JDBC Prepared Statements

The JDBC Statement interface is not so flexible. Since you have to specify each statement as a string, there is no way to set statement parameters such as a record id or a column value.

Another disadvantage of the Statement interface is that each time you execute it, the database compiles the SQL statement without caching it. It leads to an execution time increase.

To cope with such problems, JDBC API provides a special type of statement that is called a **prepared statement**, represented by the java.sql.PreparedStatement interface.

Similarly to the statements, prepared statements can create, select, update, and delete database data. However, prepared statements have several advantages.

Prepared Statement parameters

To understand the difference between Statement and PreparedStatement and see how PreparedStatement handles SQL query parameters, let's consider an example. First, we create an SQLite database called music and add the artists table to the database:

```
CREATE TABLE music.artists
(
   id          INTEGER PRIMARY KEY,
   name         TEXT    NOT NULL,
   origin      TEXT,
   songs_number INTEGER
);
```

To add the most influential band of all time to the artists table by using Statement interface, we can write the following code:

```
try (Statement statement = con.createStatement()) {
   statement.executeUpdate("INSERT INTO artists (name, origin,
songs_number) VALUES " +
           "('The Beatles', 'Liverpool, England', 213)");
}
```

Now if we need to add more bands, we have to provide a new string with a changed name, origin, and songs_number values. However, PreparedStatement provides a more elegant way to do so:

```
String insert = "INSERT INTO artists (name, origin, songs_number) VALUES
(?, ?, ?)";
```

```
try (PreparedStatement preparedStatement = con.prepareStatement(insert)) {
    preparedStatement.setString(1, "The Beatles");
    preparedStatement.setString(2, "Liverpool, England");
    preparedStatement.setInt(3, 213);

    preparedStatement.executeUpdate();
}
```

To create a PreparedStatement object, we used the prepareStatement(String sql) method of the Connection object. Note, how the value of the insert variable that we passed as a parameter to the prepareStatement method contains the question mark (?) symbols. Such symbols are placeholders for the values that will be passed to the query. The first placeholder corresponds to the name of the artist, the second one to its origin, and the third one to the number of the artist's songs.

Prepared Statement setters

PreparedStatement interface provides setter methods to replace an SQL query placeholder with the value. The signature of the setters is the following:
1) The name of the method is setXXX, where XXX is a type. We need to use the appropriate setXXX method for a given SQL type. In the example above, we've used the setString method since the SQL type of artist name is TEXT.
2) Each PreparedStatement.setXXX method accepts at least two parameters. The first one indicates the parameter index, while the second one — is the parameter value of the specific type. In the example above,
the songs_number corresponds to the third ? placeholder and has
an INTEGER type in the database, which corresponds to the Java int type.

Sometimes you need to set the parameter value to NULL. For such cases, PreparedStatement provides a setNull method that accepts the parameter index and an SQL type code defined in Java. For example, we can set the artist's origin to NULL in the following way:
String updateOrigin = "UPDATE artists SET origin = ? WHERE id = ?";

```
try (PreparedStatement preparedStatement =
con.prepareStatement(updateOrigin)) {
    preparedStatement.setNull(1, Types.VARCHAR);
    preparedStatement.setInt(2, 1);

    preparedStatement.executeUpdate();
}
```

Another possible situation is when you do not know the exact type of the parameter at runtime. In such a case, you can use the setObject method that is

the most general one. It can accept a parameter index and any Object as an argument. Let's see an example:

```
String insert = "INSERT INTO artists (name, songs_number) VALUES (?, ?)";
try (PreparedStatement preparedStatement = con.prepareStatement(insert)) {
    preparedStatement.setObject(1, new StringBuilder("Madonna"));
    preparedStatement.setObject(2, new Long(88));

    preparedStatement.executeUpdate();
}
```

Statement placeholders can be used for values only, which means that you cannot replace an SQL keyword with a placeholder.

Conclusion

PreparedStatement interface represents a precompiled SQL statement that provides methods to bind the query parameters to the values. In contrast to Statement, PreparedStatement is cached by the database. This means that if you execute a statement for the first time, the database will compile, cache, and execute it. Then, if you execute the same statement with other parameters, from the database perspective, the statement hasn't been changed, so it will not recompile the statement because the database contains a cached one. It leads to boosting database execution performance.

All things considered, using prepared statements reduces the execution time, eases inserting parameters into SQL statements, and allows us to reuse a prepared statement with new parameters. In addition, prepared statements protect against SQL injection through a separation of the query code and parameter values.