| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/CachedRowSet.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/javax/sql/rowset/BaseRowSet.html)   [**NEXT CLASS**](http://docs.google.com/javax/sql/rowset/FilteredRowSet.html) | [**FRAMES**](http://docs.google.com/index.html?javax/sql/rowset/CachedRowSet.html)    [**NO FRAMES**](http://docs.google.com/CachedRowSet.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | CONSTR | [METHOD](#tyjcwt) | DETAIL: [FIELD](#17dp8vu) | CONSTR | [METHOD](#26in1rg) |

## **javax.sql.rowset**

Interface CachedRowSet

**All Superinterfaces:** [Joinable](http://docs.google.com/javax/sql/rowset/Joinable.html), [ResultSet](http://docs.google.com/java/sql/ResultSet.html), [RowSet](http://docs.google.com/javax/sql/RowSet.html), [Wrapper](http://docs.google.com/java/sql/Wrapper.html) **All Known Subinterfaces:** [FilteredRowSet](http://docs.google.com/javax/sql/rowset/FilteredRowSet.html), [JoinRowSet](http://docs.google.com/javax/sql/rowset/JoinRowSet.html), [WebRowSet](http://docs.google.com/javax/sql/rowset/WebRowSet.html)

public interface **CachedRowSet**extends [RowSet](http://docs.google.com/javax/sql/RowSet.html), [Joinable](http://docs.google.com/javax/sql/rowset/Joinable.html)

The interface that all standard implementations of CachedRowSet must implement.

The reference implementation of the CachedRowSet interface provided by Sun Microsystems is a standard implementation. Developers may use this implementation just as it is, they may extend it, or they may choose to write their own implementations of this interface.

A CachedRowSet object is a container for rows of data that caches its rows in memory, which makes it possible to operate without always being connected to its data source. Further, it is a JavaBeansTM component and is scrollable, updatable, and serializable. A CachedRowSet object typically contains rows from a result set, but it can also contain rows from any file with a tabular format, such as a spread sheet. The reference implementation supports getting data only from a ResultSet object, but developers can extend the SyncProvider implementations to provide access to other tabular data sources.

An application can modify the data in a CachedRowSet object, and those modifications can then be propagated back to the source of the data.

A CachedRowSet object is a *disconnected* rowset, which means that it makes use of a connection to its data source only briefly. It connects to its data source while it is reading data to populate itself with rows and again while it is propagating changes back to its underlying data source. The rest of the time, a CachedRowSet object is disconnected, including while its data is being modified. Being disconnected makes a RowSet object much leaner and therefore much easier to pass to another component. For example, a disconnected RowSet object can be serialized and passed over the wire to a thin client such as a personal digital assistant (PDA).

### 1.0 Creating a CachedRowSet Object

The following line of code uses the default constructor for CachedRowSet supplied in the reference implementation (RI) to create a default CachedRowSet object.

CachedRowSetImpl crs = new CachedRowSetImpl();

This new CachedRowSet object will have its properties set to the default properties of a BaseRowSet object, and, in addition, it will have an RIOptimisticProvider object as its synchronization provider. RIOptimisticProvider, one of two SyncProvider implementations included in the RI, is the default provider that the SyncFactory singleton will supply when no synchronization provider is specified.

A SyncProvider object provides a CachedRowSet object with a reader (a RowSetReader object) for reading data from a data source to populate itself with data. A reader can be implemented to read data from a ResultSet object or from a file with a tabular format. A SyncProvider object also provides a writer (a RowSetWriter object) for synchronizing any modifications to the CachedRowSet object's data made while it was disconnected with the data in the underlying data source.

A writer can be implemented to exercise various degrees of care in checking for conflicts and in avoiding them. (A conflict occurs when a value in the data source has been changed after the rowset populated itself with that value.) The RIOptimisticProvider implementation assumes there will be few or no conflicts and therefore sets no locks. It updates the data source with values from the CachedRowSet object only if there are no conflicts. Other writers can be implemented so that they always write modified data to the data source, which can be accomplished either by not checking for conflicts or, on the other end of the spectrum, by setting locks sufficient to prevent data in the data source from being changed. Still other writer implementations can be somewhere in between.

A CachedRowSet object may use any SyncProvider implementation that has been registered with the SyncFactory singleton. An application can find out which SyncProvider implementations have been registered by calling the following line of code.

java.util.Enumeration providers = SyncFactory.getRegisteredProviders();

There are two ways for a CachedRowSet object to specify which SyncProvider object it will use. Supplying the name of the implementation to the constructor

The following line of code creates the CachedRowSet object *crs2* that is initialized with default values except that its SyncProvider object is the one specified.

CachedRowSetImpl crs2 = new CachedRowSetImpl(  
 "com.fred.providers.HighAvailabilityProvider");

#### Setting the SyncProvider using the CachedRowSet method setSyncProvider The following line of code resets the SyncProvider object for *crs*, the CachedRowSet object created with the default constructor. crs.setSyncProvider("com.fred.providers.HighAvailabilityProvider"); SyncFactory and SyncProvider for more details. 2.0 Retrieving Data from a CachedRowSet ObjectData is retrieved from a CachedRowSet object by using the getter methods inherited from the ResultSet interface. The following examples, in which crs is a CachedRowSet object, demonstrate how to iterate through the rows, retrieving the column values in each row. The first example uses the version of the getter methods that take a column number; the second example uses the version that takes a column name. Column numbers are generally used when the RowSet object's command is of the form SELECT \* FROM TABLENAME; column names are most commonly used when the command specifies columns by name. while (crs.next()) { String name = crs.getString(1); int id = crs.getInt(2); Clob comment = crs.getClob(3); short dept = crs.getShort(4); System.out.println(name + " " + id + " " + comment + " " + dept); } while (crs.next()) { String name = crs.getString("NAME"); int id = crs.getInt("ID"); Clob comment = crs.getClob("COM"); short dept = crs.getShort("DEPT"); System.out.println(name + " " + id + " " + comment + " " + dept); } 2.1 Retrieving RowSetMetaDataAn application can get information about the columns in a CachedRowSet object by calling ResultSetMetaData and RowSetMetaData methods on a RowSetMetaData object. The following code fragment, in which *crs* is a CachedRowSet object, illustrates the process. The first line creates a RowSetMetaData object with information about the columns in *crs*. The method getMetaData, inherited from the ResultSet interface, returns a ResultSetMetaData object, which is cast to a RowSetMetaData object before being assigned to the variable *rsmd*. The second line finds out how many columns *jrs* has, and the third line gets the JDBC type of values stored in the second column of jrs. RowSetMetaData rsmd = (RowSetMetaData)crs.getMetaData(); int count = rsmd.getColumnCount(); int type = rsmd.getColumnType(2); The RowSetMetaData interface differs from the ResultSetMetaData interface in two ways.

* + *It includes setter methods:* A RowSet object uses these methods internally when it is populated with data from a different ResultSet object.
  + *It contains fewer getter methods:* Some ResultSetMetaData methods to not apply to a RowSet object. For example, methods retrieving whether a column value is writable or read only do not apply because all of a RowSet object's columns will be writable or read only, depending on whether the rowset is updatable or not.

### NOTE: In order to return a RowSetMetaData object, implementations must override the getMetaData() method defined in java.sql.ResultSet and return a RowSetMetaData object.3.0 Updating a CachedRowSet ObjectUpdating a CachedRowSet object is similar to updating a ResultSet object, but because the rowset is not connected to its data source while it is being updated, it must take an additional step to effect changes in its underlying data source. After calling the method updateRow or insertRow, a CachedRowSet object must also call the method acceptChanges to have updates written to the data source. The following example, in which the cursor is on a row in the CachedRowSet object *crs*, shows the code required to update two column values in the current row and also update the RowSet object's underlying data source. crs.updateShort(3, 58); crs.updateInt(4, 150000); crs.updateRow(); crs.acceptChanges(); The next example demonstrates moving to the insert row, building a new row on the insert row, inserting it into the rowset, and then calling the method acceptChanges to add the new row to the underlying data source. Note that as with the getter methods, the updater methods may take either a column index or a column name to designate the column being acted upon. crs.moveToInsertRow(); crs.updateString("Name", "Shakespeare"); crs.updateInt("ID", 10098347); crs.updateShort("Age", 58); crs.updateInt("Sal", 150000); crs.insertRow(); crs.moveToCurrentRow(); crs.acceptChanges(); NOTE: Where the insertRow() method inserts the contents of a CachedRowSet object's insert row is implementation-defined. The reference implementation for the CachedRowSet interface inserts a new row immediately following the current row, but it could be implemented to insert new rows in any number of other places.Another thing to note about these examples is how they use the method acceptChanges. It is this method that propagates changes in a CachedRowSet object back to the underlying data source, calling on the RowSet object's writer internally to write changes to the data source. To do this, the writer has to incur the expense of establishing a connection with that data source. The preceding two code fragments call the method acceptChanges immediately after calling updateRow or insertRow. However, when there are multiple rows being changed, it is more efficient to call acceptChanges after all calls to updateRow and insertRow have been made. If acceptChanges is called only once, only one connection needs to be established.4.0 Updating the Underlying Data SourceWhen the method acceptChanges is executed, the CachedRowSet object's writer, a RowSetWriterImpl object, is called behind the scenes to write the changes made to the rowset to the underlying data source. The writer is implemented to make a connection to the data source and write updates to it.A writer is made available through an implementation of the SyncProvider interface, as discussed in section 1, "Creating a CachedRowSet Object." The default reference implementation provider, RIOptimisticProvider, has its writer implemented to use an optimistic concurrency control mechanism. That is, it maintains no locks in the underlying database while the rowset is disconnected from the database and simply checks to see if there are any conflicts before writing data to the data source. If there are any conflicts, it does not write anything to the data source.The reader/writer facility provided by the SyncProvider class is pluggable, allowing for the customization of data retrieval and updating. If a different concurrency control mechanism is desired, a different implementation of SyncProvider can be plugged in using the method setSyncProvider.In order to use the optimistic concurrency control routine, the RIOptismisticProvider maintains both its current value and its original value (the value it had immediately preceding the current value). Note that if no changes have been made to the data in a RowSet object, its current values and its original values are the same, both being the values with which the RowSet object was initially populated. However, once any values in the RowSet object have been changed, the current values and the original values will be different, though at this stage, the original values are still the initial values. With any subsequent changes to data in a RowSet object, its original values and current values will still differ, but its original values will be the values that were previously the current values.Keeping track of original values allows the writer to compare the RowSet object's original value with the value in the database. If the values in the database differ from the RowSet object's original values, which means that the values in the database have been changed, there is a conflict. Whether a writer checks for conflicts, what degree of checking it does, and how it handles conflicts all depend on how it is implemented.5.0 Registering and Notifying ListenersBeing JavaBeans components, all rowsets participate in the JavaBeans event model, inheriting methods for registering listeners and notifying them of changes from the BaseRowSet class. A listener for a CachedRowSet object is a component that wants to be notified whenever there is a change in the rowset. For example, if a CachedRowSet object contains the results of a query and those results are being displayed in, say, a table and a bar graph, the table and bar graph could be registered as listeners with the rowset so that they can update themselves to reflect changes. To become listeners, the table and bar graph classes must implement the RowSetListener interface. Then they can be added to the CachedRowSet object's list of listeners, as is illustrated in the following lines of code. crs.addRowSetListener(table); crs.addRowSetListener(barGraph); Each CachedRowSet method that moves the cursor or changes data also notifies registered listeners of the changes, so table and barGraph will be notified when there is a change in crs.6.0 Passing Data to Thin ClientsOne of the main reasons to use a CachedRowSet object is to pass data between different components of an application. Because it is serializable, a CachedRowSet object can be used, for example, to send the result of a query executed by an enterprise JavaBeans component running in a server environment over a network to a client running in a web browser.While a CachedRowSet object is disconnected, it can be much leaner than a ResultSet object with the same data. As a result, it can be especially suitable for sending data to a thin client such as a PDA, where it would be inappropriate to use a JDBC driver due to resource limitations or security considerations. Thus, a CachedRowSet object provides a means to "get rows in" without the need to implement the full JDBC API.7.0 Scrolling and UpdatingA second major use for CachedRowSet objects is to provide scrolling and updating for ResultSet objects that do not provide these capabilities themselves. In other words, a CachedRowSet object can be used to augment the capabilities of a JDBC technology-enabled driver (hereafter called a "JDBC driver") when the DBMS does not provide full support for scrolling and updating. To achieve the effect of making a non-scrollble and read-only ResultSet object scrollable and updatable, a programmer simply needs to create a CachedRowSet object populated with that ResultSet object's data. This is demonstrated in the following code fragment, where stmt is a Statement object. ResultSet rs = stmt.executeQuery("SELECT \* FROM EMPLOYEES"); CachedRowSetImpl crs = new CachedRowSetImpl(); crs.populate(rs); The object crs now contains the data from the table EMPLOYEES, just as the object rs does. The difference is that the cursor for crs can be moved forward, backward, or to a particular row even if the cursor for rs can move only forward. In addition, crs is updatable even if rs is not because by default, a CachedRowSet object is both scrollable and updatable.In summary, a CachedRowSet object can be thought of as simply a disconnected set of rows that are being cached outside of a data source. Being thin and serializable, it can easily be sent across a wire, and it is well suited to sending data to a thin client. However, a CachedRowSet object does have a limitation: It is limited in size by the amount of data it can store in memory at one time.8.0 Getting Universal Data AccessAnother advantage of the CachedRowSet class is that it makes it possible to retrieve and store data from sources other than a relational database. The reader for a rowset can be implemented to read and populate its rowset with data from any tabular data source, including a spreadsheet or flat file. Because both a CachedRowSet object and its metadata can be created from scratch, a component that acts as a factory for rowsets can use this capability to create a rowset containing data from non-SQL data sources. Nevertheless, it is expected that most of the time, CachedRowSet objects will contain data that was fetched from an SQL database using the JDBC API.9.0 Setting PropertiesAll rowsets maintain a set of properties, which will usually be set using a tool. The number and kinds of properties a rowset has will vary, depending on what the rowset does and how it gets its data. For example, rowsets that get their data from a ResultSet object need to set the properties that are required for making a database connection. If a rowset uses the DriverManager facility to make a connection, it needs to set a property for the JDBC URL that identifies the appropriate driver, and it needs to set the properties that give the user name and password. If, on the other hand, the rowset uses a DataSource object to make the connection, which is the preferred method, it does not need to set the property for the JDBC URL. Instead, it needs to set properties for the logical name of the data source, for the user name, and for the password.NOTE: In order to use a DataSource object for making a connection, the DataSource object must have been registered with a naming service that uses the Java Naming and Directory InterfaceTM (JNDI) API. This registration is usually done by a person acting in the capacity of a system administrator.In order to be able to populate itself with data from a database, a rowset needs to set a command property. This property is a query that is a PreparedStatement object, which allows the query to have parameter placeholders that are set at run time, as opposed to design time. To set these placeholder parameters with values, a rowset provides setter methods for setting values of each data type, similar to the setter methods provided by the PreparedStatement interface.The following code fragment illustrates how the CachedRowSet object crs might have its command property set. Note that if a tool is used to set properties, this is the code that the tool would use. crs.setCommand("SELECT FIRST\_NAME, LAST\_NAME, ADDRESS FROM CUSTOMERS " + "WHERE CREDIT\_LIMIT > ? AND REGION = ?"); The values that will be used to set the command's placeholder parameters are contained in the RowSet object's params field, which is a Vector object. The CachedRowSet class provides a set of setter methods for setting the elements in its params field. The following code fragment demonstrates setting the two parameters in the query from the previous example. crs.setInt(1, 5000); crs.setString(2, "West"); The params field now contains two elements, each of which is an array two elements long. The first element is the parameter number; the second is the value to be set. In this case, the first element of params is 1, 5000, and the second element is 2, "West". When an application calls the method execute, it will in turn call on this RowSet object's reader, which will in turn invoke its readData method. As part of its implementation, readData will get the values in params and use them to set the command's placeholder parameters. The following code fragment gives an idea of how the reader does this, after obtaining the Connection object con. PreparedStatement pstmt = con.prepareStatement(crs.getCommand()); reader.decodeParams(); // decodeParams figures out which setter methods to use and does something // like the following: // for (i = 0; i < params.length; i++) { // pstmt.setObject(i + 1, params[i]); // } At this point, the command for crs is the query "SELECT FIRST\_NAME, LAST\_NAME, ADDRESS FROM CUSTOMERS WHERE CREDIT\_LIMIT > 5000 AND REGION = "West". After the readData method executes this command with the following line of code, it will have the data from rs with which to populate crs. ResultSet rs = pstmt.executeQuery(); The preceding code fragments give an idea of what goes on behind the scenes; they would not appear in an application, which would not invoke methods like readData and decodeParams. In contrast, the following code fragment shows what an application might do. It sets the rowset's command, sets the command's parameters, and executes the command. Simply by calling the execute method, crs populates itself with the requested data from the table CUSTOMERS. crs.setCommand("SELECT FIRST\_NAME, LAST\_NAME, ADDRESS FROM CUSTOMERS" + "WHERE CREDIT\_LIMIT > ? AND REGION = ?"); crs.setInt(1, 5000); crs.setString(2, "West"); crs.execute(); 10.0 Paging DataBecause a CachedRowSet object stores data in memory, the amount of data that it can contain at any one time is determined by the amount of memory available. To get around this limitation, a CachedRowSet object can retrieve data from a ResultSet object in chunks of data, called *pages*. To take advantage of this mechanism, an application sets the number of rows to be included in a page using the method setPageSize. In other words, if the page size is set to five, a chunk of five rows of data will be fetched from the data source at one time. An application can also optionally set the maximum number of rows that may be fetched at one time. If the maximum number of rows is set to zero, or no maximum number of rows is set, there is no limit to the number of rows that may be fetched at a time.After properties have been set, the CachedRowSet object must be populated with data using either the method populate or the method execute. The following lines of code demonstrate using the method populate. Note that this version of the method takes two parameters, a ResultSet handle and the row in the ResultSet object from which to start retrieving rows. CachedRowSet crs = new CachedRowSetImpl(); crs.setMaxRows(20); crs.setPageSize(4); crs.populate(rsHandle, 10); When this code runs, *crs* will be populated with four rows from *rsHandle* starting with the tenth row.The next code fragment shows populating a CachedRowSet object using the method execute, which may or may not take a Connection object as a parameter. This code passes execute the Connection object *conHandle*.Note that there are two differences between the following code fragment and the previous one. First, the method setMaxRows is not called, so there is no limit set for the number of rows that *crs* may contain. (Remember that *crs* always has the overriding limit of how much data it can store in memory.) The second difference is that the you cannot pass the method execute the number of the row in the ResultSet object from which to start retrieving rows. This method always starts with the first row. CachedRowSet crs = new CachedRowSetImpl(); crs.setPageSize(5); crs.execute(conHandle); After this code has run, *crs* will contain five rows of data from the ResultSet object produced by the command for *crs*. The writer for *crs* will use *conHandle* to connect to the data source and execute the command for *crs*. An application is then able to operate on the data in *crs* in the same way that it would operate on data in any other CachedRowSet object.To access the next page (chunk of data), an application calls the method nextPage. This method creates a new CachedRowSet object and fills it with the next page of data. For example, assume that the CachedRowSet object's command returns a ResultSet object *rs* with 1000 rows of data. If the page size has been set to 100, the first call to the method nextPage will create a CachedRowSet object containing the first 100 rows of *rs*. After doing what it needs to do with the data in these first 100 rows, the application can again call the method nextPage to create another CachedRowSet object with the second 100 rows from *rs*. The data from the first CachedRowSet object will no longer be in memory because it is replaced with the data from the second CachedRowSet object. After the tenth call to the method nextPage, the tenth CachedRowSet object will contain the last 100 rows of data from *rs*, which are stored in memory. At any given time, the data from only one CachedRowSet object is stored in memory.The method nextPage returns true as long as the current page is not the last page of rows and false when there are no more pages. It can therefore be used in a while loop to retrieve all of the pages, as is demonstrated in the following lines of code. CachedRowSet crs = CachedRowSetImpl(); crs.setPageSize(100); crs.execute(conHandle); while(crs.nextPage()) { while(crs.next()) { . . . // operate on chunks (of 100 rows each) in crs, // row by row } } After this code fragment has been run, the application will have traversed all 1000 rows, but it will have had no more than 100 rows in memory at a time.The CachedRowSet interface also defines the method previousPage. Just as the method nextPage is analogous to the ResultSet method next, the method previousPage is analogous to the ResultSet method previous. Similar to the method nextPage, previousPage creates a CachedRowSet object containing the number of rows set as the page size. So, for instance, the method previousPage could be used in a while loop at the end of the preceding code fragment to navigate back through the pages from the last page to the first page. The method previousPage is also similar to nextPage in that it can be used in a while loop, except that it returns true as long as there is another page preceding it and false when there are no more pages ahead of it.By positioning the cursor after the last row for each page, as is done in the following code fragment, the method previous navigates from the last row to the first row in each page. The code could also have left the cursor before the first row on each page and then used the method next in a while loop to navigate each page from the first row to the last row.The following code fragment assumes a continuation from the previous code fragment, meaning that the cursor for the tenth CachedRowSet object is on the last row. The code moves the cursor to after the last row so that the first call to the method previous will put the cursor back on the last row. After going through all of the rows in the last page (the CachedRowSet object *crs*), the code then enters the while loop to get to the ninth page, go through the rows backwards, go to the eighth page, go through the rows backwards, and so on to the first row of the first page. crs.afterLast(); while(crs.previous()) { . . . // navigate through the rows, last to first { while(crs.previousPage()) { crs.afterLast(); while(crs.previous()) { . . . // go from the last row to the first row of each page } }

| **Field Summary** | |
| --- | --- |
| static boolean | [**COMMIT\_ON\_ACCEPT\_CHANGES**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#COMMIT_ON_ACCEPT_CHANGES)           Causes the CachedRowSet object's SyncProvider to commit the changes when acceptChanges() is called. |

| **Fields inherited from interface java.sql.**[**ResultSet**](http://docs.google.com/java/sql/ResultSet.html) |
| --- |
| [CLOSE\_CURSORS\_AT\_COMMIT](http://docs.google.com/java/sql/ResultSet.html#CLOSE_CURSORS_AT_COMMIT), [CONCUR\_READ\_ONLY](http://docs.google.com/java/sql/ResultSet.html#CONCUR_READ_ONLY), [CONCUR\_UPDATABLE](http://docs.google.com/java/sql/ResultSet.html#CONCUR_UPDATABLE), [FETCH\_FORWARD](http://docs.google.com/java/sql/ResultSet.html#FETCH_FORWARD), [FETCH\_REVERSE](http://docs.google.com/java/sql/ResultSet.html#FETCH_REVERSE), [FETCH\_UNKNOWN](http://docs.google.com/java/sql/ResultSet.html#FETCH_UNKNOWN), [HOLD\_CURSORS\_OVER\_COMMIT](http://docs.google.com/java/sql/ResultSet.html#HOLD_CURSORS_OVER_COMMIT), [TYPE\_FORWARD\_ONLY](http://docs.google.com/java/sql/ResultSet.html#TYPE_FORWARD_ONLY), [TYPE\_SCROLL\_INSENSITIVE](http://docs.google.com/java/sql/ResultSet.html#TYPE_SCROLL_INSENSITIVE), [TYPE\_SCROLL\_SENSITIVE](http://docs.google.com/java/sql/ResultSet.html#TYPE_SCROLL_SENSITIVE) |

| **Method Summary** | |
| --- | --- |
| void | [**acceptChanges**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#acceptChanges())()           Propagates row update, insert and delete changes made to this CachedRowSet object to the underlying data source. |
| void | [**acceptChanges**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#acceptChanges(java.sql.Connection))([Connection](http://docs.google.com/java/sql/Connection.html) con)           Propagates all row update, insert and delete changes to the data source backing this CachedRowSet object using the specified Connection object to establish a connection to the data source. |
| boolean | [**columnUpdated**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#columnUpdated(int))(int idx)           Indicates whether the designated column in the current row of this CachedRowSet object has been updated. |
| boolean | [**columnUpdated**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#columnUpdated(java.lang.String))([String](http://docs.google.com/java/lang/String.html) columnName)           Indicates whether the designated column in the current row of this CachedRowSet object has been updated. |
| void | [**commit**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#commit())()           Each CachedRowSet object's SyncProvider contains a Connection object from the ResultSet or JDBC properties passed to it's constructors. |
| [CachedRowSet](http://docs.google.com/javax/sql/rowset/CachedRowSet.html) | [**createCopy**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#createCopy())()           Creates a RowSet object that is a deep copy of the data in this CachedRowSet object. |
| [CachedRowSet](http://docs.google.com/javax/sql/rowset/CachedRowSet.html) | [**createCopyNoConstraints**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#createCopyNoConstraints())()           Creates a CachedRowSet object that is a deep copy of this CachedRowSet object's data but is independent of it. |
| [CachedRowSet](http://docs.google.com/javax/sql/rowset/CachedRowSet.html) | [**createCopySchema**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#createCopySchema())()           Creates a CachedRowSet object that is an empty copy of this CachedRowSet object. |
| [RowSet](http://docs.google.com/javax/sql/RowSet.html) | [**createShared**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#createShared())()           Returns a new RowSet object backed by the same data as that of this CachedRowSet object. |
| void | [**execute**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#execute(java.sql.Connection))([Connection](http://docs.google.com/java/sql/Connection.html) conn)           Populates this CachedRowSet object with data, using the given connection to produce the result set from which the data will be read. |
| int[] | [**getKeyColumns**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#getKeyColumns())()           Returns an array containing one or more column numbers indicating the columns that form a key that uniquely identifies a row in this CachedRowSet object. |
| [ResultSet](http://docs.google.com/java/sql/ResultSet.html) | [**getOriginal**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#getOriginal())()           Returns a ResultSet object containing the original value of this CachedRowSet object. |
| [ResultSet](http://docs.google.com/java/sql/ResultSet.html) | [**getOriginalRow**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#getOriginalRow())()           Returns a ResultSet object containing the original value for the current row only of this CachedRowSet object. |
| int | [**getPageSize**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#getPageSize())()           Returns the page-size for the CachedRowSet object |
| [RowSetWarning](http://docs.google.com/javax/sql/rowset/RowSetWarning.html) | [**getRowSetWarnings**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#getRowSetWarnings())()           Retrieves the first warning reported by calls on this RowSet object. |
| boolean | [**getShowDeleted**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#getShowDeleted())()           Retrieves a boolean indicating whether rows marked for deletion appear in the set of current rows. |
| [SyncProvider](http://docs.google.com/javax/sql/rowset/spi/SyncProvider.html) | [**getSyncProvider**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#getSyncProvider())()           Retrieves the SyncProvider implementation for this CachedRowSet object. |
| [String](http://docs.google.com/java/lang/String.html) | [**getTableName**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#getTableName())()           Returns an identifier for the object (table) that was used to create this CachedRowSet object. |
| boolean | [**nextPage**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#nextPage())()           Increments the current page of the CachedRowSet. |
| void | [**populate**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#populate(java.sql.ResultSet))([ResultSet](http://docs.google.com/java/sql/ResultSet.html) data)           Populates this CachedRowSet object with data from the given ResultSet object. |
| void | [**populate**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#populate(java.sql.ResultSet,%20int))([ResultSet](http://docs.google.com/java/sql/ResultSet.html) rs, int startRow)           Populates this CachedRowSet object with data from the given ResultSet object. |
| boolean | [**previousPage**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#previousPage())()           Decrements the current page of the CachedRowSet. |
| void | [**release**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#release())()           Releases the current contents of this CachedRowSet object and sends a rowSetChanged event to all registered listeners. |
| void | [**restoreOriginal**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#restoreOriginal())()           Restores this CachedRowSet object to its original value, that is, its value before the last set of changes. |
| void | [**rollback**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#rollback())()           Each CachedRowSet object's SyncProvider contains a Connection object from the original ResultSet or JDBC properties passed to it. |
| void | [**rollback**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#rollback(java.sql.Savepoint))([Savepoint](http://docs.google.com/java/sql/Savepoint.html) s)           Each CachedRowSet object's SyncProvider contains a Connection object from the original ResultSet or JDBC properties passed to it. |
| void | [**rowSetPopulated**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#rowSetPopulated(javax.sql.RowSetEvent,%20int))([RowSetEvent](http://docs.google.com/javax/sql/RowSetEvent.html) event, int numRows)           Notifies registered listeners that a RowSet object in the given RowSetEvent object has populated a number of additional rows. |
| void | [**setKeyColumns**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#setKeyColumns(int%5B%5D))(int[] keys)           Sets this CachedRowSet object's keyCols field with the given array of column numbers, which forms a key for uniquely identifying a row in this CachedRowSet object. |
| void | [**setMetaData**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#setMetaData(javax.sql.RowSetMetaData))([RowSetMetaData](http://docs.google.com/javax/sql/RowSetMetaData.html) md)           Sets the metadata for this CachedRowSet object with the given RowSetMetaData object. |
| void | [**setOriginalRow**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#setOriginalRow())()           Sets the current row in this CachedRowSet object as the original row. |
| void | [**setPageSize**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#setPageSize(int))(int size)           Sets the CachedRowSet object's page-size. |
| void | [**setShowDeleted**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#setShowDeleted(boolean))(boolean b)           Sets the property showDeleted to the given boolean value, which determines whether rows marked for deletion appear in the set of current rows. |
| void | [**setSyncProvider**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#setSyncProvider(java.lang.String))([String](http://docs.google.com/java/lang/String.html) provider)           Sets the SyncProvider objec for this CachedRowSet object to the one specified. |
| void | [**setTableName**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#setTableName(java.lang.String))([String](http://docs.google.com/java/lang/String.html) tabName)           Sets the identifier for the table from which this CachedRowSet object was derived to the given table name. |
| int | [**size**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#size())()           Returns the number of rows in this CachedRowSet object. |
| [Collection](http://docs.google.com/java/util/Collection.html)<?> | [**toCollection**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#toCollection())()           Converts this CachedRowSet object to a Collection object that contains all of this CachedRowSet object's data. |
| [Collection](http://docs.google.com/java/util/Collection.html)<?> | [**toCollection**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#toCollection(int))(int column)           Converts the designated column in this CachedRowSet object to a Collection object. |
| [Collection](http://docs.google.com/java/util/Collection.html)<?> | [**toCollection**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#toCollection(java.lang.String))([String](http://docs.google.com/java/lang/String.html) column)           Converts the designated column in this CachedRowSet object to a Collection object. |
| void | [**undoDelete**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#undoDelete())()           Cancels the deletion of the current row and notifies listeners that a row has changed. |
| void | [**undoInsert**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#undoInsert())()           Immediately removes the current row from this CachedRowSet object if the row has been inserted, and also notifies listeners that a row has changed. |
| void | [**undoUpdate**](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#undoUpdate())()           Immediately reverses the last update operation if the row has been modified. |

| **Methods inherited from interface javax.sql.**[**RowSet**](http://docs.google.com/javax/sql/RowSet.html) |
| --- |
| [addRowSetListener](http://docs.google.com/javax/sql/RowSet.html#addRowSetListener(javax.sql.RowSetListener)), [clearParameters](http://docs.google.com/javax/sql/RowSet.html#clearParameters()), [execute](http://docs.google.com/javax/sql/RowSet.html#execute()), [getCommand](http://docs.google.com/javax/sql/RowSet.html#getCommand()), [getDataSourceName](http://docs.google.com/javax/sql/RowSet.html#getDataSourceName()), [getEscapeProcessing](http://docs.google.com/javax/sql/RowSet.html#getEscapeProcessing()), [getMaxFieldSize](http://docs.google.com/javax/sql/RowSet.html#getMaxFieldSize()), [getMaxRows](http://docs.google.com/javax/sql/RowSet.html#getMaxRows()), [getPassword](http://docs.google.com/javax/sql/RowSet.html#getPassword()), [getQueryTimeout](http://docs.google.com/javax/sql/RowSet.html#getQueryTimeout()), [getTransactionIsolation](http://docs.google.com/javax/sql/RowSet.html#getTransactionIsolation()), [getTypeMap](http://docs.google.com/javax/sql/RowSet.html#getTypeMap()), [getUrl](http://docs.google.com/javax/sql/RowSet.html#getUrl()), [getUsername](http://docs.google.com/javax/sql/RowSet.html#getUsername()), [isReadOnly](http://docs.google.com/javax/sql/RowSet.html#isReadOnly()), [removeRowSetListener](http://docs.google.com/javax/sql/RowSet.html#removeRowSetListener(javax.sql.RowSetListener)), [setArray](http://docs.google.com/javax/sql/RowSet.html#setArray(int,%20java.sql.Array)), [setAsciiStream](http://docs.google.com/javax/sql/RowSet.html#setAsciiStream(int,%20java.io.InputStream)), [setAsciiStream](http://docs.google.com/javax/sql/RowSet.html#setAsciiStream(int,%20java.io.InputStream,%20int)), [setAsciiStream](http://docs.google.com/javax/sql/RowSet.html#setAsciiStream(java.lang.String,%20java.io.InputStream)), [setAsciiStream](http://docs.google.com/javax/sql/RowSet.html#setAsciiStream(java.lang.String,%20java.io.InputStream,%20int)), [setBigDecimal](http://docs.google.com/javax/sql/RowSet.html#setBigDecimal(int,%20java.math.BigDecimal)), [setBigDecimal](http://docs.google.com/javax/sql/RowSet.html#setBigDecimal(java.lang.String,%20java.math.BigDecimal)), [setBinaryStream](http://docs.google.com/javax/sql/RowSet.html#setBinaryStream(int,%20java.io.InputStream)), [setBinaryStream](http://docs.google.com/javax/sql/RowSet.html#setBinaryStream(int,%20java.io.InputStream,%20int)), [setBinaryStream](http://docs.google.com/javax/sql/RowSet.html#setBinaryStream(java.lang.String,%20java.io.InputStream)), [setBinaryStream](http://docs.google.com/javax/sql/RowSet.html#setBinaryStream(java.lang.String,%20java.io.InputStream,%20int)), [setBlob](http://docs.google.com/javax/sql/RowSet.html#setBlob(int,%20java.sql.Blob)), [setBlob](http://docs.google.com/javax/sql/RowSet.html#setBlob(int,%20java.io.InputStream)), [setBlob](http://docs.google.com/javax/sql/RowSet.html#setBlob(int,%20java.io.InputStream,%20long)), [setBlob](http://docs.google.com/javax/sql/RowSet.html#setBlob(java.lang.String,%20java.sql.Blob)), [setBlob](http://docs.google.com/javax/sql/RowSet.html#setBlob(java.lang.String,%20java.io.InputStream)), [setBlob](http://docs.google.com/javax/sql/RowSet.html#setBlob(java.lang.String,%20java.io.InputStream,%20long)), [setBoolean](http://docs.google.com/javax/sql/RowSet.html#setBoolean(int,%20boolean)), [setBoolean](http://docs.google.com/javax/sql/RowSet.html#setBoolean(java.lang.String,%20boolean)), [setByte](http://docs.google.com/javax/sql/RowSet.html#setByte(int,%20byte)), [setByte](http://docs.google.com/javax/sql/RowSet.html#setByte(java.lang.String,%20byte)), [setBytes](http://docs.google.com/javax/sql/RowSet.html#setBytes(int,%20byte%5B%5D)), [setBytes](http://docs.google.com/javax/sql/RowSet.html#setBytes(java.lang.String,%20byte%5B%5D)), [setCharacterStream](http://docs.google.com/javax/sql/RowSet.html#setCharacterStream(int,%20java.io.Reader)), [setCharacterStream](http://docs.google.com/javax/sql/RowSet.html#setCharacterStream(int,%20java.io.Reader,%20int)), [setCharacterStream](http://docs.google.com/javax/sql/RowSet.html#setCharacterStream(java.lang.String,%20java.io.Reader)), [setCharacterStream](http://docs.google.com/javax/sql/RowSet.html#setCharacterStream(java.lang.String,%20java.io.Reader,%20int)), [setClob](http://docs.google.com/javax/sql/RowSet.html#setClob(int,%20java.sql.Clob)), [setClob](http://docs.google.com/javax/sql/RowSet.html#setClob(int,%20java.io.Reader)), [setClob](http://docs.google.com/javax/sql/RowSet.html#setClob(int,%20java.io.Reader,%20long)), [setClob](http://docs.google.com/javax/sql/RowSet.html#setClob(java.lang.String,%20java.sql.Clob)), [setClob](http://docs.google.com/javax/sql/RowSet.html#setClob(java.lang.String,%20java.io.Reader)), [setClob](http://docs.google.com/javax/sql/RowSet.html#setClob(java.lang.String,%20java.io.Reader,%20long)), [setCommand](http://docs.google.com/javax/sql/RowSet.html#setCommand(java.lang.String)), [setConcurrency](http://docs.google.com/javax/sql/RowSet.html#setConcurrency(int)), [setDataSourceName](http://docs.google.com/javax/sql/RowSet.html#setDataSourceName(java.lang.String)), [setDate](http://docs.google.com/javax/sql/RowSet.html#setDate(int,%20java.sql.Date)), [setDate](http://docs.google.com/javax/sql/RowSet.html#setDate(int,%20java.sql.Date,%20java.util.Calendar)), [setDate](http://docs.google.com/javax/sql/RowSet.html#setDate(java.lang.String,%20java.sql.Date)), [setDate](http://docs.google.com/javax/sql/RowSet.html#setDate(java.lang.String,%20java.sql.Date,%20java.util.Calendar)), [setDouble](http://docs.google.com/javax/sql/RowSet.html#setDouble(int,%20double)), [setDouble](http://docs.google.com/javax/sql/RowSet.html#setDouble(java.lang.String,%20double)), [setEscapeProcessing](http://docs.google.com/javax/sql/RowSet.html#setEscapeProcessing(boolean)), [setFloat](http://docs.google.com/javax/sql/RowSet.html#setFloat(int,%20float)), [setFloat](http://docs.google.com/javax/sql/RowSet.html#setFloat(java.lang.String,%20float)), [setInt](http://docs.google.com/javax/sql/RowSet.html#setInt(int,%20int)), [setInt](http://docs.google.com/javax/sql/RowSet.html#setInt(java.lang.String,%20int)), [setLong](http://docs.google.com/javax/sql/RowSet.html#setLong(int,%20long)), [setLong](http://docs.google.com/javax/sql/RowSet.html#setLong(java.lang.String,%20long)), [setMaxFieldSize](http://docs.google.com/javax/sql/RowSet.html#setMaxFieldSize(int)), [setMaxRows](http://docs.google.com/javax/sql/RowSet.html#setMaxRows(int)), [setNCharacterStream](http://docs.google.com/javax/sql/RowSet.html#setNCharacterStream(int,%20java.io.Reader)), [setNCharacterStream](http://docs.google.com/javax/sql/RowSet.html#setNCharacterStream(int,%20java.io.Reader,%20long)), [setNCharacterStream](http://docs.google.com/javax/sql/RowSet.html#setNCharacterStream(java.lang.String,%20java.io.Reader)), [setNCharacterStream](http://docs.google.com/javax/sql/RowSet.html#setNCharacterStream(java.lang.String,%20java.io.Reader,%20long)), [setNClob](http://docs.google.com/javax/sql/RowSet.html#setNClob(int,%20java.sql.NClob)), [setNClob](http://docs.google.com/javax/sql/RowSet.html#setNClob(int,%20java.io.Reader)), [setNClob](http://docs.google.com/javax/sql/RowSet.html#setNClob(int,%20java.io.Reader,%20long)), [setNClob](http://docs.google.com/javax/sql/RowSet.html#setNClob(java.lang.String,%20java.sql.NClob)), [setNClob](http://docs.google.com/javax/sql/RowSet.html#setNClob(java.lang.String,%20java.io.Reader)), [setNClob](http://docs.google.com/javax/sql/RowSet.html#setNClob(java.lang.String,%20java.io.Reader,%20long)), [setNString](http://docs.google.com/javax/sql/RowSet.html#setNString(int,%20java.lang.String)), [setNString](http://docs.google.com/javax/sql/RowSet.html#setNString(java.lang.String,%20java.lang.String)), [setNull](http://docs.google.com/javax/sql/RowSet.html#setNull(int,%20int)), [setNull](http://docs.google.com/javax/sql/RowSet.html#setNull(int,%20int,%20java.lang.String)), [setNull](http://docs.google.com/javax/sql/RowSet.html#setNull(java.lang.String,%20int)), [setNull](http://docs.google.com/javax/sql/RowSet.html#setNull(java.lang.String,%20int,%20java.lang.String)), [setObject](http://docs.google.com/javax/sql/RowSet.html#setObject(int,%20java.lang.Object)), [setObject](http://docs.google.com/javax/sql/RowSet.html#setObject(int,%20java.lang.Object,%20int)), [setObject](http://docs.google.com/javax/sql/RowSet.html#setObject(int,%20java.lang.Object,%20int,%20int)), [setObject](http://docs.google.com/javax/sql/RowSet.html#setObject(java.lang.String,%20java.lang.Object)), [setObject](http://docs.google.com/javax/sql/RowSet.html#setObject(java.lang.String,%20java.lang.Object,%20int)), [setObject](http://docs.google.com/javax/sql/RowSet.html#setObject(java.lang.String,%20java.lang.Object,%20int,%20int)), [setPassword](http://docs.google.com/javax/sql/RowSet.html#setPassword(java.lang.String)), [setQueryTimeout](http://docs.google.com/javax/sql/RowSet.html#setQueryTimeout(int)), [setReadOnly](http://docs.google.com/javax/sql/RowSet.html#setReadOnly(boolean)), [setRef](http://docs.google.com/javax/sql/RowSet.html#setRef(int,%20java.sql.Ref)), [setRowId](http://docs.google.com/javax/sql/RowSet.html#setRowId(int,%20java.sql.RowId)), [setRowId](http://docs.google.com/javax/sql/RowSet.html#setRowId(java.lang.String,%20java.sql.RowId)), [setShort](http://docs.google.com/javax/sql/RowSet.html#setShort(int,%20short)), [setShort](http://docs.google.com/javax/sql/RowSet.html#setShort(java.lang.String,%20short)), [setSQLXML](http://docs.google.com/javax/sql/RowSet.html#setSQLXML(int,%20java.sql.SQLXML)), [setSQLXML](http://docs.google.com/javax/sql/RowSet.html#setSQLXML(java.lang.String,%20java.sql.SQLXML)), [setString](http://docs.google.com/javax/sql/RowSet.html#setString(int,%20java.lang.String)), [setString](http://docs.google.com/javax/sql/RowSet.html#setString(java.lang.String,%20java.lang.String)), [setTime](http://docs.google.com/javax/sql/RowSet.html#setTime(int,%20java.sql.Time)), [setTime](http://docs.google.com/javax/sql/RowSet.html#setTime(int,%20java.sql.Time,%20java.util.Calendar)), [setTime](http://docs.google.com/javax/sql/RowSet.html#setTime(java.lang.String,%20java.sql.Time)), [setTime](http://docs.google.com/javax/sql/RowSet.html#setTime(java.lang.String,%20java.sql.Time,%20java.util.Calendar)), [setTimestamp](http://docs.google.com/javax/sql/RowSet.html#setTimestamp(int,%20java.sql.Timestamp)), [setTimestamp](http://docs.google.com/javax/sql/RowSet.html#setTimestamp(int,%20java.sql.Timestamp,%20java.util.Calendar)), [setTimestamp](http://docs.google.com/javax/sql/RowSet.html#setTimestamp(java.lang.String,%20java.sql.Timestamp)), [setTimestamp](http://docs.google.com/javax/sql/RowSet.html#setTimestamp(java.lang.String,%20java.sql.Timestamp,%20java.util.Calendar)), [setTransactionIsolation](http://docs.google.com/javax/sql/RowSet.html#setTransactionIsolation(int)), [setType](http://docs.google.com/javax/sql/RowSet.html#setType(int)), [setTypeMap](http://docs.google.com/javax/sql/RowSet.html#setTypeMap(java.util.Map)), [setURL](http://docs.google.com/javax/sql/RowSet.html#setURL(int,%20java.net.URL)), [setUrl](http://docs.google.com/javax/sql/RowSet.html#setUrl(java.lang.String)), [setUsername](http://docs.google.com/javax/sql/RowSet.html#setUsername(java.lang.String)) |

| **Methods inherited from interface java.sql.**[**ResultSet**](http://docs.google.com/java/sql/ResultSet.html) |
| --- |
| [absolute](http://docs.google.com/java/sql/ResultSet.html#absolute(int)), [afterLast](http://docs.google.com/java/sql/ResultSet.html#afterLast()), [beforeFirst](http://docs.google.com/java/sql/ResultSet.html#beforeFirst()), [cancelRowUpdates](http://docs.google.com/java/sql/ResultSet.html#cancelRowUpdates()), [clearWarnings](http://docs.google.com/java/sql/ResultSet.html#clearWarnings()), [close](http://docs.google.com/java/sql/ResultSet.html#close()), [deleteRow](http://docs.google.com/java/sql/ResultSet.html#deleteRow()), [findColumn](http://docs.google.com/java/sql/ResultSet.html#findColumn(java.lang.String)), [first](http://docs.google.com/java/sql/ResultSet.html#first()), [getArray](http://docs.google.com/java/sql/ResultSet.html#getArray(int)), [getArray](http://docs.google.com/java/sql/ResultSet.html#getArray(java.lang.String)), [getAsciiStream](http://docs.google.com/java/sql/ResultSet.html#getAsciiStream(int)), [getAsciiStream](http://docs.google.com/java/sql/ResultSet.html#getAsciiStream(java.lang.String)), [getBigDecimal](http://docs.google.com/java/sql/ResultSet.html#getBigDecimal(int)), [getBigDecimal](http://docs.google.com/java/sql/ResultSet.html#getBigDecimal(int,%20int)), [getBigDecimal](http://docs.google.com/java/sql/ResultSet.html#getBigDecimal(java.lang.String)), [getBigDecimal](http://docs.google.com/java/sql/ResultSet.html#getBigDecimal(java.lang.String,%20int)), [getBinaryStream](http://docs.google.com/java/sql/ResultSet.html#getBinaryStream(int)), [getBinaryStream](http://docs.google.com/java/sql/ResultSet.html#getBinaryStream(java.lang.String)), [getBlob](http://docs.google.com/java/sql/ResultSet.html#getBlob(int)), [getBlob](http://docs.google.com/java/sql/ResultSet.html#getBlob(java.lang.String)), [getBoolean](http://docs.google.com/java/sql/ResultSet.html#getBoolean(int)), [getBoolean](http://docs.google.com/java/sql/ResultSet.html#getBoolean(java.lang.String)), [getByte](http://docs.google.com/java/sql/ResultSet.html#getByte(int)), [getByte](http://docs.google.com/java/sql/ResultSet.html#getByte(java.lang.String)), [getBytes](http://docs.google.com/java/sql/ResultSet.html#getBytes(int)), [getBytes](http://docs.google.com/java/sql/ResultSet.html#getBytes(java.lang.String)), [getCharacterStream](http://docs.google.com/java/sql/ResultSet.html#getCharacterStream(int)), [getCharacterStream](http://docs.google.com/java/sql/ResultSet.html#getCharacterStream(java.lang.String)), [getClob](http://docs.google.com/java/sql/ResultSet.html#getClob(int)), [getClob](http://docs.google.com/java/sql/ResultSet.html#getClob(java.lang.String)), [getConcurrency](http://docs.google.com/java/sql/ResultSet.html#getConcurrency()), [getCursorName](http://docs.google.com/java/sql/ResultSet.html#getCursorName()), [getDate](http://docs.google.com/java/sql/ResultSet.html#getDate(int)), [getDate](http://docs.google.com/java/sql/ResultSet.html#getDate(int,%20java.util.Calendar)), [getDate](http://docs.google.com/java/sql/ResultSet.html#getDate(java.lang.String)), [getDate](http://docs.google.com/java/sql/ResultSet.html#getDate(java.lang.String,%20java.util.Calendar)), [getDouble](http://docs.google.com/java/sql/ResultSet.html#getDouble(int)), [getDouble](http://docs.google.com/java/sql/ResultSet.html#getDouble(java.lang.String)), [getFetchDirection](http://docs.google.com/java/sql/ResultSet.html#getFetchDirection()), [getFetchSize](http://docs.google.com/java/sql/ResultSet.html#getFetchSize()), [getFloat](http://docs.google.com/java/sql/ResultSet.html#getFloat(int)), [getFloat](http://docs.google.com/java/sql/ResultSet.html#getFloat(java.lang.String)), [getHoldability](http://docs.google.com/java/sql/ResultSet.html#getHoldability()), [getInt](http://docs.google.com/java/sql/ResultSet.html#getInt(int)), [getInt](http://docs.google.com/java/sql/ResultSet.html#getInt(java.lang.String)), [getLong](http://docs.google.com/java/sql/ResultSet.html#getLong(int)), [getLong](http://docs.google.com/java/sql/ResultSet.html#getLong(java.lang.String)), [getMetaData](http://docs.google.com/java/sql/ResultSet.html#getMetaData()), [getNCharacterStream](http://docs.google.com/java/sql/ResultSet.html#getNCharacterStream(int)), [getNCharacterStream](http://docs.google.com/java/sql/ResultSet.html#getNCharacterStream(java.lang.String)), [getNClob](http://docs.google.com/java/sql/ResultSet.html#getNClob(int)), [getNClob](http://docs.google.com/java/sql/ResultSet.html#getNClob(java.lang.String)), [getNString](http://docs.google.com/java/sql/ResultSet.html#getNString(int)), [getNString](http://docs.google.com/java/sql/ResultSet.html#getNString(java.lang.String)), [getObject](http://docs.google.com/java/sql/ResultSet.html#getObject(int)), [getObject](http://docs.google.com/java/sql/ResultSet.html#getObject(int,%20java.util.Map)), [getObject](http://docs.google.com/java/sql/ResultSet.html#getObject(java.lang.String)), [getObject](http://docs.google.com/java/sql/ResultSet.html#getObject(java.lang.String,%20java.util.Map)), [getRef](http://docs.google.com/java/sql/ResultSet.html#getRef(int)), [getRef](http://docs.google.com/java/sql/ResultSet.html#getRef(java.lang.String)), [getRow](http://docs.google.com/java/sql/ResultSet.html#getRow()), [getRowId](http://docs.google.com/java/sql/ResultSet.html#getRowId(int)), [getRowId](http://docs.google.com/java/sql/ResultSet.html#getRowId(java.lang.String)), [getShort](http://docs.google.com/java/sql/ResultSet.html#getShort(int)), [getShort](http://docs.google.com/java/sql/ResultSet.html#getShort(java.lang.String)), [getSQLXML](http://docs.google.com/java/sql/ResultSet.html#getSQLXML(int)), [getSQLXML](http://docs.google.com/java/sql/ResultSet.html#getSQLXML(java.lang.String)), [getStatement](http://docs.google.com/java/sql/ResultSet.html#getStatement()), [getString](http://docs.google.com/java/sql/ResultSet.html#getString(int)), [getString](http://docs.google.com/java/sql/ResultSet.html#getString(java.lang.String)), [getTime](http://docs.google.com/java/sql/ResultSet.html#getTime(int)), [getTime](http://docs.google.com/java/sql/ResultSet.html#getTime(int,%20java.util.Calendar)), [getTime](http://docs.google.com/java/sql/ResultSet.html#getTime(java.lang.String)), [getTime](http://docs.google.com/java/sql/ResultSet.html#getTime(java.lang.String,%20java.util.Calendar)), [getTimestamp](http://docs.google.com/java/sql/ResultSet.html#getTimestamp(int)), [getTimestamp](http://docs.google.com/java/sql/ResultSet.html#getTimestamp(int,%20java.util.Calendar)), [getTimestamp](http://docs.google.com/java/sql/ResultSet.html#getTimestamp(java.lang.String)), [getTimestamp](http://docs.google.com/java/sql/ResultSet.html#getTimestamp(java.lang.String,%20java.util.Calendar)), [getType](http://docs.google.com/java/sql/ResultSet.html#getType()), [getUnicodeStream](http://docs.google.com/java/sql/ResultSet.html#getUnicodeStream(int)), [getUnicodeStream](http://docs.google.com/java/sql/ResultSet.html#getUnicodeStream(java.lang.String)), [getURL](http://docs.google.com/java/sql/ResultSet.html#getURL(int)), [getURL](http://docs.google.com/java/sql/ResultSet.html#getURL(java.lang.String)), [getWarnings](http://docs.google.com/java/sql/ResultSet.html#getWarnings()), [insertRow](http://docs.google.com/java/sql/ResultSet.html#insertRow()), [isAfterLast](http://docs.google.com/java/sql/ResultSet.html#isAfterLast()), [isBeforeFirst](http://docs.google.com/java/sql/ResultSet.html#isBeforeFirst()), [isClosed](http://docs.google.com/java/sql/ResultSet.html#isClosed()), [isFirst](http://docs.google.com/java/sql/ResultSet.html#isFirst()), [isLast](http://docs.google.com/java/sql/ResultSet.html#isLast()), [last](http://docs.google.com/java/sql/ResultSet.html#last()), [moveToCurrentRow](http://docs.google.com/java/sql/ResultSet.html#moveToCurrentRow()), [moveToInsertRow](http://docs.google.com/java/sql/ResultSet.html#moveToInsertRow()), [next](http://docs.google.com/java/sql/ResultSet.html#next()), [previous](http://docs.google.com/java/sql/ResultSet.html#previous()), [refreshRow](http://docs.google.com/java/sql/ResultSet.html#refreshRow()), [relative](http://docs.google.com/java/sql/ResultSet.html#relative(int)), [rowDeleted](http://docs.google.com/java/sql/ResultSet.html#rowDeleted()), [rowInserted](http://docs.google.com/java/sql/ResultSet.html#rowInserted()), [rowUpdated](http://docs.google.com/java/sql/ResultSet.html#rowUpdated()), [setFetchDirection](http://docs.google.com/java/sql/ResultSet.html#setFetchDirection(int)), [setFetchSize](http://docs.google.com/java/sql/ResultSet.html#setFetchSize(int)), [updateArray](http://docs.google.com/java/sql/ResultSet.html#updateArray(int,%20java.sql.Array)), [updateArray](http://docs.google.com/java/sql/ResultSet.html#updateArray(java.lang.String,%20java.sql.Array)), [updateAsciiStream](http://docs.google.com/java/sql/ResultSet.html#updateAsciiStream(int,%20java.io.InputStream)), [updateAsciiStream](http://docs.google.com/java/sql/ResultSet.html#updateAsciiStream(int,%20java.io.InputStream,%20int)), [updateAsciiStream](http://docs.google.com/java/sql/ResultSet.html#updateAsciiStream(int,%20java.io.InputStream,%20long)), [updateAsciiStream](http://docs.google.com/java/sql/ResultSet.html#updateAsciiStream(java.lang.String,%20java.io.InputStream)), [updateAsciiStream](http://docs.google.com/java/sql/ResultSet.html#updateAsciiStream(java.lang.String,%20java.io.InputStream,%20int)), [updateAsciiStream](http://docs.google.com/java/sql/ResultSet.html#updateAsciiStream(java.lang.String,%20java.io.InputStream,%20long)), [updateBigDecimal](http://docs.google.com/java/sql/ResultSet.html#updateBigDecimal(int,%20java.math.BigDecimal)), [updateBigDecimal](http://docs.google.com/java/sql/ResultSet.html#updateBigDecimal(java.lang.String,%20java.math.BigDecimal)), [updateBinaryStream](http://docs.google.com/java/sql/ResultSet.html#updateBinaryStream(int,%20java.io.InputStream)), [updateBinaryStream](http://docs.google.com/java/sql/ResultSet.html#updateBinaryStream(int,%20java.io.InputStream,%20int)), [updateBinaryStream](http://docs.google.com/java/sql/ResultSet.html#updateBinaryStream(int,%20java.io.InputStream,%20long)), [updateBinaryStream](http://docs.google.com/java/sql/ResultSet.html#updateBinaryStream(java.lang.String,%20java.io.InputStream)), [updateBinaryStream](http://docs.google.com/java/sql/ResultSet.html#updateBinaryStream(java.lang.String,%20java.io.InputStream,%20int)), [updateBinaryStream](http://docs.google.com/java/sql/ResultSet.html#updateBinaryStream(java.lang.String,%20java.io.InputStream,%20long)), [updateBlob](http://docs.google.com/java/sql/ResultSet.html#updateBlob(int,%20java.sql.Blob)), [updateBlob](http://docs.google.com/java/sql/ResultSet.html#updateBlob(int,%20java.io.InputStream)), [updateBlob](http://docs.google.com/java/sql/ResultSet.html#updateBlob(int,%20java.io.InputStream,%20long)), [updateBlob](http://docs.google.com/java/sql/ResultSet.html#updateBlob(java.lang.String,%20java.sql.Blob)), [updateBlob](http://docs.google.com/java/sql/ResultSet.html#updateBlob(java.lang.String,%20java.io.InputStream)), [updateBlob](http://docs.google.com/java/sql/ResultSet.html#updateBlob(java.lang.String,%20java.io.InputStream,%20long)), [updateBoolean](http://docs.google.com/java/sql/ResultSet.html#updateBoolean(int,%20boolean)), [updateBoolean](http://docs.google.com/java/sql/ResultSet.html#updateBoolean(java.lang.String,%20boolean)), [updateByte](http://docs.google.com/java/sql/ResultSet.html#updateByte(int,%20byte)), [updateByte](http://docs.google.com/java/sql/ResultSet.html#updateByte(java.lang.String,%20byte)), [updateBytes](http://docs.google.com/java/sql/ResultSet.html#updateBytes(int,%20byte%5B%5D)), [updateBytes](http://docs.google.com/java/sql/ResultSet.html#updateBytes(java.lang.String,%20byte%5B%5D)), [updateCharacterStream](http://docs.google.com/java/sql/ResultSet.html#updateCharacterStream(int,%20java.io.Reader)), [updateCharacterStream](http://docs.google.com/java/sql/ResultSet.html#updateCharacterStream(int,%20java.io.Reader,%20int)), [updateCharacterStream](http://docs.google.com/java/sql/ResultSet.html#updateCharacterStream(int,%20java.io.Reader,%20long)), [updateCharacterStream](http://docs.google.com/java/sql/ResultSet.html#updateCharacterStream(java.lang.String,%20java.io.Reader)), [updateCharacterStream](http://docs.google.com/java/sql/ResultSet.html#updateCharacterStream(java.lang.String,%20java.io.Reader,%20int)), [updateCharacterStream](http://docs.google.com/java/sql/ResultSet.html#updateCharacterStream(java.lang.String,%20java.io.Reader,%20long)), [updateClob](http://docs.google.com/java/sql/ResultSet.html#updateClob(int,%20java.sql.Clob)), [updateClob](http://docs.google.com/java/sql/ResultSet.html#updateClob(int,%20java.io.Reader)), [updateClob](http://docs.google.com/java/sql/ResultSet.html#updateClob(int,%20java.io.Reader,%20long)), [updateClob](http://docs.google.com/java/sql/ResultSet.html#updateClob(java.lang.String,%20java.sql.Clob)), [updateClob](http://docs.google.com/java/sql/ResultSet.html#updateClob(java.lang.String,%20java.io.Reader)), [updateClob](http://docs.google.com/java/sql/ResultSet.html#updateClob(java.lang.String,%20java.io.Reader,%20long)), [updateDate](http://docs.google.com/java/sql/ResultSet.html#updateDate(int,%20java.sql.Date)), [updateDate](http://docs.google.com/java/sql/ResultSet.html#updateDate(java.lang.String,%20java.sql.Date)), [updateDouble](http://docs.google.com/java/sql/ResultSet.html#updateDouble(int,%20double)), [updateDouble](http://docs.google.com/java/sql/ResultSet.html#updateDouble(java.lang.String,%20double)), [updateFloat](http://docs.google.com/java/sql/ResultSet.html#updateFloat(int,%20float)), [updateFloat](http://docs.google.com/java/sql/ResultSet.html#updateFloat(java.lang.String,%20float)), [updateInt](http://docs.google.com/java/sql/ResultSet.html#updateInt(int,%20int)), [updateInt](http://docs.google.com/java/sql/ResultSet.html#updateInt(java.lang.String,%20int)), [updateLong](http://docs.google.com/java/sql/ResultSet.html#updateLong(int,%20long)), [updateLong](http://docs.google.com/java/sql/ResultSet.html#updateLong(java.lang.String,%20long)), [updateNCharacterStream](http://docs.google.com/java/sql/ResultSet.html#updateNCharacterStream(int,%20java.io.Reader)), [updateNCharacterStream](http://docs.google.com/java/sql/ResultSet.html#updateNCharacterStream(int,%20java.io.Reader,%20long)), [updateNCharacterStream](http://docs.google.com/java/sql/ResultSet.html#updateNCharacterStream(java.lang.String,%20java.io.Reader)), [updateNCharacterStream](http://docs.google.com/java/sql/ResultSet.html#updateNCharacterStream(java.lang.String,%20java.io.Reader,%20long)), [updateNClob](http://docs.google.com/java/sql/ResultSet.html#updateNClob(int,%20java.sql.NClob)), [updateNClob](http://docs.google.com/java/sql/ResultSet.html#updateNClob(int,%20java.io.Reader)), [updateNClob](http://docs.google.com/java/sql/ResultSet.html#updateNClob(int,%20java.io.Reader,%20long)), [updateNClob](http://docs.google.com/java/sql/ResultSet.html#updateNClob(java.lang.String,%20java.sql.NClob)), [updateNClob](http://docs.google.com/java/sql/ResultSet.html#updateNClob(java.lang.String,%20java.io.Reader)), [updateNClob](http://docs.google.com/java/sql/ResultSet.html#updateNClob(java.lang.String,%20java.io.Reader,%20long)), [updateNString](http://docs.google.com/java/sql/ResultSet.html#updateNString(int,%20java.lang.String)), [updateNString](http://docs.google.com/java/sql/ResultSet.html#updateNString(java.lang.String,%20java.lang.String)), [updateNull](http://docs.google.com/java/sql/ResultSet.html#updateNull(int)), [updateNull](http://docs.google.com/java/sql/ResultSet.html#updateNull(java.lang.String)), [updateObject](http://docs.google.com/java/sql/ResultSet.html#updateObject(int,%20java.lang.Object)), [updateObject](http://docs.google.com/java/sql/ResultSet.html#updateObject(int,%20java.lang.Object,%20int)), [updateObject](http://docs.google.com/java/sql/ResultSet.html#updateObject(java.lang.String,%20java.lang.Object)), [updateObject](http://docs.google.com/java/sql/ResultSet.html#updateObject(java.lang.String,%20java.lang.Object,%20int)), [updateRef](http://docs.google.com/java/sql/ResultSet.html#updateRef(int,%20java.sql.Ref)), [updateRef](http://docs.google.com/java/sql/ResultSet.html#updateRef(java.lang.String,%20java.sql.Ref)), [updateRow](http://docs.google.com/java/sql/ResultSet.html#updateRow()), [updateRowId](http://docs.google.com/java/sql/ResultSet.html#updateRowId(int,%20java.sql.RowId)), [updateRowId](http://docs.google.com/java/sql/ResultSet.html#updateRowId(java.lang.String,%20java.sql.RowId)), [updateShort](http://docs.google.com/java/sql/ResultSet.html#updateShort(int,%20short)), [updateShort](http://docs.google.com/java/sql/ResultSet.html#updateShort(java.lang.String,%20short)), [updateSQLXML](http://docs.google.com/java/sql/ResultSet.html#updateSQLXML(int,%20java.sql.SQLXML)), [updateSQLXML](http://docs.google.com/java/sql/ResultSet.html#updateSQLXML(java.lang.String,%20java.sql.SQLXML)), [updateString](http://docs.google.com/java/sql/ResultSet.html#updateString(int,%20java.lang.String)), [updateString](http://docs.google.com/java/sql/ResultSet.html#updateString(java.lang.String,%20java.lang.String)), [updateTime](http://docs.google.com/java/sql/ResultSet.html#updateTime(int,%20java.sql.Time)), [updateTime](http://docs.google.com/java/sql/ResultSet.html#updateTime(java.lang.String,%20java.sql.Time)), [updateTimestamp](http://docs.google.com/java/sql/ResultSet.html#updateTimestamp(int,%20java.sql.Timestamp)), [updateTimestamp](http://docs.google.com/java/sql/ResultSet.html#updateTimestamp(java.lang.String,%20java.sql.Timestamp)), [wasNull](http://docs.google.com/java/sql/ResultSet.html#wasNull()) |

| **Methods inherited from interface java.sql.**[**Wrapper**](http://docs.google.com/java/sql/Wrapper.html) |
| --- |
| [isWrapperFor](http://docs.google.com/java/sql/Wrapper.html#isWrapperFor(java.lang.Class)), [unwrap](http://docs.google.com/java/sql/Wrapper.html#unwrap(java.lang.Class)) |

| **Methods inherited from interface javax.sql.rowset.**[**Joinable**](http://docs.google.com/javax/sql/rowset/Joinable.html) |
| --- |
| [getMatchColumnIndexes](http://docs.google.com/javax/sql/rowset/Joinable.html#getMatchColumnIndexes()), [getMatchColumnNames](http://docs.google.com/javax/sql/rowset/Joinable.html#getMatchColumnNames()), [setMatchColumn](http://docs.google.com/javax/sql/rowset/Joinable.html#setMatchColumn(int)), [setMatchColumn](http://docs.google.com/javax/sql/rowset/Joinable.html#setMatchColumn(int%5B%5D)), [setMatchColumn](http://docs.google.com/javax/sql/rowset/Joinable.html#setMatchColumn(java.lang.String)), [setMatchColumn](http://docs.google.com/javax/sql/rowset/Joinable.html#setMatchColumn(java.lang.String%5B%5D)), [unsetMatchColumn](http://docs.google.com/javax/sql/rowset/Joinable.html#unsetMatchColumn(int)), [unsetMatchColumn](http://docs.google.com/javax/sql/rowset/Joinable.html#unsetMatchColumn(int%5B%5D)), [unsetMatchColumn](http://docs.google.com/javax/sql/rowset/Joinable.html#unsetMatchColumn(java.lang.String)), [unsetMatchColumn](http://docs.google.com/javax/sql/rowset/Joinable.html#unsetMatchColumn(java.lang.String%5B%5D)) |

| **Field Detail** |
| --- |

COMMIT\_ON\_ACCEPT\_CHANGESstatic final boolean **COMMIT\_ON\_ACCEPT\_CHANGES**Causes the CachedRowSet object's SyncProvider to commit the changes when acceptChanges() is called. If set to false, the changes will **not** be committed until one of the CachedRowSet interface transaction methods is called.**See Also:**[commit()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#commit()), [rollback()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#rollback()), [Constant Field Values](http://docs.google.com/constant-values.html#javax.sql.rowset.CachedRowSet.COMMIT_ON_ACCEPT_CHANGES)

| **Method Detail** |
| --- |

populatevoid **populate**([ResultSet](http://docs.google.com/java/sql/ResultSet.html) data)  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Populates this CachedRowSet object with data from the given ResultSet object.This method can be used as an alternative to the execute method when an application has a connection to an open ResultSet object. Using the method populate can be more efficient than using the version of the execute method that takes no parameters because it does not open a new connection and re-execute this CachedRowSet object's command. Using the populate method is more a matter of convenience when compared to using the version of execute that takes a ResultSet object.**Parameters:**data - the ResultSet object containing the data to be read into this CachedRowSet object **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if a null ResultSet object is supplied or this CachedRowSet object cannot retrieve the associated ResultSetMetaData object**See Also:**[execute(java.sql.Connection)](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#execute(java.sql.Connection)), [ResultSet](http://docs.google.com/java/sql/ResultSet.html), [ResultSetMetaData](http://docs.google.com/java/sql/ResultSetMetaData.html)executevoid **execute**([Connection](http://docs.google.com/java/sql/Connection.html) conn)  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Populates this CachedRowSet object with data, using the given connection to produce the result set from which the data will be read. This method should close any database connections that it creates to ensure that this CachedRowSet object is disconnected except when it is reading data from its data source or writing data to its data source.The reader for this CachedRowSet object will use *conn* to establish a connection to the data source so that it can execute the rowset's command and read data from the the resulting ResultSet object into this CachedRowSet object. This method also closes *conn* after it has populated this CachedRowSet object.If this method is called when an implementation has already been populated, the contents and the metadata are (re)set. Also, if this method is called before the method acceptChanges has been called to commit outstanding updates, those updates are lost.**Parameters:**conn - a standard JDBC Connection object with valid properties **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if an invalid Connection object is supplied or an error occurs in establishing the connection to the data source**See Also:**[populate(java.sql.ResultSet)](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#populate(java.sql.ResultSet)), [Connection](http://docs.google.com/java/sql/Connection.html)acceptChangesvoid **acceptChanges**()  
 throws [SyncProviderException](http://docs.google.com/javax/sql/rowset/spi/SyncProviderException.html)Propagates row update, insert and delete changes made to this CachedRowSet object to the underlying data source.This method calls on this CachedRowSet object's writer to do the work behind the scenes. Standard CachedRowSet implementations should use the SyncFactory singleton to obtain a SyncProvider instance providing a RowSetWriter object (writer). The writer will attempt to propagate changes made in this CachedRowSet object back to the data source.When the method acceptChanges executes successfully, in addition to writing changes to the data source, it makes the values in the current row be the values in the original row.Depending on the synchronization level of the SyncProvider implementation being used, the writer will compare the original values with those in the data source to check for conflicts. When there is a conflict, the RIOptimisticProvider implementation, for example, throws a SyncProviderException and does not write anything to the data source.An application may choose to catch the SyncProviderException object and retrieve the SyncResolver object it contains. The SyncResolver object lists the conflicts row by row and sets a lock on the data source to avoid further conflicts while the current conflicts are being resolved. Further, for each conflict, it provides methods for examining the conflict and setting the value that should be persisted in the data source. After all conflicts have been resolved, an application must call the acceptChanges method again to write resolved values to the data source. If all of the values in the data source are already the values to be persisted, the method acceptChanges does nothing.Some provider implementations may use locks to ensure that there are no conflicts. In such cases, it is guaranteed that the writer will succeed in writing changes to the data source when the method acceptChanges is called. This method may be called immediately after the methods updateRow, insertRow, or deleteRow have been called, but it is more efficient to call it only once after all changes have been made so that only one connection needs to be established.Note: The acceptChanges() method will determine if the COMMIT\_ON\_ACCEPT\_CHANGES is set to true or not. If it is set to true, all updates in the synchronization are committed to the data source. Otherwise, the application **must** explicity call the commit() or rollback() methods as appropriate.**Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if the cursor is on the insert row [SyncProviderException](http://docs.google.com/javax/sql/rowset/spi/SyncProviderException.html) - if the underlying synchronization provider's writer fails to write the updates back to the data source**See Also:**[acceptChanges(java.sql.Connection)](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#acceptChanges(java.sql.Connection)), [RowSetWriter](http://docs.google.com/javax/sql/RowSetWriter.html), [SyncFactory](http://docs.google.com/javax/sql/rowset/spi/SyncFactory.html), [SyncProvider](http://docs.google.com/javax/sql/rowset/spi/SyncProvider.html), [SyncProviderException](http://docs.google.com/javax/sql/rowset/spi/SyncProviderException.html), [SyncResolver](http://docs.google.com/javax/sql/rowset/spi/SyncResolver.html)acceptChangesvoid **acceptChanges**([Connection](http://docs.google.com/java/sql/Connection.html) con)  
 throws [SyncProviderException](http://docs.google.com/javax/sql/rowset/spi/SyncProviderException.html)Propagates all row update, insert and delete changes to the data source backing this CachedRowSet object using the specified Connection object to establish a connection to the data source.The other version of the acceptChanges method is not passed a connection because it uses the Connection object already defined within the RowSet object, which is the connection used for populating it initially.This form of the method acceptChanges is similar to the form that takes no arguments; however, unlike the other form, this form can be used only when the underlying data source is a JDBC data source. The updated Connection properties must be used by the SyncProvider to reset the RowSetWriter configuration to ensure that the contents of the CachedRowSet object are synchronized correctly.When the method acceptChanges executes successfully, in addition to writing changes to the data source, it makes the values in the current row be the values in the original row.Depending on the synchronization level of the SyncProvider implementation being used, the writer will compare the original values with those in the data source to check for conflicts. When there is a conflict, the RIOptimisticProvider implementation, for example, throws a SyncProviderException and does not write anything to the data source.An application may choose to catch the SyncProviderException object and retrieve the SyncResolver object it contains. The SyncResolver object lists the conflicts row by row and sets a lock on the data source to avoid further conflicts while the current conflicts are being resolved. Further, for each conflict, it provides methods for examining the conflict and setting the value that should be persisted in the data source. After all conflicts have been resolved, an application must call the acceptChanges method again to write resolved values to the data source. If all of the values in the data source are already the values to be persisted, the method acceptChanges does nothing.Some provider implementations may use locks to ensure that there are no conflicts. In such cases, it is guaranteed that the writer will succeed in writing changes to the data source when the method acceptChanges is called. This method may be called immediately after the methods updateRow, insertRow, or deleteRow have been called, but it is more efficient to call it only once after all changes have been made so that only one connection needs to be established.Note: The acceptChanges() method will determine if the COMMIT\_ON\_ACCEPT\_CHANGES is set to true or not. If it is set to true, all updates in the synchronization are committed to the data source. Otherwise, the application **must** explicity call the commit or rollback methods as appropriate.**Parameters:**con - a standard JDBC Connection object **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if the cursor is on the insert row [SyncProviderException](http://docs.google.com/javax/sql/rowset/spi/SyncProviderException.html) - if the underlying synchronization provider's writer fails to write the updates back to the data source**See Also:**[acceptChanges()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#acceptChanges()), [RowSetWriter](http://docs.google.com/javax/sql/RowSetWriter.html), [SyncFactory](http://docs.google.com/javax/sql/rowset/spi/SyncFactory.html), [SyncProvider](http://docs.google.com/javax/sql/rowset/spi/SyncProvider.html), [SyncProviderException](http://docs.google.com/javax/sql/rowset/spi/SyncProviderException.html), [SyncResolver](http://docs.google.com/javax/sql/rowset/spi/SyncResolver.html)restoreOriginalvoid **restoreOriginal**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Restores this CachedRowSet object to its original value, that is, its value before the last set of changes. If there have been no changes to the rowset or only one set of changes, the original value is the value with which this CachedRowSet object was populated; otherwise, the original value is the value it had immediately before its current value.When this method is called, a CachedRowSet implementation must ensure that all updates, inserts, and deletes to the current rowset instance are replaced by the previous values. In addition, the cursor should be reset to the first row and a rowSetChanged event should be fired to notify all registered listeners.**Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if an error occurs rolling back the current value of this CachedRowSet object to its previous value**See Also:**[RowSetListener.rowSetChanged(javax.sql.RowSetEvent)](http://docs.google.com/javax/sql/RowSetListener.html#rowSetChanged(javax.sql.RowSetEvent))releasevoid **release**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Releases the current contents of this CachedRowSet object and sends a rowSetChanged event to all registered listeners. Any outstanding updates are discarded and the rowset contains no rows after this method is called. There are no interactions with the underlying data source, and any rowset content, metadata, and content updates should be non-recoverable.This CachedRowSet object should lock until its contents and associated updates are fully cleared, thus preventing 'dirty' reads by other components that hold a reference to this RowSet object. In addition, the contents cannot be released until all all components reading this CachedRowSet object have completed their reads. This CachedRowSet object should be returned to normal behavior after firing the rowSetChanged event.The metadata, including JDBC properties and Synchronization SPI properties, are maintained for future use. It is important that properties such as the command property be relevant to the originating data source from which this CachedRowSet object was originally established.This method empties a rowset, as opposed to the close method, which marks the entire rowset as recoverable to allow the garbage collector the rowset's Java VM resources.**Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if an error occurs flushing the contents of this CachedRowSet object**See Also:**[RowSetListener.rowSetChanged(javax.sql.RowSetEvent)](http://docs.google.com/javax/sql/RowSetListener.html#rowSetChanged(javax.sql.RowSetEvent)), [ResultSet.close()](http://docs.google.com/java/sql/ResultSet.html#close())undoDeletevoid **undoDelete**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Cancels the deletion of the current row and notifies listeners that a row has changed. After this method is called, the current row is no longer marked for deletion. This method can be called at any time during the lifetime of the rowset.In addition, multiple cancellations of row deletions can be made by adjusting the position of the cursor using any of the cursor position control methods such as:

### CachedRowSet.absolute

* + CachedRowSet.first
  + CachedRowSet.last

**Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if (1) the current row has not been deleted or (2) the cursor is on the insert row, before the first row, or after the last row**See Also:**[undoInsert()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#undoInsert()), [ResultSet.cancelRowUpdates()](http://docs.google.com/java/sql/ResultSet.html#cancelRowUpdates())undoInsertvoid **undoInsert**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Immediately removes the current row from this CachedRowSet object if the row has been inserted, and also notifies listeners that a row has changed. This method can be called at any time during the lifetime of a rowset and assuming the current row is within the exception limitations (see below), it cancels the row insertion of the current row.In addition, multiple cancellations of row insertions can be made by adjusting the position of the cursor using any of the cursor position control methods such as:

### CachedRowSet.absolute

* + CachedRowSet.first
  + CachedRowSet.last

**Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if (1) the current row has not been inserted or (2) the cursor is before the first row, after the last row, or on the insert row**See Also:**[undoDelete()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#undoDelete()), [ResultSet.cancelRowUpdates()](http://docs.google.com/java/sql/ResultSet.html#cancelRowUpdates())undoUpdatevoid **undoUpdate**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Immediately reverses the last update operation if the row has been modified. This method can be called to reverse updates on all columns until all updates in a row have been rolled back to their state just prior to the last synchronization (acceptChanges) or population. This method may also be called while performing updates to the insert row.undoUpdate may be called at any time during the lifetime of a rowset; however, after a synchronization has occurred, this method has no effect until further modification to the rowset data has occurred.**Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if the cursor is before the first row or after the last row in in this CachedRowSet object**See Also:**[undoDelete()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#undoDelete()), [undoInsert()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#undoInsert()), [ResultSet.cancelRowUpdates()](http://docs.google.com/java/sql/ResultSet.html#cancelRowUpdates())columnUpdatedboolean **columnUpdated**(int idx)  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Indicates whether the designated column in the current row of this CachedRowSet object has been updated.**Parameters:**idx - an int identifying the column to be checked for updates **Returns:**true if the designated column has been visibly updated; false otherwise **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if the cursor is on the insert row, before the first row, or after the last row**See Also:**[DatabaseMetaData.updatesAreDetected(int)](http://docs.google.com/java/sql/DatabaseMetaData.html#updatesAreDetected(int))columnUpdatedboolean **columnUpdated**([String](http://docs.google.com/java/lang/String.html) columnName)  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Indicates whether the designated column in the current row of this CachedRowSet object has been updated.**Parameters:**columnName - a String object giving the name of the column to be checked for updates **Returns:**true if the column has been visibly updated; false otherwise **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if the cursor is on the insert row, before the first row, or after the last row**See Also:**[DatabaseMetaData.updatesAreDetected(int)](http://docs.google.com/java/sql/DatabaseMetaData.html#updatesAreDetected(int))toCollection[Collection](http://docs.google.com/java/util/Collection.html)<?> **toCollection**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Converts this CachedRowSet object to a Collection object that contains all of this CachedRowSet object's data. Implementations have some latitude in how they can represent this Collection object because of the abstract nature of the Collection framework. Each row must be fully represented in either a general purpose Collection implementation or a specialized Collection implementation, such as a TreeMap object or a Vector object. An SQL NULL column value must be represented as a null in the Java programming language.The standard reference implementation for the CachedRowSet interface uses a TreeMap object for the rowset, with the values in each row being contained in Vector objects. It is expected that most implementations will do the same.The TreeMap type of collection guarantees that the map will be in ascending key order, sorted according to the natural order for the key's class. Each key references a Vector object that corresponds to one row of a RowSet object. Therefore, the size of each Vector object must be exactly equal to the number of columns in the RowSet object. The key used by the TreeMap collection is determined by the implementation, which may choose to leverage a set key that is available within the internal RowSet tabular structure by virtue of a key already set either on the RowSet object itself or on the underlying SQL data.**Returns:**a Collection object that contains the values in each row in this CachedRowSet object **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if an error occurs generating the collection**See Also:**[toCollection(int)](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#toCollection(int)), [toCollection(String)](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#toCollection(java.lang.String))toCollection[Collection](http://docs.google.com/java/util/Collection.html)<?> **toCollection**(int column)  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Converts the designated column in this CachedRowSet object to a Collection object. Implementations have some latitude in how they can represent this Collection object because of the abstract nature of the Collection framework. Each column value should be fully represented in either a general purpose Collection implementation or a specialized Collection implementation, such as a Vector object. An SQL NULL column value must be represented as a null in the Java programming language.The standard reference implementation uses a Vector object to contain the column values, and it is expected that most implementations will do the same. If a Vector object is used, it size must be exactly equal to the number of rows in this CachedRowSet object.**Parameters:**column - an int indicating the column whose values are to be represented in a Collection object **Returns:**a Collection object that contains the values stored in the specified column of this CachedRowSet object **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if an error occurs generating the collection or an invalid column id is provided**See Also:**[toCollection()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#toCollection()), [toCollection(String)](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#toCollection(java.lang.String))toCollection[Collection](http://docs.google.com/java/util/Collection.html)<?> **toCollection**([String](http://docs.google.com/java/lang/String.html) column)  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Converts the designated column in this CachedRowSet object to a Collection object. Implementations have some latitude in how they can represent this Collection object because of the abstract nature of the Collection framework. Each column value should be fully represented in either a general purpose Collection implementation or a specialized Collection implementation, such as a Vector object. An SQL NULL column value must be represented as a null in the Java programming language.The standard reference implementation uses a Vector object to contain the column values, and it is expected that most implementations will do the same. If a Vector object is used, it size must be exactly equal to the number of rows in this CachedRowSet object.**Parameters:**column - a String object giving the name of the column whose values are to be represented in a collection **Returns:**a Collection object that contains the values stored in the specified column of this CachedRowSet object **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if an error occurs generating the collection or an invalid column id is provided**See Also:**[toCollection()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#toCollection()), [toCollection(int)](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#toCollection(int))getSyncProvider[SyncProvider](http://docs.google.com/javax/sql/rowset/spi/SyncProvider.html) **getSyncProvider**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Retrieves the SyncProvider implementation for this CachedRowSet object. Internally, this method is used by a rowset to trigger read or write actions between the rowset and the data source. For example, a rowset may need to get a handle on the the rowset reader (RowSetReader object) from the SyncProvider to allow the rowset to be populated. RowSetReader rowsetReader = null;  
 SyncProvider provider =   
 SyncFactory.getInstance("javax.sql.rowset.provider.RIOptimisticProvider");  
 if (provider instanceof RIOptimisticProvider) {  
 rowsetReader = provider.getRowSetReader();  
 }  
 Assuming *rowsetReader* is a private, accessible field within the rowset implementation, when an application calls the execute method, it in turn calls on the reader's readData method to populate the RowSet object.   
 rowsetReader.readData((RowSetInternal)this);  
 In addition, an application can use the SyncProvider object returned by this method to call methods that return information about the SyncProvider object, including information about the vendor, version, provider identification, synchronization grade, and locks it currently has set.**Returns:**the SyncProvider object that was set when the rowset was instantiated, or if none was was set, the default provider **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if an error occurs while returning the SyncProvider object**See Also:**[setSyncProvider(java.lang.String)](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#setSyncProvider(java.lang.String))setSyncProvidervoid **setSyncProvider**([String](http://docs.google.com/java/lang/String.html) provider)  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Sets the SyncProvider objec for this CachedRowSet object to the one specified. This method allows the SyncProvider object to be reset.A CachedRowSet implementation should always be instantiated with an available SyncProvider mechanism, but there are cases where resetting the SyncProvider object is desirable or necessary. For example, an application might want to use the default SyncProvider object for a time and then choose to use a provider that has more recently become available and better fits its needs.Resetting the SyncProvider object causes the RowSet object to request a new SyncProvider implementation from the SyncFactory. This has the effect of resetting all previous connections and relationships with the originating data source and can potentially drastically change the synchronization behavior of a disconnected rowset.**Parameters:**provider - a String object giving the fully qualified class name of a SyncProvider implementation **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if an error occurs while attempting to reset the SyncProvider implementation**See Also:**[getSyncProvider()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#getSyncProvider())sizeint **size**()Returns the number of rows in this CachedRowSet object.**Returns:**number of rows in the rowsetsetMetaDatavoid **setMetaData**([RowSetMetaData](http://docs.google.com/javax/sql/RowSetMetaData.html) md)  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Sets the metadata for this CachedRowSet object with the given RowSetMetaData object. When a RowSetReader object is reading the contents of a rowset, it creates a RowSetMetaData object and initializes it using the methods in the RowSetMetaData implementation. The reference implementation uses the RowSetMetaDataImpl class. When the reader has completed reading the rowset contents, this method is called internally to pass the RowSetMetaData object to the rowset.**Parameters:**md - a RowSetMetaData object containing metadata about the columns in this CachedRowSet object **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if invalid metadata is supplied to the rowsetgetOriginal[ResultSet](http://docs.google.com/java/sql/ResultSet.html) **getOriginal**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Returns a ResultSet object containing the original value of this CachedRowSet object.The cursor for the ResultSet object should be positioned before the first row. In addition, the returned ResultSet object should have the following properties:

### ResultSet.TYPE\_SCROLL\_INSENSITIVE

* + ResultSet.CONCUR\_UPDATABLE

The original value for a RowSet object is the value it had before the last synchronization with the underlying data source. If there have been no synchronizations, the original value will be the value with which the RowSet object was populated. This method is called internally when an aplication calls the method acceptChanges and the SyncProvider object has been implemented to check for conflicts. If this is the case, the writer compares the original value with the value currently in the data source to check for conflicts.**Returns:**a ResultSet object that contains the original value for this CachedRowSet object **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if an error occurs producing the ResultSet objectgetOriginalRow[ResultSet](http://docs.google.com/java/sql/ResultSet.html) **getOriginalRow**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Returns a ResultSet object containing the original value for the current row only of this CachedRowSet object.The cursor for the ResultSet object should be positioned before the first row. In addition, the returned ResultSet object should have the following properties:

### ResultSet.TYPE\_SCROLL\_INSENSITIVE

* + ResultSet.CONCUR\_UPDATABLE

**Returns:**the original result set of the row **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if there is no current row**See Also:**[setOriginalRow()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#setOriginalRow())setOriginalRowvoid **setOriginalRow**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Sets the current row in this CachedRowSet object as the original row.This method is called internally after the any modified values in the current row have been synchronized with the data source. The current row must be tagged as no longer inserted, deleted or updated.A call to setOriginalRow is irreversible.**Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if there is no current row or an error is encountered resetting the contents of the original row**See Also:**[getOriginalRow()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#getOriginalRow())getTableName[String](http://docs.google.com/java/lang/String.html) **getTableName**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Returns an identifier for the object (table) that was used to create this CachedRowSet object. This name may be set on multiple occasions, and the specification imposes no limits on how many times this may occur or whether standard implementations should keep track of previous table names.**Returns:**a String object giving the name of the table that is the source of data for this CachedRowSet object or null if no name has been set for the table **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if an error is encountered returning the table name**See Also:**[ResultSetMetaData.getTableName(int)](http://docs.google.com/java/sql/ResultSetMetaData.html#getTableName(int))setTableNamevoid **setTableName**([String](http://docs.google.com/java/lang/String.html) tabName)  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Sets the identifier for the table from which this CachedRowSet object was derived to the given table name. The writer uses this name to determine which table to use when comparing the values in the data source with the CachedRowSet object's values during a synchronization attempt. The table identifier also indicates where modified values from this CachedRowSet object should be written.The implementation of this CachedRowSet object may obtain the the name internally from the RowSetMetaDataImpl object.**Parameters:**tabName - a String object identifying the table from which this CachedRowSet object was derived; cannot be null but may be an empty string **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if an error is encountered naming the table or *tabName* is null**See Also:**[RowSetMetaData.setTableName(int, java.lang.String)](http://docs.google.com/javax/sql/RowSetMetaData.html#setTableName(int,%20java.lang.String)), [RowSetWriter](http://docs.google.com/javax/sql/RowSetWriter.html), [SyncProvider](http://docs.google.com/javax/sql/rowset/spi/SyncProvider.html)getKeyColumnsint[] **getKeyColumns**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Returns an array containing one or more column numbers indicating the columns that form a key that uniquely identifies a row in this CachedRowSet object.**Returns:**an array containing the column number or numbers that indicate which columns constitute a primary key for a row in this CachedRowSet object. This array should be empty if no columns are representative of a primary key. **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if this CachedRowSet object is empty**See Also:**[setKeyColumns(int[])](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#setKeyColumns(int%5B%5D)), [Joinable.getMatchColumnIndexes()](http://docs.google.com/javax/sql/rowset/Joinable.html#getMatchColumnIndexes()), [Joinable.getMatchColumnNames()](http://docs.google.com/javax/sql/rowset/Joinable.html#getMatchColumnNames())setKeyColumnsvoid **setKeyColumns**(int[] keys)  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Sets this CachedRowSet object's keyCols field with the given array of column numbers, which forms a key for uniquely identifying a row in this CachedRowSet object.If a CachedRowSet object becomes part of a JoinRowSet object, the keys defined by this method and the resulting constraints are maintained if the columns designated as key columns also become match columns.**Parameters:**keys - an array of int indicating the columns that form a primary key for this CachedRowSet object; every element in the array must be greater than 0 and less than or equal to the number of columns in this rowset **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if any of the numbers in the given array are not valid for this rowset**See Also:**[getKeyColumns()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#getKeyColumns()), [Joinable.setMatchColumn(String)](http://docs.google.com/javax/sql/rowset/Joinable.html#setMatchColumn(java.lang.String)), [Joinable.setMatchColumn(int)](http://docs.google.com/javax/sql/rowset/Joinable.html#setMatchColumn(int))createShared[RowSet](http://docs.google.com/javax/sql/RowSet.html) **createShared**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Returns a new RowSet object backed by the same data as that of this CachedRowSet object. In effect, both CachedRowSet objects have a cursor over the same data. As a result, any changes made by a duplicate are visible to the original and to any other duplicates, just as a change made by the original is visible to all of its duplicates. If a duplicate calls a method that changes the underlying data, the method it calls notifies all registered listeners just as it would when it is called by the original CachedRowSet object.In addition, any RowSet object created by this method will have the same properties as this CachedRowSet object. For example, if this CachedRowSet object is read-only, all of its duplicates will also be read-only. If it is changed to be updatable, the duplicates also become updatable.NOTE: If multiple threads access RowSet objects created from the createShared() method, the following behavior is specified to preserve shared data integrity: reads and writes of all shared RowSet objects should be made serially between each object and the single underlying tabular structure.**Returns:**a new shared RowSet object that has the same properties as this CachedRowSet object and that has a cursor over the same data **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if an error occurs or cloning is not supported in the underlying platform**See Also:**[RowSetEvent](http://docs.google.com/javax/sql/RowSetEvent.html), [RowSetListener](http://docs.google.com/javax/sql/RowSetListener.html)createCopy[CachedRowSet](http://docs.google.com/javax/sql/rowset/CachedRowSet.html) **createCopy**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Creates a RowSet object that is a deep copy of the data in this CachedRowSet object. In contrast to the RowSet object generated from a createShared call, updates made to the copy of the original RowSet object must not be visible to the original RowSet object. Also, any event listeners that are registered with the original RowSet must not have scope over the new RowSet copies. In addition, any constraint restrictions established must be maintained.**Returns:**a new RowSet object that is a deep copy of this CachedRowSet object and is completely independent of this CachedRowSet object **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if an error occurs in generating the copy of the of this CachedRowSet object**See Also:**[createShared()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#createShared()), [createCopySchema()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#createCopySchema()), [createCopyNoConstraints()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#createCopyNoConstraints()), [RowSetEvent](http://docs.google.com/javax/sql/RowSetEvent.html), [RowSetListener](http://docs.google.com/javax/sql/RowSetListener.html)createCopySchema[CachedRowSet](http://docs.google.com/javax/sql/rowset/CachedRowSet.html) **createCopySchema**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Creates a CachedRowSet object that is an empty copy of this CachedRowSet object. The copy must not contain any contents but only represent the table structure of the original CachedRowSet object. In addition, primary or foreign key constraints set in the originating CachedRowSet object must be equally enforced in the new empty CachedRowSet object. In contrast to the RowSet object generated from a createShared method call, updates made to a copy of this CachedRowSet object with the createCopySchema method must not be visible to it.Applications can form a WebRowSet object from the CachedRowSet object returned by this method in order to export the RowSet schema definition to XML for future use.**Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if an error occurs in cloning the structure of this CachedRowSet object**See Also:**[createShared()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#createShared()), [createCopySchema()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#createCopySchema()), [createCopyNoConstraints()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#createCopyNoConstraints()), [RowSetEvent](http://docs.google.com/javax/sql/RowSetEvent.html), [RowSetListener](http://docs.google.com/javax/sql/RowSetListener.html)createCopyNoConstraints[CachedRowSet](http://docs.google.com/javax/sql/rowset/CachedRowSet.html) **createCopyNoConstraints**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Creates a CachedRowSet object that is a deep copy of this CachedRowSet object's data but is independent of it. In contrast to the RowSet object generated from a createShared method call, updates made to a copy of this CachedRowSet object must not be visible to it. Also, any event listeners that are registered with this CachedRowSet object must not have scope over the new RowSet object. In addition, any constraint restrictions established for this CachedRowSet object must **not** be maintained in the copy.**Returns:**a new CachedRowSet object that is a deep copy of this CachedRowSet object and is completely independent of this CachedRowSet object **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if an error occurs in generating the copy of the of this CachedRowSet object**See Also:**[createCopy()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#createCopy()), [createShared()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#createShared()), [createCopySchema()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#createCopySchema()), [RowSetEvent](http://docs.google.com/javax/sql/RowSetEvent.html), [RowSetListener](http://docs.google.com/javax/sql/RowSetListener.html)getRowSetWarnings[RowSetWarning](http://docs.google.com/javax/sql/rowset/RowSetWarning.html) **getRowSetWarnings**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Retrieves the first warning reported by calls on this RowSet object. Subsequent warnings on this RowSet object will be chained to the RowSetWarning object that this method returns. The warning chain is automatically cleared each time a new row is read. This method may not be called on a RowSet object that has been closed; doing so will cause a SQLException to be thrown.**Returns:**RowSetWarning the first RowSetWarning object reported or null if there are none **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if this method is called on a closed RowSet**See Also:**[RowSetWarning](http://docs.google.com/javax/sql/rowset/RowSetWarning.html)getShowDeletedboolean **getShowDeleted**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Retrieves a boolean indicating whether rows marked for deletion appear in the set of current rows. If true is returned, deleted rows are visible with the current rows. If false is returned, rows are not visible with the set of current rows. The default value is false.Standard rowset implementations may choose to restrict this behavior due to security considerations or to better fit certain deployment scenarios. This is left as implementation defined and does not represent standard behavior.Note: Allowing deleted rows to remain visible complicates the behavior of some standard JDBC RowSet Implementations methods. However, most rowset users can simply ignore this extra detail because only very specialized applications will likely want to take advantage of this feature.**Returns:**true if deleted rows are visible; false otherwise **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if a rowset implementation is unable to to determine whether rows marked for deletion are visible**See Also:**[setShowDeleted(boolean)](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#setShowDeleted(boolean))setShowDeletedvoid **setShowDeleted**(boolean b)  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Sets the property showDeleted to the given boolean value, which determines whether rows marked for deletion appear in the set of current rows. If the value is set to true, deleted rows are immediately visible with the set of current rows. If the value is set to false, the deleted rows are set as invisible with the current set of rows.Standard rowset implementations may choose to restrict this behavior due to security considerations or to better fit certain deployment scenarios. This is left as implementations defined and does not represent standard behavior.**Parameters:**b - true if deleted rows should be shown; false otherwise **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if a rowset implementation is unable to to reset whether deleted rows should be visible**See Also:**[getShowDeleted()](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#getShowDeleted())commitvoid **commit**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Each CachedRowSet object's SyncProvider contains a Connection object from the ResultSet or JDBC properties passed to it's constructors. This method wraps the Connection commit method to allow flexible auto commit or non auto commit transactional control support.Makes all changes that are performed by the acceptChanges() method since the previous commit/rollback permanent. This method should be used only when auto-commit mode has been disabled.**Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if a database access error occurs or this Connection object within this CachedRowSet is in auto-commit mode**See Also:**[Connection.setAutoCommit(boolean)](http://docs.google.com/java/sql/Connection.html#setAutoCommit(boolean))rollbackvoid **rollback**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Each CachedRowSet object's SyncProvider contains a Connection object from the original ResultSet or JDBC properties passed to it.Undoes all changes made in the current transaction. This method should be used only when auto-commit mode has been disabled.**Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if a database access error occurs or this Connection object within this CachedRowSet is in auto-commit mode.rollbackvoid **rollback**([Savepoint](http://docs.google.com/java/sql/Savepoint.html) s)  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Each CachedRowSet object's SyncProvider contains a Connection object from the original ResultSet or JDBC properties passed to it.Undoes all changes made in the current transaction back to the last Savepoint transaction marker. This method should be used only when auto-commit mode has been disabled.**Parameters:**s - A Savepoint transaction marker **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if a database access error occurs or this Connection object within this CachedRowSet is in auto-commit mode.rowSetPopulatedvoid **rowSetPopulated**([RowSetEvent](http://docs.google.com/javax/sql/RowSetEvent.html) event,  
 int numRows)  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Notifies registered listeners that a RowSet object in the given RowSetEvent object has populated a number of additional rows. The numRows parameter ensures that this event will only be fired every numRow.The source of the event can be retrieved with the method event.getSource.**Parameters:**event - a RowSetEvent object that contains the RowSet object that is the source of the eventsnumRows - when populating, the number of rows interval on which the CachedRowSet populated should fire; the default value is zero; cannot be less than fetchSize or zero **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html)populatevoid **populate**([ResultSet](http://docs.google.com/java/sql/ResultSet.html) rs,  
 int startRow)  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Populates this CachedRowSet object with data from the given ResultSet object. While related to the populate(ResultSet) method, an additional parameter is provided to allow starting position within the ResultSet from where to populate the CachedRowSet instance.This method can be used as an alternative to the execute method when an application has a connection to an open ResultSet object. Using the method populate can be more efficient than using the version of the execute method that takes no parameters because it does not open a new connection and re-execute this CachedRowSet object's command. Using the populate method is more a matter of convenience when compared to using the version of execute that takes a ResultSet object.**Parameters:**startRow - the position in the ResultSet from where to start populating the records in this CachedRowSetrs - the ResultSet object containing the data to be read into this CachedRowSet object **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if a null ResultSet object is supplied or this CachedRowSet object cannot retrieve the associated ResultSetMetaData object**See Also:**[execute(java.sql.Connection)](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#execute(java.sql.Connection)), [populate(ResultSet)](http://docs.google.com/javax/sql/rowset/CachedRowSet.html#populate(java.sql.ResultSet)), [ResultSet](http://docs.google.com/java/sql/ResultSet.html), [ResultSetMetaData](http://docs.google.com/java/sql/ResultSetMetaData.html)setPageSizevoid **setPageSize**(int size)  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Sets the CachedRowSet object's page-size. A CachedRowSet may be configured to populate itself in page-size sized batches of rows. When either populate() or execute() are called, the CachedRowSet fetches an additional page according to the original SQL query used to populate the RowSet.**Parameters:**size - the page-size of the CachedRowSet **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if an error occurs setting the CachedRowSet page size or if the page size is less than 0.getPageSizeint **getPageSize**()Returns the page-size for the CachedRowSet object**Returns:**an int page sizenextPageboolean **nextPage**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Increments the current page of the CachedRowSet. This causes the CachedRowSet implementation to fetch the next page-size rows and populate the RowSet, if remaining rows remain within scope of the original SQL query used to populated the RowSet.**Returns:**true if more pages exist; false if this is the last page **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if an error occurs fetching the next page, or if this method is called prematurely before populate or execute.previousPageboolean **previousPage**()  
 throws [SQLException](http://docs.google.com/java/sql/SQLException.html)Decrements the current page of the CachedRowSet. This causes the CachedRowSet implementation to fetch the previous page-size rows and populate the RowSet. The amount of rows returned in the previous page must always remain within scope of the original SQL query used to populate the RowSet.**Returns:**true if the previous page is successfully retrieved; false if this is the first page. **Throws:** [SQLException](http://docs.google.com/java/sql/SQLException.html) - if an error occurs fetching the previous page, or if this method is called prematurely before populate or execute.

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/CachedRowSet.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform*** ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/javax/sql/rowset/BaseRowSet.html)   [**NEXT CLASS**](http://docs.google.com/javax/sql/rowset/FilteredRowSet.html) | [**FRAMES**](http://docs.google.com/index.html?javax/sql/rowset/CachedRowSet.html)    [**NO FRAMES**](http://docs.google.com/CachedRowSet.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | CONSTR | [METHOD](#tyjcwt) | DETAIL: [FIELD](#17dp8vu) | CONSTR | [METHOD](#26in1rg) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)  
For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).