumentManager`, which can

- Convert `org.w3c.dom.Document` to XML as a serialized `String` object
- Convert XML as a serialized `String` object into an XML document—`org.w3c.dom.Document`

In general, it is efficient to create XML as a serialized `String` object. The Java/JDBC solutions are grouped in a Java package called `jcb` (**J**DBC **C**ook **B**ook). The `jcb` package and all its associated classes will be available from the Apress website as well.

For some questions in this chapter, I will combine all of our answers into a single XML output. This can save lots of time and enables a client to get all of the required information regarding result set metadata with a single call (this way, you can improve performance of your database applications for metadata access).

4.1. What Is ResultSet Metadata?

What is *metadata*? Metadata is *data about data*, which provides structured descriptive information about other data. Result set metadata relieves the user of having to obtain full knowledge of the characteristics of relational databases in advance. Therefore, we conclude that metadata describes the data and its properties, but is not the actual data itself. For example, the records in a card catalog in a local library give brief details about the actual book. A record provides enough information to know what the book is called, its unique identification number, and how and where to find it. These details are metadata—in this case, bibliographic elements such as author, title, abstract, publisher, and published date.

In a nutshell, result set metadata enables dynamic discovery of a `ResultSet` object's structure. Typically, most JDBC programmers will be programming with knowledge of their target database's schema definitions (the names of tables, views, columns, and their associated types). In this case, programmers can use the strongly typed JDBC interfaces. However, there is also another important class of database access where an application (or an application builder) dynamically generates the result sets from ad hoc queries (submitted by different clients). This chapter describes the JDBC support for dynamic access.

`ResultSet` metadata has many applications. It can be used to

- Discover the number of columns, column names, types, and length
- Find out the table names for all columns in the result set
- Determine whether a given column is readable/writable
- Determine whether a given column is searchable (indicates whether the designated column can be used in a SQL `WHERE` clause or in SQL `SELECT`, `UPDATE`, and `DELETE` queries)

As we observe, metadata not only enables us to do effective management of resources, but also helps us to use metadata to find the data we need and determine how best to use it. Also, metadata provides structured descriptions of database information resources and services.

4.2. What Is a ResultSetMetaData Object?

JDBC's result set metadata, expressed as the `java.sql.ResultSetMetaData` interface, is a set of structured data that can be used to get information about the types and properties of the columns in a `ResultSet` object.

It is very easy to create a `ResultSetMetaData` object. The following code fragment creates a `ResultSet` object `rs`, and then uses `rs` to create a `ResultSetMetaData` object `rsMetadata`, and finally uses `rsMetadata` to find out how many columns `rs` has:

```
ResultSet rs = null;
Statement stmt = null;
Connection conn = null;
ResultSetMetaData rsmd = null;
String query = "SELECT id, name, photo FROM  employees";
```

```
try {
  conn = getConnection();
  stmt = conn.createStatement();
  rs = stmt.executeQuery(query);
  rsMetaData = rs.getMetaData();
  if (rsMetaData == null) {
     // ResultSetMetaData is not supported by db vendor
     // or there was a problem in getting the ResultSetMetaData.
     // In general, it is a good idea to check if ResultSetMetaData
     // is null or not.
     ...
  }
  else {
     // ResultSetMetaData is supported by db vendor
     int numberOfColumns = rsMetaData.getColumnCount();
  }
  …
}
catch(SQLException e) {
   // handle exception
}
finally {
   // close resources such as ResultSet, Statement, and Connection
}
```

If you do not know exactly the table structure (the schema) of the `ResultSet`, you can obtain it via a `ResultSetMetaData` object. The `ResultSetMetaData` interface answers the following questions:

- How many columns are in a result set (`ResultSet` object)?
- What is the name of a given column?
- What is the name of a table for a given column?
- Are the column names case sensitive?
- What is the data type of a specific column?
- What is the maximum character size of a column?
- Can you search on a given column?
- Is a given column readable/writable?

4.3. How Do You Create a ResultSetMetaData Object?

`ResultSetMetaData` is a Java class, which provides information about the `ResultSet` and implements the `java.sql.ResultSetMetaData` interface. The column name, column type, and so forth are provided by the `ResultSetMetaData` object. Database vendors provide different classes for implementing the `ResultSetMetaData` interface.

To learn about a given result set from a database (expressed as a `ResultSet` object), you must obtain (or create) a `ResultSetMetaData` object. Once a client program has obtained a valid `ResultSet`, the following code gets a metadata object:

```
ResultSet rs = null;
...
try {
   ...
   rs = <get a result set>;
   ResultSetMetaData rsMetaData = rs.getMetaData();
   if (rsMetaData == null) {
      // there is no metadata for a given result set
      ...
   }
   else {
     // once you are here, then you can execute methods
     // given in the ResultSetMetaData.

     // get the number of columns for a given result set
     int numberOfColumns = rsMetaData.getColumnCount();
   }
   ...
}
catch(SQLException e) {
   // handle the exception
}
finally {
   // close the rs (result set object)
}
```

The `ResultSetMetaData` interface has over two dozen methods. Using `ResultSetMetaData`, an application can discover the number of columns returned, as well as an individual column's suggested display size, column names, column types, and so on. Note that a given JDBC driver may not provide information for all of the methods, so you need to check returned objects for `null`s or empty strings.

A `ResultSet` object provides a table of data representing a database result set, which is usually generated by executing a statement that queries the database. The `ResultSet` object is one of the most important objects in the JDBC API. `ResultSet` is basically an abstraction of a table of general width and unknown length. Most of the methods and queries return result data as a `ResultSet`. A `ResultSet` contains any number of named columns that you can ask for by name. It also consists of one or more rows, which you can move through sequentially from top to bottom, one at a time. Before you can use a `ResultSet`, you need to inspect the number of columns it contains. This information is stored in the `ResultSetMetaData` object.

```
    //get the number of columns from the "ResultSet metadata"
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;
    ...
    try {
        // get a result set
        conn = ... get a java.sql.Connection object ...
        stmt = conn.createStatement();
        rs = stmt.executeQuery("SELECT id, name FROM employees");
        ResultSetMetaData rsmd = rs.getMetaData();
        if (rsmd == null) {
            // no result set metadata
            ...
        }
        else {
            int numberOfColumns = rsmd.getColumnCount();
            if (numberOfColumns != 2) {
              // error
            }
        }
    }
    catch(SQLException e) {
      // handle the exception
    }
    finally {
       // close the rs (result set object)
       // close the stmt (Statement object)
       // close the conn (connection object)
    }
```

When you obtain a `ResultSet`, it points just before the first row. To iterate all of the rows, you can use the `next()` method to obtain each additional row, and the method returns `false` when no more rows remain. Since fetching data from a database can result in errors, you must always enclose your result set manipulations in a `try/catch` block.

Here's an example:

```
import java.sql.*;
import jcb.util.DatabaseUtil;
...
public static Connection getConnection()
   throws Exception {
   // return a valid database connection object
}

...
```

```
ResultSet rs = null;
Statement stmt = null;
Connection conn = null;
try {
    conn = getConnection();
    stmt = conn.createStatement();
    rs = stmt.executeQuery("SELECT id, name FROM employees");
    ResultSetMetaData rsmd = rs.getMetaData();
    if (rsmd == null) {
        // ResultSetMetaData is not supported
    }
    else {
        int numberOfColumns = rsmd.getColumnCount();
        while(rs.next()) {
            for (i = 1; i <= numberOfColumns; i++) {
                System.out.println(rs.getString(i)+"     ");
            }
            System.out.println("------");
        }
    }
}
catch(Exception e) {
    // handle and deal with the exception
    System.out.println(e.getMessage());
}
finally {
    // close database resources
    DatabaseUtil.close(rs);
    DatabaseUtil.close(stmt);
    DatabaseUtil.close(conn);
}
```

4.4. How Does JDBC Define ResultSetMetaData?

In this section, I provide `ResultSetMetaData`'s definition, field summary, and method summary (the output is slightly modified). The J2SE 5.0 documentation defines `ResultSetMetaData` as "an object that can be used to get information about the types and properties of the columns in a `ResultSet` object." The `ResultSetMetaData` is an interface defined in the `java.sql` package:

```
package java.sql;
public interface ResultSetMetaData {
}
```

The following code fragment creates the `ResultSet` object `rs`, then creates the `ResultSetMetaData` object `rsmd` and uses `rsmd` to find out how many columns `rs` has and whether the first column in `rs` can be used in a `WHERE` clause:

```
ResultSet rs = null;
Statement stmt = null;
Connection conn = null;
ResultSetMetaData rsmd = null;
try {
   conn = getConnection(); // get a valid Connection object
   stmt = conn.createStatement();
   rs = stmt.executeQuery("SELECT column1, column2 FROM MY_TABLE");
   rsmd = rs.getMetaData();
   if (rsmd == null) {
      // ResultSetMetaData is not supported
      ...
   }
   else {
      // ResultSetMetaData is supported
      int numberOfColumns = rsmd.getColumnCount();
      boolean b = rsmd.isSearchable(1);
   }
}
catch(SQLException se) {
   // handle database exceptions
}
catch(Exception e) {
   // handle other exceptions
}
finally {
   // close ResultSet, Statement, Connection objects
}
```

Therefore, the `ResultSetMetaData` interface provides

- Information about the types and properties of the DDL properties of a `ResultSet` object
- Various methods for finding out information about the structure of a `ResultSet` object

ResultSetMetaData Field Summary

Table 4-1 shows the `ResultSetMetaData` field summary.

Table 4-1. *ResultSetMetaData Field Summary*

Type	Field	Summary
`static int`	`columnNoNulls`	The constant indicating that a column does not allow `NULL` values
`static int`	`columnNullable`	The constant indicating that a column allows `NULL` values
`static int`	`columnNullableUnknown`	The constant indicating that the nullability of a column's values is unknown

ResultSetMetaData Method Summary

Table 4-2 shows the `ResultSetMetaData` method summary.

Table 4-2. *ResultSetMetaData Method Summary*

Method Signature	Summary
`String getCatalogName(int column)`	Gets the designated column's table's catalog name
`String getColumnClassName(int column)`	Returns the fully qualified name of the Java class whose instances are manufactured if the method `ResultSet.getObject` is called to retrieve a value from the column
`int getColumnCount()`	Returns the number of columns in this `ResultSet` object
`int getColumnDisplaySize(int column)`	Indicates the designated column's normal maximum width in characters
`String getColumnLabel(int column)`	Gets the designated column's suggested title for use in printouts and displays
`String getColumnName(int column)`	Gets the designated column's name
`int getColumnType(int column)`	Retrieves the designated column's SQL type
`String getColumnTypeName(int column)`	Retrieves the designated column's database-specific type name
`int getPrecision(int column)`	Gets the designated column's number of decimal digits
`int getScale(int column)`	Gets the designated column's number of digits to right of the decimal point
`String getSchemaName(int column)`	Gets the designated column's table's schema
`String getTableName(int column)`	Gets the designated column's table name
`boolean isAutoIncrement(int column)`	Indicates whether the designated column is automatically numbered, thus read-only
`boolean isCaseSensitive(int column)`	Indicates whether a column's case matters
`boolean isCurrency(int column)`	Indicates whether the designated column is a cash value
`boolean isDefinitelyWritable(int column)`	Indicates whether a write on the designated column will definitely succeed
`int isNullable(int column)`	Indicates the nullability of values in the designated column
`boolean isReadOnly(int column)`	Indicates whether the designated column is definitely not writable
`boolean isSearchable(int column)`	Indicates whether the designated column can be used in a SQL `WHERE` clause
`boolean isSigned(int column)`	Indicates whether values in the designated column are signed numbers
`boolean isWritable(int column)`	Indicates whether it is possible for a write on the designated column to succeed

4.5. What Is the Weakness of the ResultSetMetaData Interface?

In distributed client-server applications, `ResultSetMetaData` might provide a minor performance problem. That is, for every bit of information, you have to call a specific method. There is no single method call to get most of the metadata from a `ResultSet` object. In order to eliminate this minor problem, server-side code should return the metadata result as an XML object (which represents `ResultSetMetaData` information in a structured fashion) rather than returning an instance of `ResultSetMetaData`. This way, the client can invoke a single method and get the desired answer from an XML document.

4.6. What Is the Relationship of ResultSetMetaData to Other Objects?

Before delving into the solution, let's take a look at the relationships (Figure 4-1) between important low-level interfaces and classes.

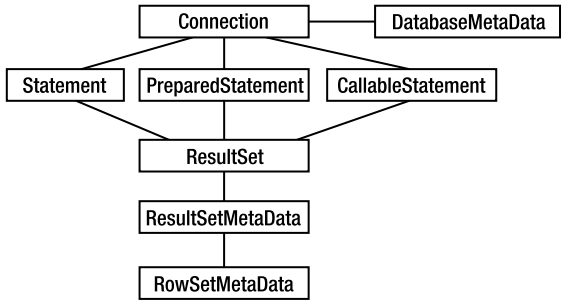

Figure 4-1. *Relationships between important low-level interfaces and classes*

To connect to a database, you use an instance of the `Connection` class. Then, to find out the names of the database tables and fields, you need to get an instance of the `DatabaseMetaData` class from the `Connection`. Next, to issue a query, you compose the SQL query string and use the `Connection` to create a `Statement` class. By executing the statement, you obtain a `ResultSet` class, and to find out the names of the column rows in that `ResultSet`, you must obtain an instance of the `ResultsetMetaData` object.

To obtain the `ResultSet` metadata, follow these steps:

1. To connect to a database, you use an instance of the `Connection` object.
2. To find out the names of the database schemas/tables/columns, you get an instance of the `DatabaseMetaData` object from the `Connection`.
3. To issue a query, you compose the SQL query string and use the `Connection` to create a `Statement`.
4. By executing the statement, you obtain a `ResultSet` object.
5. Finally, to find out the names of the column rows in that `ResultSet`, you obtain an instance of the `ResultsetMetaData` object.

What does ResultSetMetaData provide? ResultSetMetaData objects can be used to get information about the types and properties of the columns in a ResultSet object. ResultSetMetaData encapsulates the column's information—such as the number of columns, their names, and the column data type and its precision—contained in a ResultSet. ResultSetMetaData provides many methods to deal with the column's information; for example, the getColumnName() method of the ResultSetMetaData object returns the column's name in the ResultSet.

4.7. How Do You Express ResultSetMetaData in XML?

What follows is a single method, ResultSetMetaDataTool.getResultSetMetaData(), which creates an XML document for a ResultSetMetaData from a given ResultSet. The method getResultSetMetaData() can be used by any type of client (based on XML tags, clients may extract their needed information). The generated XML will have the following syntax:

```
<?xml version="1.0"?>
<resultSetMetaData columnCount="number-of-columns">
    <columnMetaData column="column-number"
                    columnDisplaySize="column-display-size"
                    columnLabel="column-label"
                    columnName="column-name"
                    columnType="column-type"
                    columnTypeName="column-type-name"
                    columnClassName="column-class-name"
                    tableName="table-name"
                    precision="precision-of-column"
                    scale="scale-of-column"
                    isAutoIncrement="true/false"
                    isCurrency="true/false"
                    isWritable="true/false"
                    isDefinitelyWritable="true/false"
                    isNullable="0/1"
                    isReadOnly="true/false"
                    isCaseSensitive="true/false"
                    isSearchable="true/false"
                    isSigned="true/false"
                    catalog="catalog-name"
                    schema="schema-name" />
    <columnMetaData ... />
    ...
    <columnMetaData ... />
</resultSetMetaData>
```

getResultSetMetaData()

The `getResultSetMetaData()` method accepts an instance of `ResultSet` and outputs its metadata as XML (expressed as a serialized `String` object):

```
/**
 * Gets column names and their associated attributes (type,
 * size, nullable). The result is returned as XML (as
 * a String object);  if table name is null/empty
 * it returns null.
 *
 * @param rs the result set (ResultSet) object.
 * @return result set's metadata as XML as String object;
 * this metadata includes column names and their associated
 * attributes: type, size, nullable.
 * @exception Failed to get the result set's metadata as XML.
 */
public static String getResultSetMetaData(ResultSet rs)
    throws Exception {

    if (rs == null ) {
        return null;
    }

    // Retrieves the number, types and properties
    // of this ResultSet object's columns.
    ResultSetMetaData rsMetaData = rs.getMetaData();
    if (rsMetaData == null ) {
        return null;
    }

    StringBuffer sb = new StringBuffer();
    sb.append("<resultSetMetaData columnCount=\"");
    int numberOfColumns = rsMetaData.getColumnCount();
    sb.append(numberOfColumns);
    sb.append("\">");

    for (int i=1; i<=numberOfColumns; i++) {
        sb.append(getColumnMetaData(rsMetaData, i));
    }

    sb.append("</resultSetMetaData>");
    return sb.toString();

}
```

getColumnMetaData() method:

```
    /**
     * Gets specific column's associated attributes
     * (type, size, nullable). The result is returned
     * as XML (represented as a String object).
     * XML attributes (as constants) are prefixed
     * with the "XML_METADATA_TAG_".
     *
     * @param rsMetaData the result set metadata object.
     * @param columnNumber the column number.
     * @return result set's metadata as XML as
     *  String object; this metadata includes
     *  column names and their associated attributes:
     *  type, size, nullable.
     * @exception Failed to get the result set's
     *  meta data as an XML.
     */
    private static String getColumnMetaData
            (ResultSetMetaData rsMetaData,
             int columnNumber)
        throws Exception {
        StringBuffer sb = new StringBuffer();
        sb.append("<columnMetaData ");
        append(sb, XML_METADATA_TAG_COLUMN, columnNumber);

        // Indicates the designated column's normal
        // maximum width in characters
        append(sb, XML_METADATA_TAG_COLUMN_DISPLAY_SIZE,
            rsMetaData.getColumnDisplaySize(columnNumber));

        // Gets the designated column's suggested title
        // for use in printouts and displays.
        append(sb, XML_METADATA_TAG_COLUMN_LABEL,
            rsMetaData.getColumnLabel(columnNumber));

        // Gets the designated column's name.
        append(sb, XML_METADATA_TAG_COLUMN_NAME,
            rsMetaData.getColumnName(columnNumber));

        // Gets the designated column's SQL type.
        append(sb, XML_METADATA_TAG_COLUMN_TYPE,
            rsMetaData.getColumnType(columnNumber));

        // Gets the designated column's SQL type name.
        append(sb, XML_METADATA_TAG_COLUMN_TYPE_NAME,
            rsMetaData.getColumnTypeName(columnNumber));
```

```
// Gets the designated column's class name.
append(sb, XML_METADATA_TAG_COLUMN_CLASS_NAME,
    rsMetaData.getColumnClassName(columnNumber));

// Gets the designated column's table name.
append(sb, XML_METADATA_TAG_TABLE_NAME,
    rsMetaData.getTableName(columnNumber));

// Gets the designated column's number of decimal digits.
append(sb, XML_METADATA_TAG_PRECISION,
    rsMetaData.getPrecision(columnNumber));

// Gets the designated column's number of
// digits to right of the decimal point.
append(sb, XML_METADATA_TAG_SCALE,
    rsMetaData.getScale(columnNumber));

// Indicates whether the designated column is
// automatically numbered, thus read-only.
append(sb, XML_METADATA_TAG_IS_AUTO_INCREMENT,
    rsMetaData.isAutoIncrement(columnNumber));

// Indicates whether the designated column is a cash value.
append(sb, XML_METADATA_TAG_IS_CURRENCY,
    rsMetaData.isCurrency(columnNumber));

// Indicates whether a write on the designated
// column will succeed.
append(sb, XML_METADATA_TAG_IS_WRITABLE,
    rsMetaData.isWritable(columnNumber));

// Indicates whether a write on the designated
// column will definitely succeed.
append(sb, XML_METADATA_TAG_IS_DEFINITELY_WRITABLE,
    rsMetaData.isDefinitelyWritable(columnNumber));

// Indicates the nullability of values
// in the designated column.
append(sb, XML_METADATA_TAG_IS_NULLABLE,
    rsMetaData.isNullable(columnNumber));

// Indicates whether the designated column
// is definitely not writable.
append(sb, XML_METADATA_TAG_IS_READ_ONLY,
    rsMetaData.isReadOnly(columnNumber));
```

```
        // Indicates whether a column's case matters
        // in the designated column.
        append(sb, XML_METADATA_TAG_IS_CASE_SENSITIVE,
            rsMetaData.isCaseSensitive(columnNumber));

        // Indicates whether a column's case matters
        // in the designated column.
        append(sb, XML_METADATA_TAG_IS_SEARCHABLE,
            rsMetaData.isSearchable(columnNumber));

        // Indicates whether values in the designated
        // column are signed numbers.
        append(sb, XML_METADATA_TAG_IS_SIGNED,
            rsMetaData.isSigned(columnNumber));

        // Gets the designated column's table's catalog name.
        append(sb, XML_METADATA_TAG_CATALOG_NAME,
            rsMetaData.getCatalogName(columnNumber));

        // Gets the designated column's table's schema name.
        append(sb, XML_METADATA_TAG_SCHEMA_NAME,
            rsMetaData.getSchemaName(columnNumber));

        sb.append("/>");
        return sb.toString();

    }
```

Support methods are provided here:

```
/**
 * Append attribute=value to the string buffer denoted by sb.
 * @param sb the string buffer.
 * @param attribute the attribute name.
 * @param value the value of the attribute.
 */
private static void append(StringBuffer sb,
                           String attribute,
                           String value) {
    sb.append(attribute);
    sb.append("=\"");
    sb.append(value);
    sb.append("\" ");
}
```

```
/**
 * Append attribute=value to the string buffer denoted by sb.
 * @param sb the string buffer.
 * @param attribute the attribute name.
 * @param value the value of the attribute.
 */
private static void append(StringBuffer sb,
                           String attribute,
                           int value) {
    sb.append(attribute);
    sb.append("=\"");
    sb.append(value);
    sb.append("\" ");
}

/**
 * Append attribute=value to the string buffer denoted by sb.
 * @param sb the string buffer.
 * @param attribute the attribute name.
 * @param value the value of the attribute.
 */
private static void append(StringBuffer sb,
                           String attribute,
                           boolean value) {
    sb.append(attribute);
    sb.append("=\"");
    sb.append(value);
    sb.append("\" ");
}
```

Oracle Database Setup

The client using Oracle database will use the `employees` table described here:

```
$ sqlplus octopus/octopus
SQL*Plus: Release 9.2.0.1.0 - Production on Tue Feb 25 08:13:46 2003
Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
SQL> describe employees;
 Name              Null?    Type
 ----------------- -------- ----------------
 BADGENUMBER       NOT NULL NUMBER(38)
 NAME                       VARCHAR2(60)
 EMPLOYEETYPE               VARCHAR2(30)
 PHOTO                      BINARY FILE LOB
```

Client Using Oracle

```
import java.util.*;
import java.io.*;
import java.sql.*;

import jcb.db.*;
import jcb.meta.*;

public class TestOracleResultSetMetaDataTool {

    public static Connection getConnection() throws Exception {
        String driver = "oracle.jdbc.driver.OracleDriver";
        String url = "jdbc:oracle:thin:@localhost:1521:maui";
        String username = "octopus";
        String password = "octopus";
        Class.forName(driver);  // load Oracle driver
        return DriverManager.getConnection(url, username, password);
    }

    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
            conn = getConnection();
            // Create a result set
            stmt = conn.createStatement();
            rs = stmt.executeQuery("SELECT * FROM employees");

            System.out.println("-------- getResultSetMetaData -------------");
            System.out.println("conn="+conn);
            String rsMetaData = ResultSetMetaDataTool.getResultSetMetaData(rs);
            System.out.println(rsMetaData);
            System.out.println("-----------------------------------");
        }
        catch(Exception e){
            e.printStackTrace();
            System.exit(1);
        }
        finally {
            DatabaseUtil.close(stmt);
            DatabaseUtil.close(rs);
            DatabaseUtil.close(conn);
        }
    }
}
```

Client Output Using Oracle

```
-------- getResultSetMetaData -------------
conn=oracle.jdbc.driver.OracleConnection@66e815
<?xml version='1.0'>
<resultSetMetaData columnCount="4">
    <columnMetaData column="1"
                columnDisplaySize="22" columnLabel="BADGENUMBER"
                columnName="BADGENUMBER" columnType="2"
                columnTypeName="NUMBER" columnClassName="java.math.BigDecimal"
                tableName=""  precision="38" scale="0"
                isAutoIncrement="false" isCurrency="true"
                isWritable="true" isDefinitelyWritable="false"
                isNullable="0" isReadOnly="false"
                isCaseSensitive="false" isSearchable="true" isSigned="true"
                catalog=""  schema="" />
    <columnMetaData column="2"
                columnDisplaySize="60" columnLabel="NAME"
                columnName="NAME" columnType="12"
                columnTypeName="VARCHAR2" columnClassName="java.lang.String"
                tableName="" precision="60" scale="0"
                isAutoIncrement="false" isCurrency="false"
                isWritable="true" isDefinitelyWritable="false"
                isNullable="1" isReadOnly="false"
                isCaseSensitive="true" isSearchable="true" isSigned="true"
                catalog="" schema="" />
    <columnMetaData column="3"
                columnDisplaySize="30" columnLabel="EMPLOYEETYPE"
                columnName="EMPLOYEETYPE" columnType="12"
                columnTypeName="VARCHAR2" columnClassName="java.lang.String"
                tableName="" precision="30" scale="0"
                isAutoIncrement="false" isCurrency="false"
                isWritable="true" isDefinitelyWritable="false"
                isNullable="1" isReadOnly="false"
                isCaseSensitive="true" isSearchable="true" isSigned="true"
                catalog="" schema="" />
    <columnMetaData column="4"
                columnDisplaySize="530" columnLabel="PHOTO"
                columnName="PHOTO" columnType="-13"
                columnTypeName="BFILE" columnClassName="oracle.sql.BFILE"
                tableName="" precision="0" scale="0"
                isAutoIncrement="false" isCurrency="false"
                isWritable="true" isDefinitelyWritable="false"
                isNullable="1" isReadOnly="false"
                isCaseSensitive="false" isSearchable="false" isSigned="true"
                catalog="" schema="" />
</resultSetMetaData>
-----------------------------------
```

Oracle's Limitations on ResultSetMetaData.getTableName()

Note that the Oracle implementation of `ResultSetMetaData` does not provide table names for result set metadata. According to the Oracle documentation (`http://www.oracle.com/technology/sample_code/tech/java/codesnippet/jdbc/OracleResultSetMetaData.html`), the `OracleResultSetMetaData` interface does not implement the `getSchemaName()` and `getTableName()` methods because the underlying protocol does not make this feasible.

MySQL Database Setup

The client using MySQL database will use the `mypictures` table described here:

```
mysql> describe mypictures;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     |      | PRI | 0       |       |
| name  | varchar(20) | YES  |     | NULL    |       |
| photo | blob        | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Client Using MySQL

```
import java.util.*;
import java.io.*;
import java.sql.*;

import jcb.db.*;
import jcb.meta.*;

public class TestMySqlResultSetMetaDataTool {
    public static Connection getConnection() throws Exception {
        String driver = "org.gjt.mm.mysql.Driver";
        String url = "jdbc:mysql://localhost/octopus";
        String username = "root";
        String password = "root";
        Class.forName(driver);  // load MySQL driver
        return DriverManager.getConnection(url, username, password);
    }
    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
            conn = getConnection();
            // Create a result set
            stmt = conn.createStatement();
            rs = stmt.executeQuery("SELECT * FROM mypictures");
```

```
            System.out.println("-------- getResultSetMetaData -------------");
            System.out.println("conn="+conn);
            String rsMetaData = ResultSetMetaDataTool.getResultSetMetaData(rs);
            System.out.println(rsMetaData);
            System.out.println("-----------------------------------");
        }
        catch(Exception e){
            e.printStackTrace();
            System.exit(1);
        }
        finally {
            DatabaseUtil.close(stmt);
            DatabaseUtil.close(rs);
            DatabaseUtil.close(conn);
        }
    }
}
```

Client Output Using MySQL

```
-------- getResultSetMetaData -------------
conn=com.mysql.jdbc.Connection@1837697
<?xml version='1.0'>
<resultSetMetaData columnCount="3">
    <columnMetaData column="1"
            columnDisplaySize="11" columnLabel="id"
            columnName="id" columnType="4"
            columnTypeName="LONG" columnClassName="java.lang.Integer"
            tableName="mypictures" precision="11" scale="0"
            isAutoIncrement="false" isCurrency="false"
            isWritable="false" isDefinitelyWritable="false"
            isNullable="0" isReadOnly="false"
            isCaseSensitive="false" isSearchable="true" isSigned="true"
            catalog="null" schema="" />
    <columnMetaData column="2"
            columnDisplaySize="20" columnLabel="name"
            columnName="name" columnType="12"
            columnTypeName="VARCHAR" columnClassName="java.lang.String"
            tableName="mypictures" precision="0" scale="0"
            isAutoIncrement="false" isCurrency="false"
            isWritable="false" isDefinitelyWritable="false"
            isNullable="1" isReadOnly="false"
            isCaseSensitive="true" isSearchable="true" isSigned="false"
            catalog="null" schema="" />
```

```
    <columnMetaData column="3"
            columnDisplaySize="65535" columnLabel="photo"
            columnName="photo" columnType="-4"
            columnTypeName="BLOB" columnClassName="java.lang.Object"
            tableName="mypictures" precision="0" scale="0"
            isAutoIncrement="false" isCurrency="false"
            isWritable="false" isDefinitelyWritable="false"
            isNullable="1" isReadOnly="false"
            isCaseSensitive="true" isSearchable="true" isSigned="false"
            catalog="null" schema="" />
</resultSetMetaData>
-----------------------------------
```

4.8. How Do You Get a Table's Metadata Without Selecting Any Rows?

How do you get table's metadata without selecting any rows? This is possible by selecting required columns without selecting any records or rows. If you are interested in only the `ResultSetMetaData` (and not the `ResultSet` itself), then select required columns from a table so that the SQL query's `where` condition will be `false`. For example:

```
select id, name from employees where 1 = 0;
```

The condition "`1 = 0`" (as a boolean expression) is always `false`; therefore, no data will be selected at all, but you will get the result set metadata information.

The Solution

Our solution generates result set metadata for a given table identified with the `tableName` parameter. Using `tableName`, we create the following query (note that Oracle requires the table name to be in uppercase characters, while MySQL does not care):

```
String query = "select * from " + tableName.toUpperCase() + " where 1 = 0";
```

Then, we execute query (note that our query does not select any records from a given table because the `where` clause is always `false`) and then call the `getResultSetMetaData➥(ResultSet rs)` method. The reason for conversion of table name to uppercase characters is that some databases (such as Oracle) prefer, or even require, table names in uppercase.

```
/**
 * Get table's column names and their associated attributes
 * (type, size, nullable) The result is returned as XML
 * (as a String object);  if table name is null/empty
 * it returns null.
 *
 * @param conn the Connection object.
 * @param tableName the table name.
 * @return result set's metadata as an XML as String object;
```

```
     * this metadata includes column names and their associated
     * attributes: type, size, nullable.
     * @exception Failed to get the result set's metadata as XML.
     */
    public static String getTableMetaData(Connection conn,
                                          String tableName)
        throws Exception {
        Statement stmt = null;
        ResultSet rs = null;
        String query = null;
        try {
            if ((conn == null) ||
                (tableName == null) ||
                (tableName.length() == 0)) {
                return null;
            }

            query = "select * from "+ tableName.toUpperCase() + " where 1 = 0";
            stmt = conn.createStatement();
            rs = stmt.executeQuery(query);

            // Retrieves the number, types and properties
            // of this ResultSet object's columns.
            return getResultSetMetaData(rs);
        }
        finally {
            DatabaseUtil.close(rs);
            DatabaseUtil.close((stmt);
        }
    }
```

Oracle Database Setup

```
$ sqlplus octopus/octopus
SQL*Plus: Release 9.2.0.1.0 - Production on Wed Feb 26 17:38:26 2003
Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.
SQL> describe zdepts;
 Name              Null?    Type
 ----------------- -------- ------------
 DEPT              NOT NULL VARCHAR2(32)
 NAME              NOT NULL VARCHAR2(32)
 LOCATION          NOT NULL VARCHAR2(64)
 COSTCENTER                 CHAR(32)
```

Client Using an Oracle Table

```
import java.util.*;
import java.io.*;
import java.sql.*;

import jcb.db.*;
import jcb.meta.*;

public class TestOracleTableResultSetMetaDataTool {

    public static Connection getConnection() throws Exception {
        String driver = "oracle.jdbc.driver.OracleDriver";
        String url = "jdbc:oracle:thin:@localhost:1521:maui";
        String username = "octopus";
        String password = "octopus";
        Class.forName(driver);  // load Oracle driver
        return DriverManager.getConnection(url, username, password);
    }

    public static void main(String[] args) {
        Connection conn = null;
        try {
            conn = getConnection();
            System.out.println("-------- getResultSetMetaData -------------");
            System.out.println("conn="+conn);
            String deptTableName = "zdepts";
            String rsMetaData =
                ResultSetMetaDataTool.getTableMetaData(conn, deptTableName);
            System.out.println(rsMetaData);
            System.out.println("-----------------------------------");
        }
        catch(Exception e){
            e.printStackTrace();
            System.exit(1);
        }
        finally {
            DatabaseUtil.close(conn);
        }
    }
}
```

Output Using an Oracle Table

```
-------- getTableMetaData -------------
conn=oracle.jdbc.driver.OracleConnection@169ca65
<?xml version='1.0'>
<resultSetMetaData columnCount="4">
    <columnMetaData column="1" columnDisplaySize="32"
                columnLabel="DEPT" columnName="DEPT"
                columnType="12" columnTypeName="VARCHAR2"
                columnClassName="java.lang.String" tableName=""
                precision="32" scale="0"
                isAutoIncrement="false" isCurrency="false"
                isWritable="true" isDefinitelyWritable="false"
                isNullable="0" isReadOnly="false"
                isCaseSensitive="true" isSearchable="true"
                isSigned="true" catalog="" schema="" />
    <columnMetaData column="2" columnDisplaySize="32"
                columnLabel="NAME" columnName="NAME"
                columnType="12" columnTypeName="VARCHAR2"
                columnClassName="java.lang.String" tableName=""
                precision="32" scale="0"
                isAutoIncrement="false" isCurrency="false"
                isWritable="true" isDefinitelyWritable="false"
                isNullable="0" isReadOnly="false"
                isCaseSensitive="true" isSearchable="true"
                isSigned="true" catalog="" schema="" />
    <columnMetaData column="3" columnDisplaySize="64"
                columnLabel="LOCATION" columnName="LOCATION"
                columnType="12" columnTypeName="VARCHAR2"
                columnClassName="java.lang.String" tableName=""
                precision="64" scale="0"
                isAutoIncrement="false" isCurrency="false"
                isWritable="true" isDefinitelyWritable="false"
                isNullable="0" isReadOnly="false"
                isCaseSensitive="true" isSearchable="true"
                isSigned="true" catalog="" schema="" />
    <columnMetaData column="4" columnDisplaySize="32"
                columnLabel="COSTCENTER" columnName="COSTCENTER"
                columnType="1" columnTypeName="CHAR"
                columnClassName="java.lang.String" tableName=""
                precision="32" scale="0"
                isAutoIncrement="false" isCurrency="false"
                isWritable="true" isDefinitelyWritable="false"
                isNullable="1" isReadOnly="false"
                isCaseSensitive="true" isSearchable="true"
                isSigned="true" catalog="" schema="" />
</resultSetMetaData>
-----------------------------------
```

MySQL Database Setup

```
mysql> describe zperson;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| id    | int(11) | YES  |     | NULL    |       |
| photo | blob    | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.03 sec)
```

Client Using a MySQL Table

```
import java.util.*;
import java.io.*;
import java.sql.*;

import jcb.db.*;
import jcb.meta.*;

public class TestMySqlTableResultSetMetaDataTool {

    public static Connection getConnection() throws Exception {
        String driver = "org.gjt.mm.mysql.Driver";
        String url = "jdbc:mysql://localhost/octopus";
        String username = "root";
        String password = "root";
        Class.forName(driver);  // load MySQL driver
        return DriverManager.getConnection(url, username, password);
    }

    public static void main(String[] args) {
        Connection conn = null;
        try {
            conn = getConnection();
            System.out.println("-------- getTableMetaData -------------");
            System.out.println("conn="+conn);
            String deptTableName = "zperson";
            String rsMetaData =
              ResultSetMetaDataTool.getTableMetaData(conn, deptTableName);
            System.out.println(rsMetaData);
            System.out.println("-----------------------------------");
        }
        catch(Exception e){
            e.printStackTrace();
            System.exit(1);
        }
```

```
        finally {
            DatabaseUtil.close(conn);
        }
    }
}
```

Output Using a MySQL Table

```
-------- getTableMetaData -------------
conn=com.mysql.jdbc.Connection@1837697
<?xml version='1.0'>
<resultSetMetaData columnCount="2">
    <columnMetaData column="1" columnDisplaySize="11" columnLabel="id"
                columnName="id" columnType="4" columnTypeName="LONG"
                columnClassName="java.lang.Integer" tableName="ZPERSON"
                precision="11" scale="0" isAutoIncrement="false"
                isCurrency="false" isWritable="false"
                isDefinitelyWritable="false" isNullable="1"
                isReadOnly="false" isCaseSensitive="false"
                isSearchable="true" isSigned="true"
                catalog="null" schema="" />
    <columnMetaData column="2" columnDisplaySize="65535" columnLabel="photo"
                columnName="photo" columnType="-4" columnTypeName="BLOB"
                columnClassName="java.lang.Object" tableName="ZPERSON"
                precision="0" scale="0" isAutoIncrement="false"
                isCurrency="false" isWritable="false"
                isDefinitelyWritable="false" isNullable="1"
                isReadOnly="false" isCaseSensitive="true"
                isSearchable="true" isSigned="false"
                catalog="null" schema="" />
</resultSetMetaData>
-----------------------------------
```

4.9. How Do You Retrieve the Column Types from a ResultSet?

In GUI database applications, you must determine the type of a `ResultSet`'s columns in order to validate column values before sending them to the database server. So, to retrieve the column data type from a result set object, let's return the result as a `java.util.List`, where each element of the list is a `java.lang.String`, which represents a data type.

The Solution: getTypes()

```
/**
 * Get the column data type from a result set object.
 * @param rs a ResultSet object to process
 * @throws SQLException Failed to get data types from a ResultSet
```

```
 * @return the result as a java.util.List, where each element of the
 *  list is a String object (designates types).
 */
public static java.util.List getTypes(ResultSet rs)
    throws SQLException {
    if (rs == null) {
        return null;
    }

    ResultSetMetaData meta = rs.getMetaData();
    if (meta == null) {
        return null;
    }

    java.util.List types = new java.util.ArrayList();
    for (int i = 1; i <= meta.getColumnCount(); i++){
        String columnType = meta.getColumnTypeName(i);
        types.add(columnType);
    }

    return types;
}
```

Discussion

- The solution presented is a general one: this solution can be applied to any `ResultSet` object.
- `ResultSetMetaData.getColumnCount()` returns the total number of columns for a `ResultSet` object.
- `ResultSetMetaData.getColumnTypeName(i)` returns the column data type name for the *i*th position (positions start from 1).

4.10. How Do You Retrieve the Column Name/Data/Type from a ResultSet?

Next we'll see how to retrieve the column name from a result set as well as its associated type and data. To solve this problem, let's return the result as a `java.util.List`, where each element of the list is a `java.util.Map`, which represents a retrieved row or record.

The Solution

We'll use two methods:

- `getNameAndData(ResultSet rs)`: Retrieves the column name from a result set as well as its associated data.
- `getNameAndType(ResultSet rs)`: Retrieves the column name from a result set as well as its associated data type.

getNameAndData()

```
/**
 * Get the column name from a result set as well as its associated data.
 * @param rs a ResultSet object to process
 * @throws SQLException Failed to get name and data from a ResultSet
 * @return the result as a java.util.List (where each element of the
 *  list is a java.util.Map, which represents a retrieved row/record).
 */
public static java.util.List getNameAndData(ResultSet rs)
    throws SQLException {
    if (rs == null) {
        return null;
    }

    ResultSetMetaData meta = rs.getMetaData();
    if (meta == null) {
        return null;
    }

    java.util.List rows = new java.util.ArrayList();
    while(rs.next()) {
        Map row = new HashMap();
        for (int i = 1; i <= meta.getColumnCount(); i++){
            String column = meta.getColumnName(i);
            String value = rs.getString(i);
            if (value == null){
                value="";
            }
            row.put(column,value.trim());
        }
        rows.add(row);
    }
    return rows;
}
```

Discussion

- The solution presented is a simplistic one: it assumes that all column types are either `VARCHAR` or `CHAR`. For getting values, we invoke `ResultSet.getString()`, which might not work for some data types such as `BLOB`s. The revised solution might be to get both the name and data type of each column, and then, based on the data type, invoke `ResultSet.getXXX()`, where `XXX` is derived from the data type of a column name.
- `ResultSetMetaData.getColumnCount()` returns the number of columns returned from a `ResultSet` object.
- `ResultSetMetaData.getColumnName(i)` returns the column name for the *i*th position (positions start from 1).

getNameAndType()

```
/**
 * Get the column name from a result set as well as its associated data type.
 * @param rs a ResultSet object to process
 * @throws SQLException Failed to get name and data type from a ResultSet
 * @return the result as a java.util.Map, where each element of the
 *  map is a pair of (name, data type).
 */
public static java.util.Map getNameAndType(ResultSet rs)
    throws SQLException {
    if (rs == null) {
        return null;
    }

    ResultSetMetaData meta = rs.getMetaData();
    if (meta == null) {
        return null;
    }

    java.util.Map result = new HashMap();
    for (int i = 1; i <= meta.getColumnCount(); i++){
        String columnName = meta.getColumnName(i);
        String columnType = meta.getColumnTypeName(i);
        result.put(columnName, columnType);
    }

    return result;
}
```

Discussion

- The solution I've presented is a general one, which means you can apply it to any `ResultSet` object.
- `ResultSetMetaData.getColumnName(i)` returns the column name for the *i*th position (positions start from 1).
- `ResultSetMetaData.getColumnTypeName(i)` returns the column data type name for the *i*th position (positions start from 1).

4.11. What Is ResultSet Holdability?

JDBC 3.0 adds support for specifying result set (or cursor) holdability. Result set *holdability* is the ability to specify whether cursors (or a result set as `java.sql.ResultSet` objects) should be held open or closed at the end of a transaction. A holdable cursor, or result set, is one that does not automatically close when the transaction that contains the cursor is committed. You may improve database performance by including the `ResultSet` holdability. If `ResultSet`

objects are closed when a commit operation is implicitly or explicitly called, this can also improve performance.

According to Java Database Connectivity, Version 3.0:

> *When the method Connection.commit is called, open ResultSet objects that have been created during the current transaction are closed. With ResultSet holdability, it is possible to exercise greater control on when the ResultSet objects are closed when the commit operation is performed. Using the ResultSet holdability, ResultSet objects that are closed when a commit operation is performed can result in better performance in some applications.*

To specify the holdability of a `ResultSet` object, you must do so when preparing a `java.sql.Statement` (or `java.sql.PreparedStatement` or `java.sql.CallableStatement`) using the `Connection.createStatement()`, `Connection.prepareStatement()`, or `Connection.prepareCall()` method. The `ResultSet` holdability may be one of the following constants (defined in the `ResultSet` interface):

- `HOLD_CURSORS_OVER_COMMIT`: The constant indicating that `ResultSet` objects should not be closed when the method `Connection.commit()` is called.
- `CLOSE_CURSORS_AT_COMMIT`: The constant indicating that `ResultSet` objects should be closed when the method `Connection.commit()` is called.

In general, closing a cursor (as a `ResultSet` object) when a transaction is committed results in better performance.