Step 3:

Scrollable and Updatable result set

By default, what ever the resultset we are getting, that is uni directional result set – always moves in forward direction

Statement st = con.createStatement()

OR

PreparedStatement pstmt = con.prepareStatement(Select-Query);

In order to create a scrollable and updatable result set, we have pass two additional parameters at the time creating either statement or prepared statement object.

**Example Using Statement**

Statement st=con.createStatement (

ResultSet.TYPE\_SCROLL\_INSENSITIVE,

ResultSet.CONCUR\_READ\_ONLY

);

**Example Using Prepared Statement**

PreparedStatement stmt = conn.prepareStatement(sql,

ResultSet.TYPE\_SCROLL\_INSENSITIVE, ResultSet.CONCUR\_UPDATABLE);

while creating the object of Statement or Prepared statement, we have to pass two arguments:

ResultSet.TYPE\_SCROLL\_INSENSITIVE,

ResultSet.CONCUR\_READ\_ONLY

First Argument to Scrollable and updatable resultset

|  |  |
| --- | --- |
| static int | [TYPE\_SCROLL\_INSENSITIVE](http://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html#TYPE_SCROLL_INSENSITIVE)  The constant indicating the type for a ResultSet object that is scrollable but generally not sensitive to changes to the data that underlies the ResultSet.  (modifications done in the database are not going to reflect in the resultset) |
| static int | [TYPE\_SCROLL\_SENSITIVE](http://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html#TYPE_SCROLL_SENSITIVE)  The constant indicating the type for a ResultSet object that is scrollable and generally sensitive to changes to the data that underlies the ResultSet.  (modifications done in the database are immediately reflected in the resultset) |

Second Argument to Scrollable and updatable resultset

|  |  |
| --- | --- |
| static int | [CLOSE\_CURSORS\_AT\_COMMIT](http://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html#CLOSE_CURSORS_AT_COMMIT)  The constant indicating that open ResultSet objects with this hold ability will be closed when the current transaction is commited. |
| static int | [CONCUR\_READ\_ONLY](http://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html#CONCUR_READ_ONLY)  The constant indicating the concurrency mode for a ResultSet object that may NOT be updated.  (result set object that allows only read operations on database table) |
| static int | [CONCUR\_UPDATABLE](http://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html#CONCUR_UPDATABLE)  The constant indicating the concurrency mode for a ResultSet object that may be updated.  (modifications done in the resultset object are directly reflected in a database table) |
| static int | [FETCH\_FORWARD](http://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html#FETCH_FORWARD)  The constant indicating that the rows in a result set will be processed in a forward direction; first-to-last. |
| static int | [FETCH\_REVERSE](http://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html#FETCH_REVERSE)  The constant indicating that the rows in a result set will be processed in a reverse direction; last-to-first. |
| static int | [FETCH\_UNKNOWN](http://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html#FETCH_UNKNOWN)  The constant indicating that the order in which rows in a result set will be processed is unknown. |
| static int | [HOLD\_CURSORS\_OVER\_COMMIT](http://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html#HOLD_CURSORS_OVER_COMMIT)  The constant indicating that open ResultSet objects with this hold ability will remain open when the current transaction is commited. |
| static int | [TYPE\_FORWARD\_ONLY](http://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html#TYPE_FORWARD_ONLY)  The constant indicating the type for a ResultSet object whose cursor may move only forward. |

Step 4:

ResultSet rs=st.executeQuery (“select \* from empleyee”);

-- or --

ResultSet rs = stmt.executeQuery();

Step 5: Processing the result set.

List of Methods available in Scrollable and Updatable Result set

* public boolean next ();

It returns true when rs contains next record otherwise false.

* public void beforeFirst ();

It is used for making the ResultSet object to point to just before the first record (it is by default)

* public boolean isFirst ();

It returns true when rs is pointing to first record otherwise false.

* public void first ();

It is used to point the ResultSet object to first record.

* public boolean isBeforeFirst ();

It returns true when rs pointing to before first record otherwise false.

* public boolean previous ();

It returns true when rs contains previous record otherwise false.

* public void afterLast ();

It is used for making the ResultSet object to point to just after the last record.

* public boolean isLast ();

It returns true when rs is pointing to last record otherwise false.

* public void last ();

It is used to point the ResultSet object to last record.

* public boolean isAfterLast ();

It returns true when rs is pointing after last record otherwise false.

* public void absolute (int);

It is used for moving the ResultSet object to a particular record either in forward direction or in backward direction with respect to first record and last record respectively. If int value is positive, rs move in forward direction to that with respect to first record. If int value is negative, rs move in backward direction to that with respect to last record.

* public void relative (int);

It is used for moving rs to that record either in forward direction or in backward direction with respect to current record.

**package** com.sssit.product.controller;

**import** java.sql.Connection;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** com.sssit.product.pojo.ProductPojo;

**import** com.sssit.product.util.JDBCConnection;

**public** **class** PaginationDemo {

**public** **static** **void** main(String[] args) {

Connection con = **null**;

PreparedStatement pstmt = **null**;

**try**

{

con=JDBCConnection.*getConnection*();

**final** String SQLQuery = "select \* from product";

pstmt = con.prepareStatement(SQLQuery,

ResultSet.***TYPE\_SCROLL\_INSENSITIVE***,

ResultSet.***CONCUR\_UPDATABLE***);

ResultSet rs = pstmt.executeQuery();

**while**(rs.next()) {

System.***out***.printf("%10d%20s%10.2f\n", rs.getInt(1),rs.getString(2),rs.getDouble(3));

}

// retreive the data in reverse order

// place the cursor at last of after last position

System.***out***.println("Data in Reverse Order....");

rs.last();

**do**{

System.***out***.printf("%10d%20s%10.2f\n", rs.getInt(1),rs.getString(2),rs.getDouble(3));

}**while**(rs.previous()) ;

System.***out***.println("Data in reverse order.....1....");

rs.afterLast();

**while**(rs.previous()){

System.***out***.printf("%10d%20s%10.2f\n", rs.getInt(1),rs.getString(2),rs.getDouble(3));

}

System.***out***.println("5th record.....");

rs.absolute(5);

System.***out***.printf("%10d%20s%10.2f\n", rs.getInt(1),rs.getString(2),rs.getDouble(3));

System.***out***.println("2nd next record from 5th record");

rs.relative(+2);

System.***out***.printf("%10d%20s%10.2f\n", rs.getInt(1),rs.getString(2),rs.getDouble(3));

System.***out***.println("3rd previous record from 7th record");

rs.relative(-3);

System.***out***.printf("%10d%20s%10.2f\n", rs.getInt(1),rs.getString(2),rs.getDouble(3));

}

**catch** (Exception e) {

// **TODO**: handle exception

}

}

}