JDBC Maintenance Release 4.2

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 as well as several minor enhancements requested by the JDBC EG and user community.

Proposed Changes:

1. **Addition of REF CURSOR Support**

The REF CURSOR data type is supported by several databases. To return a REF CURSOR from a stored procedure, the CallableStatement method registerOutParameter may be used specifying Types.REF_CURSOR as the data type to be returned. The CallableStatement method getObject, specifying ResultSet as the type to convert the returned object to, would be called to retrieve the ResultSet representing the REF CURSOR. The returned result set is a forward, read-only result set.

If registerOutParameter is called specifying Types.REF_CURSOR and the JDBC driver does not support this data type, a SQLFeatureNotSupportedException will be thrown.

The following example demonstrates the a

CallableStatement that returns a ResultSet using a REF CURSOR:

```
CallableStatement cstmt = conn.prepareCall(" { call
mySproc(?) }");
cstmt.registerOutParameter(1, Types.REF_CURSOR);
cstmt.executeQuery();
ResultSet rs =   cstmt.getObject(1, ResultSet.class);
while (rs.next ())  {
  System.out.println("Name="+ rs.getString(1));
}
```

To determine if a JDBC Driver supports REF CURSOR, an application may call DatabaseMetaData.supportsRefCursors.

2. **Addition of the java.sql.DriverAction Interface**

   This is an interface that a java.sql.Driver must implement when it wants to be notified by DriverManager. The JDBC driver's static initialization block must call DriverManager.registerDriver(java.sql.Driver, java.sql.DriverAction) in order to inform DriverManager which DriverAction implementation to call when the JDBC driver is de-registered.

   java.sql.DriverAction contains the following method:

   - void deregister()

3. **Addition of the java.sql.SQLType Interface**

   This interface is used to create an object that is used to identify a generic SQL type, called a JDBC type or a vendor specific data type.

java.sql.SQLType contains the following methods:

- String getName()
- String getVendor();
- Integer getVendorTypeNumber()

4. **Addition of the java.sql.JDBCType Enum**

   An Enum used to identify generic SQL Types, called JDBC Types. The intent is to use JDBCType in place of the constants defined in Types.java.

5. **Add Support for large update counts**

   JDBC methods that return an update count only return an int value. As datasets continue to grow, this has caused problems for some JDBC vendors.

   To address this problem, the following methods have been added to Statement that will return a long. These methods should be used when the returned row count may exceed Integer.MAX_VALUE:

   - default long[] executeLargeBatch() throws SQLException
   - default long executeLargeUpdate(String sql) throws SQLException
   - default long executeLargeUpdate(String sql, int autoGeneratedKeys) throws SQLException
   - default long executeLargeUpdate(String sql, int[] columnIndexes) throws SQLException
   - default long executeLargeUpdate(String sql, String[] columnNames) throws SQLException
   - default long getLargeUpdateCount() throws SQLException
   - default long getLargeMaxRows()throws SQLException
   - default void setLargeMaxRows(long rows) throws SQLException

BatchUpdateException has been enhanced to include the following additional methods and constructors:

- public long[] getLargeUpdateCounts() throws SQLException
- BatchUpdateException(String reason, String SQLState, int vendorCode, long[] updateCounts, Throwable cause)

6. **java.sql.Driver changes**

   The following methods have been clarified in java.sql.Driver:

   - boolean acceptsURL() throws SQLException
     - A SQLException will be thrown if the url is null
   - Connection connect(String, Properties) throws SQLException
     - A SQLException will be thrown if the url is null
     - If a property is specified as part of the url and is also specified in the Properties object, it is implementation-defined as to which value will take precedence. For maximum portability, an application should only specify a property once.

7. **java.sql.DriverManager changes**

   The following methods have been added to to java.sql.DriverManager:

   - public static void registerDriver(Driver, DriverAction) throws SQLException

   The following methods have been clarified in java.sql.DriverManager:

   - public static void deregisterDriver(Driver) throws SQLException
     - Indicate that no action is taken if the specified driver

is null
    - A SecurityException will be thrown if a security manager exists and its checkPermission method denies permission to deregister a driver.
- Connection getConnection(String, String, String) throws SQLException
    - A SQLException will be thrown if the url is null
    - A SQLTimeoutException may be thrown if the value specified by the setLoginTimeout method has been exceeded and has at least tried to cancel the current database connection attempt
    - If a property is specified as part of the url and is also specified in the Properties object, it is implementation-defined as to which value will take precedence. For maximum portability, an application should only specify a property once
- Connection getConnection(String, Properties) throws SQLException
    - A SQLException will be thrown if the url is null
    - A SQLTimeoutException may be thrown if the value specified by the setLoginTimeout method has been exceeded and has at least tried to cancel the current database connection attempt
    - If a property is specified as part of the url and is also specified in the Properties object, it is implementation-defined as to which value will take precedence. For maximum portability, an application should only specify a property once
- static void registerDriver(Driver) throws SQLException
    - No action is taken if the specified driver was previously registered

8. **java.sql.DatabaseMetaData changes**

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

    - default boolean supportsRefCursors() throws

SQLException
- default long getMaxLogicalLobSize() throws SQLException

The following methods have been modified in java.sql.DatabaseMetaData

- getIndexInfo() – The returned CARDINALITY and PAGES column values now return a long value

9. **java.sql.Date changes**

The following methods have been added to java.sql.Date

- public java.time.instant toInstant()
- public java.time.LocalDate toLocalDate()
- public static Date valueOf(java.time.LocalDate)

10. **java.sql.Time changes**

The following methods have been added to java.sql.Time:

- public java.time.instant toInstant()
- public java.time.LocalTime toLocalTime()
- public static Time valueOf(java.time.LocalTime)

11. **java.sql.Timestamp changes**

The following methods have been added to java.sql.Timestamp:

- public static Timestamp from(java.time.Instant)
- public java.time.instant toInstant()
- public java.time.LocalDateTime toLocalDateTime()
- public static Timestamp valueOf(java.time.LocalDateTime)

12. **java.sql.Statement changes**

The following clarification was made to the setEscapeProcessing method:

> The Connection and DataSource property escapeProcessing may be used to change the default escape processing behavior. A value of true (the default) enables escape Processing for all Statement objects. A value of false disables escape processing for all Statement objects. The setEscapeProcessing method may be used to specify the escape processing behavior for an individual Statement object.

13. java.sql.CallableStatement changes

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

- default void registerOutputParameter(int parameterIncex, SQLType sqlType) throws SQLException
- default void registerOutputParameter(int parameterIndex, SQLType sqlType, int scale) throws SQLException
- default void registerOutputParameter(int parameterIndex, SQLType sqlType, String typeName) throws SQLException
- default void registerOutputParameter(String parameterName, SQLType sqlType) throws SQLException
- default void registerOutputParameter(String parameterName, SQLType sqlType, int scale) throws SQLException
- default void registerOutputParameter(String parameterName, SQLType sqlType, String typeName) throws SQLException
- default void setObject(String parameterName, Object x, SQLType targetSqlType) throws SQLException
- default void setObject(String parameterName, Object x, SQLType tragetSqlType, int scaleOrLength) throws SQLException

The following methods have been clarified:

- <T> T getObject(int, Class<T>) throws SQLException
  - Add the description of the Type parameter
- <T> T getObject(String, Class<T> throws SQLException
  - Add the description of the Type parameter

14. java.sql.PreparedStatement changes

The following methods have been added to java.sql.PreparedStatement

- default long executeLargeUpdate() throws SQLException
- default void setObject(int parameterIndex, Object x, SQLType targetSqlType) throws SQLException
- default long setObject(int parameterIndex, Object x, SQLType targetSqlType, int scaleOrLength) throws SQLException

15. java.sql.ResultSet changes

The following methods have been added to java.sql.ResultSet:

- default long executeLargeUpdate() throws SQLException
- default void updateObject(int columnIndex, Object x, SQLType targetSqlType) throws SQLException
- default void updateObject (int columnIndex, Object x, SQLType targetSqlType, int scaleOrLength) throws SQLException
- default void updateObject (String columnLabel, Object x, SQLType targetSqlType) throws SQLException
- default void updateObject (String columnLabel, Object x, SQLType targetSqlType, int scaleOrLength) throws SQLException

The following methods have been clarified:

- <T> T getObject(int, Class<T>) throws SQLException
  - Add the description of the Type parameter
- <T> T getObject(String, Class<T> throws SQLException
  - Add the description of the Type parameter

16. **java.sql.SQLInput changes**

The following methods have been added to java.sql.SQLInput:

- default <T> T readObject(Class<T>) throws SQLException

17. **java.sql.SQLOutput changes**

The following methods have been added to java.sql.SQLOutput:

- default void writeObject(Class, SQLType) throws SQLException

18. **java.sql.Types changes**

The following constants have been added to java.sql.Types:

- public static final int REF_CURSOR = 2012
- public static final int TIME_WITH_TIMEZONE = 2013
- public static final int TIMESTAMP_WITH_TIMEZONE = 2014

19. **javax.sql.DataSource and javax.sql.XADataSource changes**

Clarify that a no-arg constructor is required to be provided

20. **java.sql.SQLXML changes**

The following methods have been clarified:

- <T extends javax.xml.transform.Source> T getSource(Class<T>) throws SQLException
    - Add the description of the Type parameter
- <T extends javax.xml.transform.Source> T setResult(Class<T> throws SQLException
    - Add the description of the Type parameter

21. **Add the following mapping to table B-4, Mapping from Java Objects to JDBC Types**
    - Map the Java Object Type javax.time.LocalDate to the JDBC Type DATE
    - Map the Java Object Type javax.time.LocalTime to the JDBC Type TIME
    - Map the Java Object Type javax.time.LocalDateTime to the JDBC Type TIMSTAMP
    - Map the Java Object Type javax.time.OffsetTime to the JDBC Type TIME_WITH_TIMEZONE
    - Map the Java Object Type javax.time.OffsetDateTime to the JDBC Type TIMESTAMP_WITH_TIMEZONE

22. **Add the following mapping to table B-5, Mapping, conversions Performed by setObject and setNull between Java Object Types**
    - Allow the conversion of javax.time.LocalDate to CHAR, VARCHAR, LONGVARCHAR, and DATE
    - Allow the conversion of javax.time.LocalTime to CHAR, VARCHAR, LONGVARCHAR, and TIME
    - Allow the conversion of javax.time.LocalDateTime to CHAR, VARCHAR, LONGVARCHAR, DATE, TIME, and TIMESTAMP
    - Allow the conversion of javax.time.OffsetTime to CHAR, VARCHAR, LONGVARCHAR, and TIME_WITH_TIMEZONE
    - Allow the conversion of javax.time.OffsetDateTime to CHAR, VARCHAR, LONGVARCHAR, TIME_WITH_TIMEZONE, and TIMESTAMP_WITH_TIMEZONE

23. **Add the following mapping to table B-6, Use ResultSet getter Methods to Retrieve JDBC Data Types**

- Allow getObject to return TIME_WITH_TIMEZONE and TIMESTAMP_WITH_TIMEZONE