JDBC Maintenance Release 4.1

Description:

Maintenance review of the JDBC 4.0 Specification

Maintenance Lead:

Lance Andersen, Oracle Corporation

Feedback:

Comments should be sent to jsr221-comments@jcp.org

Rationale for Changes:

The goal is to address several specification issues and to provide support for Automatic Resource Management introduced in Java SE 7 as well as several minor enhancements requested by the JDBC EG and user community.

Accepted Changes:

1. The following interfaces now extend java.lang.AutoClosable interface in order to support Automatic Resource Management:

    1. java.sql.Connection
    2. java.sql.Statement
    3. java.sql.ResultSet

1. The following methods have been added to java.sql.CallableStatement:

    1. `<T> T getObject(int parameterIndex, Class<T> type)throws SQLException`
    2. `<T> T getObject(String parameterName, Class<T> type)throws SQLException`

1. The following methods have been added to java.sql.Connection:

    1. `void abort(Executor executor)` throws SQLException

1. int **getNetworkTimeout**() throws SQLException

1. String **getSchema**() throws SQLException

1. void **setNetworkTimeout**(Executor executor,int milliseconds) throws SQLException

1. void **setSchema**(String schema) throws SQLException

1. The following methods have been clarified in java.sql.Connection:

1. Map <java.lang.String,java.lang.Class<?>> **getTypeMap**()throws SQLException

1. void **setCatalog**(String catalog) throws SQLException

1. void **setTypeMap**(Map<java.lang.String,java.lang.Class<?>> map) throws SQLException

1. The following methods have been added to java.sql.DatabaseMetaData:

1. ResultSet **getPseudoColumns**(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern) throws SQLException
2. boolean **generatedKeyAlwaysReturned**() throws SQLException

1. The following methods have been clarified in java.sql.DatabaseMetaData:

1. ResultSet **getProcedureColumns**(String catalog,String schemaPattern,java.lang.String procedureNamePattern,String columnNamePattern)throws SQLException
2. ResultSet **getColumns**(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern) throws SQLException;

1. The methods java.sql.Date.valueOf() and java.sql.Timestamp.valueOf () now allow you to omit the leading zero for month or day

1. The following methods have been clarified in java.sql.Timestamp:
   1. `public int compareTo(java.util.Date o)`

1. The following method has been added to java.sql.Driver and javax.sql.CommonDataSource:

   1. `Logger getParentLogger() throws SQLFeatureNotSupportedException`

1. The following methods have been clarified in java.sql.PreparedStatement:

   1. `boolean execute() throws SQLException`
   2. `ResultSet executeQuery() throws SQLException`
   3. `int executeUpdate() throws SQLException`

1. The enum java.sql.PseudoColumnUsage has been added

1. The following methods have been added to java.sql.ResultSet:
   1. `<T> T getObject(int columnIndex, Class<T> type)throws SQLException`
   2. `<T> T getObject(String columnName, Class<T> type)throws SQLException`

1. The following method has been clarified in java.sql.ResultSet:

   1. `boolean absolute(int row)` throws SQLException

1. The following subclasses of SQLException have been clarified to indicate that they can be thrown for vendor specific reasons:

    1. java.sql.SQLDataException
    2. java.sql.SQLIntegrityConstraintViolationException
    3. java.sql.SQLInvalidAuthorizationSpecException
    4. java.sql.SQLNontransientConnectionException
    5. java.sql.SQLSyntaxErrorException
    6. java.sql.SQLTransactionRollbackException
    7. java.sql.SQLTransientConnectionException

1. The following permission target names have been added to java.sql.SQLPermission:
    1. `callAbort`
    2. `setNetworkTimeout`
    3. `setSyncFactory`

1. The following methods have been added to java.sql.Statement:

    1. void **`closeOnCompletion`**`() throws SQLException`
    2. boolean **`isCloseOnCompletion`**`()throws SQLException`

1. The following methods have been clarified in java.sql.Statement:

    1. `void` **`addBatch`**`(String sql) throws SQLException`
    2. `boolean` **`execute`**`(String sql) throws SQLException`
    3. `boolean` **`execute`**`(String sql, int autoGeneratedKeys) throws SQLException`
    4. `boolean` **`execute`**`(String sql, int[] columnIndexes) throws SQLException`
    5. `boolean` **`execute`**`(String sql, String[] columnNames) throws SQLException`
    6. `int[]` **`executeBatch`**`() throws SQLException`
    7. `ResultSet` **`executeQuery`**`(String sql) throws SQLException`
    8. `int` **`executeUpdate`**`(String sql) throws SQLException`
    9. `int` **`executeUpdate`**`(String sql, int autoGeneratedKeys) throws SQLException`
    10. `int` **`executeUpdate`**`(String sql, int[] columnIndexes) throws SQLException`

11. `int` **`executeUpdate`**`(String sql, String[] columnNames) throws SQLException`
12. `void` **`setQueryTimeout`**`(int seconds) throws SQLException`

1. Added the Limiting Returned Rows Escape to section 13.4.6:

   The escape syntax for limiting the number of rows returned by a query is:

   {limit <limit clause>}

   where the format for the <limit clause> is:

   rows [offset row_offset]

   The square brackets indicate that the 'offset row_offset' portion is optional. The value given for rows indicates the maximum number of rows to be returned from this query. The row_offset indicates the number of rows to skip from the rows returned from the query before beginning to return rows. A value of 0 for row_offset means do not skip any rows. The value for rows and row_offset must be a 0 or greater integer value.

   The following query will return no more than 20 rows:

   Statement stmt = con.createStatement();
   stmt.executeQuery("SELECT * FROM TABLE1 " +
   "WHERE F1 >100 {limit 20}");

   Note:
   A value of 0 for rows may return no rows or all rows depending on the underlying database.

1. Add the following functions to Appendix C.2, String Functions:

   1. POSITION(substring in string [, CHARACTERS|OCTETS)
   2. CHAR_LENGTH(string [, CHARACTERS|OCTETS)
   3. CHARACTER_LENGTH(string[, CHARACTERS|OCTETS)
   4. SUBSTRING(string, start, length[, CHARACTERS|OCTETS)
   5. LENGTH(string[, CHARACTERS|OCTETS)

   Requires SQL 2003 Feature T061, "UCS Support"

1. Add the following mapping to table B-4, Mapping from Java Object to JDBC Types:

   1. Map Java Object Type java.util.Date and java.util.Calendar to the JDBC Type TIMESTAMP
   2. Map Java Object Type java.math.BigInteger to JDBC Type BIGINT

1. Add the following mapping to table B-5, conversions Performed by setObject and setNull between Java Object Types and Target JDBC Types:
    1. Allow conversion of java.util.Date and java.util.Calendar to CHAR, VARCHAR, LONGVARCHAR, DATE, TIME and TIMESTAMP
    2. Allow conversion of java.math.BigInteger to CHAR, VARCHAR, LONGVARCHAR and BIGINT