3 JDBC Standards Support

Oracle Java Database Connectivity (JDBC) drivers support different versions of the JDBC standard features. These features are provided through the `oracle.jdbc` and `oracle.sql` packages. These packages support JDK 8, JDK 11, and JDK 17.

This chapter discusses the JDBC standards support in Oracle JDBC drivers for the most recent releases. It contains the following sections:

3.1 Support for JDBC 4.2 Standard

Oracle Database Release 23ai JDBC drivers provide support for JDBC 4.2 standard through JDK 8. This section lists some of the important methods available in this release.

The %Large% Methods

This release of Oracle JDBC drivers support the following methods introduced in JDBC 4.2 standard, which deal with `long` values:

- `executeLargeBatch()`
- `executeLargeUpdate(String sql)`
- `executeLargeUpdate(String sql, int autoGeneratedKeys)`
- `executeLargeUpdate(String sql, int[] columnIndexes)`
- `executeLargeUpdate(String sql, String[] columnNames)`
- `getLargeMaxRows()`
- `getLargeUpdateCount()`
- `setLargeMaxRows(long max)`

These new methods are available as part of the `java.sql.Statement` interface. The `%Large%` methods are identical to the corresponding *non-large* methods, except that they work with `long` values instead of `int` values. For example, the `executeUpdate` method returns the number of rows updated as an `int` value, whereas, the `executeLargeUpdate` method returns the number of rows updated as a `long` value. If the number of rows is greater than the value of `Integer.MAX_VALUE`, then your application must use the `executeLargeUpdate` method.

The following code snippet shows how to use the `executeLargeUpdate(String sql)` method:

```
...
Statement stmt = conn.createStatement();
stmt.executeQuery("create table BloggersData (FIRST_NAME varchar(100), ID
int)");
long updateCount = stmt.executeLargeUpdate("insert into BloggersData
(FIRST_NAME,ID) values('John',1)");
...
```

The SQLType Methods

This release of Oracle JDBC drivers support the following methods introduced in JDBC 4.2 standard, which take `SQLType` parameters:

- `setObject`

  The `setObject` method sets the value of the designated parameter for the specified object. This method is similar to the `setObject(int parameterIndex, Object x, SQLType targetSqlType, int scaleOrLength)` method, except that it assumes a scale of zero. The default implementation of this method throws `SQLFeatureNotSupportedException`.

  ```
  void setObject(int parameterIndex, java.lang.Object x, SQLType
  targetSqlType) throws SQLException
  ```

- `updateObject`

  The `updateObject` method takes the column index as a parameter and updates the designated column with an Object value.
- `registerOutParameter`

  The `registerOutParameter` method registers a specified parameter to be of JDBC type `SQLType`.

The following code snippet shows how to use the `setObject` method:

```
...
int empId = 100;
connection.prepareStatement("SELECT FIRST_NAME, LAST_NAME FROM EMPLOYEES
WHERE EMPNO = ?");
preparedStatement.setObject(1, Integer.valueOf(empId), OracleType.NUMBER);
...
```

See Also:

JDBC 4.2 Documentation

3.2 Support for JDBC 4.3 Standard

Starting from Release 19c, Oracle Database JDBC drivers provide support for Standard JDBC 4.3 features through JDK 11 and JDK 17. This section describes some of the important APIs available in this release.

Sharding Support

Sharding is a data tier architecture, where data is horizontally partitioned across independent databases. Each database in such a configuration is called a shard. All shards together make up a single logical database, which is referred to as a sharded database (SDB). You can use the `DatabaseMetaData.supportsSharding` method to determine whether a JDBC Driver supports sharding or not.

Sharding support includes addition of the following APIs:

- `javax.sql.XAConnectionBuilder` Interface
  This is a builder interface created from a `XADataSource` object, which you can use to establish a connection to the database that the data source object represents.
- `java.sql.ShardingKey` Interface
  This interface is used to indicate that the current object represents a Sharding Key. You can create a `ShardingKey` instance using the `ShardingKeyBuilder` interface.
- `java.sql.ShardingKeyBuilder` Interface
  This is a builder interface created from a `DataSource` or `XADataSource` object, used to create a `ShardingKey` with subkeys of supported data types.

See Also:

Overview of Database Sharding for JDBC Users

Enhancements to the java.sql.Connection Interface

The following methods have been added to the `java.sql.Connection` Interface:

- `default void beginRequest throws SQLException`
- `default void endRequest throws SQLException`
- `default void setShardingKey(ShardingKey shardingKey) throws SQLException`
- `default void setShardingKey(ShardingKey shardingKey, ShardingKey superShardingKey) throws SQLException`
- `default void setShardingKeyIfValid(ShardingKey shardingKey, int timeout) throws SQLException`
- `default void setShardingKeyIfValid(ShardingKey shardingKey, ShardingKey superShardingKey, int timeout) throws SQLException`

Enhancements to the java.sql.DatabaseMetaData Interface

The following methods have been added to the `java.sql.DatabaseMetaData` Interface:

```
default boolean supportsSharding() throws SQLException
```

Enhancements to the java.sql.Statement Interface

The following methods have been added to the `java.sql.Statement` Interface:

- `default String enquoteIdentifier(String identifier, Boolean alwaysQuote) throws SQLException`
- `default String enquoteLiteral(String val) throws SQLException`
- `default String enquoteNCharLiteral(String val) throws SQLException`
- `default boolean isSimpleIdentifier(String identifier) throws SQLException`

See Also:

Oracle Database JDBC Java API Reference