# JDBC: ResultSet



Last update: 2014-06-23

The text about **queries** I shows how the result of a query is returned as a `java.sql.ResultSet`. This`ResultSet` is then iterated to inspect the result. This text will examine the `ResultSet` interface in a few more details.

## A ResultSet Contains Records

A `ResultSet` consists of records. Each records contains a set of columns. Each record contains the same amount of columns, although not all columns may have a value. A column can have a `null` value. Here is an illustration of a `ResultSet`:

| Name | Age | Gender |
|---|---|---|
| John | 27 | Male |
| Jane | 21 | Female |
| Jeanie | 31 | Female |

**ResultSet example - records with columns**

This `ResultSet` has 3 different columns (Name, Age, Gender), and 3 records with different values for each column.

## Creating a ResultSet

You create a `ResultSet` by executing a `Statement` or `PreparedStatement`, like this:

```
Statement statement = connection.createStatement();

ResultSet result = statement.executeQuery("select * from people");
```

Or like this:

```
String sql = "select * from people";
PreparedStatement statement = connection.prepareStatement(sql);

ResultSet result = statement.executeQuery();
```

### ResultSet Type, Concurrency and Holdability

When you create a `ResultSet` there are three attributes you can set. These are:

1. Type
2. Concurrency
3. Holdability

You set these already when you create the `Statement` or `PreparedStatement`, like this:

```
Statement statement = connection.createStatement(
    ResultSet.TYPE_FORWARD_ONLY,
    ResultSet.CONCUR_READ_ONLY,
    ResultSet.CLOSE_CURSORS_OVER_COMMIT
   );

PreparedStatement statement = connection.prepareStatement(sql,
    ResultSet.TYPE_FORWARD_ONLY,
    ResultSet.CONCUR_READ_ONLY,
    ResultSet.CLOSE_CURSORS_OVER_COMMIT
   );
```

Precisely what these attributes mean is explained later in this text. But now you now where to specify them.

# Iterating the ResultSet

To iterate the `ResultSet` you use its `next()` method. The `next()` method returns true if the `ResultSet` has a next record, and moves the `ResultSet` to point to the next record. If there were no more records, `next()`returns false, and you can no longer. Once the `next()` method has returned false, you should not call it anymore. Doing so may result in an exception.

Here is an example of iterating a `ResultSet` using the `next()` method:

```
while(result.next()) {
    // ... get column values from this record
}
```

As you can see, the `next()` method is actually called before the first record is accessed. That means, that the `ResultSet` starts out pointing before the first record. Once `next()` has been called once, it points at the first record.

Similarly, when `next()` is called and returns false, the `ResultSet` is actually pointing after the last record.

You cannot obtain the number of rows in a `ResultSet` except if you iterate all the way through it and count the rows. However, if the `ResultSet` is forward-only, you cannot afterwards move backwards through it. Even if you could move backwards, it would a slow way of counting the rows in the `ResultSet`. You are better off structuring your code so that you do not need to know the number of records ahead of time.

# Accessing Column Values

When iterating the `ResultSet` you want to access the column values of each record. You do so by calling one or more of the many `getXXX()` methods. You pass the name of the column to get the value of, to the many `getXXX()` methods. For instance:

```
while(result.next()) {

    result.getString    ("name");
    result.getInt       ("age");
    result.getBigDecimal("coefficient");

    // etc.
}
```

There are a lot of `getXXX()` methods you can call, which return the value of the column as a

certain data type, e.g. String, int, long, double, BigDecimal etc. They all take the name of the column to obtain the column value for, as parameter.

The `getXXX()` methods also come in versions that take a column index instead of a column name. For instance:

```
while(result.next()) {

    result.getString    (1);
    result.getInt       (2);
    result.getBigDecimal(3);

    // etc.
}
```

The index of a column typically depends on the index of the column in the SQL statement. For instance, the SQL statement

```
select name, age, coefficient from person
```

has three columns. The column name is listed first, and will thus have index 1 in the `ResultSet`. The column age will have index 2, and the column coefficient will have index 3.

Sometimes you do not know the index of a certain column ahead of time. For instance, if you use a `select * from` type of SQL query, you do not know the sequence of the columns.

If you do not know the index of a certain column you can find the index of that column using the`ResultSet.findColumn(String columnName)` method, like this:

```
int nameIndex   = result.findColumn("name");
int ageIndex    = result.findColumn("age");
int coeffIndex  = result.findColumn("coefficient");

while(result.next()) {
    String     name        = result.getString     (nameIndex);
    int        age         = result.getInt        (ageIndex);
    BigDecimal coefficient = result.getBigDecimal (coeffIndex);
}
```

# ResultSet Types

A `ResultSet` can be of a certain type. The type determines some characteristics and abilities of the`ResultSet`.

Not all types are supported by all databases and JDBC drivers. You will have to check your database and JDBC driver to see if it supports the type you want to use.
The `DatabaseMetaData.supportsResultSetType(int type)` method returns true or false depending on whether the given type is supported or not. The`DatabaseMetaData` class is covered in a later text.

At the time of writing there are three `ResultSet` types:

1. ResultSet.TYPE_FORWARD_ONLY
2. ResultSet.TYPE_SCROLL_INSENSITIVE
3. ResultSet.TYPE_SCROLL_SENSITIVE

The default type is `TYPE_FORWARD_ONLY`

`TYPE_FORWARD_ONLY` means that the `ResultSet` can only be navigated forward. That is, you can only move from row 1, to row 2, to row 3 etc. You cannot move backwards in the `ResultSet`.

`TYPE_SCROLL_INSENSITIVE` means that the `ResultSet` can be navigated (scrolled) both forward and backwards. You can also jump to a position relative to the current position, or jump to an absolute position. The`ResultSet` is insensitive to changes in the underlying data source while the `ResultSet` is open. That is, if a record in the `ResultSet` is changed in the database by another thread or process, it will not be reflected in already opened `ResulsSet`'s of this type.

TYPE_SCROLL_SENSITIVE means that the ResultSet can be navigated (scrolled) both forward and backwards. You can also jump to a position relative to the current position, or jump to an absolute position. TheResultSet is sensitive to changes in the underlying data source while the ResultSet is open. That is, if a record in the ResultSet is changed in the database by another thread or process, it will be reflected in already opened ResulsSet's of this type.

### Navigation Methods

The ResultSet interface contains the following navigation methods. Remember, not all methods work with all ResultSet types. What methods works depends on your database, JDBC driver, and the ResultSet type.

| Method | Description |
|---|---|
| absolute() | Moves the ResultSet to point at an absolute position. The position is a row number passed as parameter to the absolute() method. |
| afterLast() | Moves the ResultSet to point after the last row in the ResultSet. |
| beforeFirst() | Moves the ResultSet to point before the first row in the ResultSet. |
| first() | Moves the ResultSet to point at the first row in the ResultSet. |
| last() | Moves the ResultSet to point at the last row in the ResultSet. |
| next() | Moves the ResultSet to point at the next row in the ResultSet. |
| previous() | Moves the ResultSet to point at the previous row in the ResultSet. |
| relative() | Moves the ResultSet to point to a position relative to its current position. The relative position is passed as a parameter to the relative method, and can be both positive and negative. |
| | Moves the ResultSet |

The ResultSet interface also contains a set of methods you can use to inquire about the current position of the ResultSet. These are:

| Method | Description |
|---|---|
| getRow() | Returns the row number of the current row - the row currently pointed to by theResultSet. |
| getType() | Returns the ResultSet type. |
| isAfterLast() | Returns true if the ResultSet points after the last row. False if not. |
| isBeforeFirst() | Returns true if the ResultSet points before the first row. False if not. |
| isFirst() | Returns true if the ResultSet points at the first row. False if not. |

Finally the ResultSet interface also contains a method to update a row with database changes, if theResultSet is sensitive to change.

| Method | Description |
|---|---|
| refreshRow() | Refreshes the column values of that row with the latest values from the database. |

## ResultSet Concurrency

The ResultSet concurrency determines whether the ResultSet can be updated, or only read.

Some databases and JDBC drivers support that the ResultSet is updated, but not all databases and JDBC drivers do. The DatabaseMetaData.supportsResultSetConcurrency(int concurrency) method returns true or false depending on whether the given concurrency mode is

supported or not. The `DatabaseMetaData` class is covered in a later text.

A `ResultSet` can have one of two concurrency levels:

1. ResultSet.CONCUR_READ_ONLY
2. ResultSet.CONCUR_UPDATABLE

CONCUR_READ_ONLY means that the `ResultSet` can only be read.

CONCUR_UPDATABLE means that the `ResultSet` can be both read and updated.

# Updating a ResultSet

If a `ResultSet` is updatable, you can update the columns of each row in the `ResultSet`. You do so using the many `updateXXX()` methods. For instance:

```
    result.updateString     ("name"       , "Alex");
    result.updateInt        ("age"        , 55);
    result.updateBigDecimal ("coefficient", new BigDecimal("0.1323");
    result.updateRow();
```

You can also update a column using column index instead of column name. Here is an example:

```
    result.updateString     (1, "Alex");
    result.updateInt        (2, 55);
    result.updateBigDecimal (3, new BigDecimal("0.1323");
    result.updateRow();
```

Notice the `updateRow()` call. It is when `updateRow()` is called that the database is updated with the values of the row. Id you do not call this method, the values updated in the `ResultSet` are never sent to the database. If you call `updateRow()` inside a transaction, the data is not actually committed to the database until the transaction is committed.

# Inserting Rows into a ResultSet

If the `ResultSet` is updatable it is also possible to insert rows into it. You do so by:

1. call ResultSet.moveToInsertRow()
2. update row column values
3. call ResultSet.insertRow()

Here is an example:

```
result.moveToInsertRow();
result.updateString     (1, "Alex");
result.updateInt        (2, 55);
result.updateBigDecimal (3, new BigDecimal("0.1323");
result.insertRow();

result.beforeFirst();
```

The row pointed to after calling `moveToInsertRow()` is a special row, a buffer, which you can use to build up the row until all column values has been set on the row.

Once the row is ready to be inserted into the `ResultSet`, call the `insertRow()` method.

After inserting the row the `ResultSet` still pointing to the insert row. However, you cannot be certain what will happen if you try to access it, once the row has been inserted. Therefore you should move the`ResultSet` to a valid position after inserting the new row. If you need to insert another row, explicitly call`moveToInsertRow()` to signal this to the `ResultSet`.

# ResultSet Holdability

The `ResultSet` holdability determines if a `ResultSet` is closed when the `commit()` method of the underlying`connection` is called.

Not all holdability modes are supported by all databases and JDBC drivers.
The`DatabaseMetaData.supportsResultSetHoldability(int holdability)` returns true or false depending on whether the given holdability mode is supported or not.
The `DatabaseMetaData` class is covered in a later text.

There are two types of holdability:

1. ResultSet.CLOSE_CURSORS_OVER_COMMIT
2. ResultSet.HOLD_CURSORS_OVER_COMMIT

The `CLOSE_CURSORS_OVER_COMMIT` holdability means that all `ResultSet` instances are closed when`connection.commit()` method is called on the connection that created the `ResultSet`.

The `HOLD_CURSORS_OVER_COMMIT` holdability means that the `ResultSet` is kept open when the`connection.commit()` method is called on the connection that created the `ResultSet`.

The `HOLD_CURSORS_OVER_COMMIT` holdability might be useful if you use the `ResultSet` to update values in the database. Thus, you can open a `ResultSet`, update rows in it,
call `connection.commit()` and still keep the same `ResultSet` open for future transactions on the same rows.