java.sql

Interface DataSet<T>

All Superinterfaces:
Collection<T>, Iterable<T>, List<T>

```
public interface DataSet<T>
extends List<T>
```

A `DataSet` provides a type safe view of the data returned from the execution of a SQL Query. It is a subinterface of `java.util.List`. A `DataSet` is also a parameterized type. The parameter type is a *data class* describing the columns for the rows returned by invoking a method on a `Query` interface decorated by a `Select` annotation. The *data class* must have an access modifier of `public`.

A `DataSet` may operate in a connected or disconnected mode. When used in a connected mode, the `DataSet` functions in a manner similar to ResultSet. A disconnected `DataSet` functions in a manner similar to a CachedRowSet.

A `DataSet` objects allows for the iteration through the rows that were returned using the Iterator API.

Navigating a DataSet

A `DataSet` is a subinterface of the `List` allowing for an application to navigate through a `DataSet` object:

```
public class Mammal {
   public String name;
   public String description;
   public int age;
}

interface MyQueries extends BaseQuery {
   @Select("select name, description, age from mammal")
   DataSet<Mammal> getAllMammals();
}

MyQueries mq = con.createQueryObject(MyQueries.class);
DataSet rows = mq.getAllMammals();

for (Mammal m: rows) {
    System.out.println("Name = " + m.name);
    System.out.println("Description =  " + m.description);
}
```

Modifying a Row within a DataSet

You may update rows within a `DataSet` instance by positioning to the row to be modified, making the required modifications and then calling the `modify` method on the `DataSet`. If the `DataSet` is disconnected, the `tableName` annotation element must be specified in the `Select` annotation.

```
DataSet rows = mq.getAllMammals();

for (Mammal m: rows) {
    if (m.description.equals("")) {
        m.description="a mammal";
        rows.modify();
    }
}
```

Deleting a Row within a DataSet

You may delete rows within a `DataSet` instance by positioning to the row to be deleted, and then calling the `delete` method on the `DataSet`. If the `DataSet` is disconnected, the `tableName` annotation element must be specified in the `Select` annotation.

```
DataSet rows = mq.getAllMammals();

for (Mammal m: rows) {
    if (m.description.equals("tbd")) {
        rows.delete();
    }
}
```

Inserting a Row into a DataSet

You may insert rows into an existing `DataSet` instance. If the `DataSet` is disconnected, the `tableName` annotation element must be specified in the `Select` annotation. To insert a row, create an instance of the *data class* for the given `DataSet` and populate it. The method `DataSet.insert` is called to add the row to the `DataSet`.

```
DataSet rows = mq.getAllMammals();
Mammal newMammal = new Mammal();
newMammal.name="Jane Doe";
newMammal.description = "a real dear";
rows.insert(newMammal);
```

Syncronizing a DataSet

A `DataSet` can be configured to operate in a disconnected manner by setting the `Select` annotation element `connected` to `false`. This requires any changes made to the `DataSet` to be explicitly syncronized to the underlying data store. The `DataSet.sync` method must be called to propagate the `DatasSet` modifications to the data store. The modifications are done as an atomic operation.

If an invocation of `DataSet.sync` fails, a `SQLDataSetSyncException` will be thrown. The `SQLDataSetSyncException.getDataSetResolver` method can be called to navigate the rows that could not be updated in the data store along with the cause of the failure.

Since:
1.6

Method Summary

`javax.sql.RowSet`	`asRowSet()` Retrieves a copy of this `DataSet` as a `RowSet`.
`void`	`clearWarnings()` Clears all warnings reported on this `DataSet` object.
`void`	`close()`

	Releases this `DataSet` object's database and JDBC resources immediately instead of waiting for this to happen when it is automatically closed.
`boolean`	`delete()` Delete the currently positioned row within a `DataSet`
`T`	`getRow()` Return an object representing the currently positioned row within the `DataSet`.
`SQLWarning`	`getWarnings()` Retrieves the first warning reported by invoking methods on this `DataSet` object.
`boolean`	`insert(T row)` Inserts the specified row into this `DataSet`.
`boolean`	`isClosed()` Retrieves whether this `DataSet` object has been closed.
`boolean`	`isConnected()` Indicates whether this `DataSet` is connected to the underlying data store .
`boolean`	`isReadOnly()` Indicates whether this `DataSet` is read-only or not.
`boolean`	`isScrollable()` Indicates whether a `DataSet` is scrollable.
`boolean`	`modify()` Modify the currently positioned row within a `DataSet`.
`boolean`	`modify(T row)` Modifies the currently positioned row with the specified row in this `DataSet`.
`void`	`sync()` Propagates modifications made to a disconnected `DataSet` to the underlying data store.
`void`	`sync(Connection con)` Propagates modifications made to a disconnected `DataSet` to the underlying data store using the specified `Connection` object.

Methods inherited from interface java.util.List
`add`, `add`, `addAll`, `addAll`, `clear`, `contains`, `containsAll`, `equals`, `get`, `hashCode`, `indexOf`, `isEmpty`, `iterator`, `lastIndexOf`, `listIterator`, `listIterator`, `remove`, `remove`, `removeAll`, `retainAll`, `set`, `size`, `subList`, `toArray`, `toArray`

Method Detail

asRowSet

```
javax.sql.RowSet asRowSet()
```

Retrieves a copy of this `DataSet` as a `RowSet`.

Returns:

A copy the `DataSet` as a `RowSet`

Throws:

`SQLRuntimeException` - if an error occurs while creating the RowSet

Since:

1.6

See Also:

`RowSet`, `javax.sql.rowset.JdbcRowSet`, `javax.sql.rowset.CachedRowSet`, `javax.sql.rowset.JoinRowSet`, `javax.sql.rowset.FilteredRowSet`

---

getRow

`T getRow()`

Return an object representing the currently positioned row within the `DataSet`.

Returns:

a row of type T

Throws:

`SQLRuntimeException` - if an error occurs retrieving the current row

Since:

1.6

---

insert

`boolean insert(T row)`

Inserts the specified row into this `DataSet`. If the `DataSet` is connected and is not marked as read-only, the insert is applied directly to the underlying data store. If the `DataSet` is marked as disconnected, the `DataSet.sync` method must be called in order to apply the changes to the data store.

Parameters:

`row` - A row of type T

Returns:

`true` if the insert of the row to this `DataSet` is successful; `false` otherwise

Throws:

`SQLRuntimeException` - if an error occurs inserting the a new row; or if this `DataSet` is read-only

Since:

1.6

See Also:

`List.add(E)`

---

modify

`boolean modify(T row)`

Modifies the currently positioned row with the specified row in this `DataSet`. If the `DataSet` is connected and is not marked as read-only, the update is applied directly to the underlying data store. If the `DataSet` is marked as disconnected, the `DataSet.sync` method must be called in order to apply the changes to the data store.

Parameters:

`row` - the row of type T that that replaces the current row

Returns:

`true` if the `DataSet` is modified successfully; `false` otherwise

Throws:

`SQLRuntimeException` - if an error occurs modifying the row; or if this `DataSet` is read-only

Since:

1.6

---

modify

```
boolean modify()
```

Modify the currently positioned row within a `DataSet`.

Returns:

true if this `DataSet` is modified succesfully; `false` otherwise

Throws:

`SQLRuntimeException` - if an error occurs modifying the current row; an attempt to invoke `modify` without positioning to a row; or the `DataSet` is read-only

Since:

1.6

---

delete

```
boolean delete()
```

Delete the currently positioned row within a `DataSet`

Returns:

true if this `DataSet` is modified successfully; `false` otherwise

Throws:

`SQLRuntimeException` - if an error occurs deleting the current row; an attempt to invoke `delete` without positioning to a row; or the `DataSet` is read-only

Since:

1.6

---

close

```
void close()
```

Releases this `DataSet` object's database and JDBC resources immediately instead of waiting for this to happen when it is automatically closed.

Throws:

`SQLRuntimeException` - if an error occurs closing this `DataSet`

Since:

1.6

---

isReadOnly

```
boolean isReadOnly()
```

Indicates whether this `DataSet` is read-only or not. Developers may configure a `DataSet` instance as read-only by setting the `readOnly` annotation element to `true` for the `Query` annotation whose method returned this `DataSet`.

A value of `true` must be returned if the `readOnly` annotation element is set to `true`. Some JDBC drivers may also return a value of `true` if the query is characterized to be read-only which can occur when a query involves the use of DISTINCT, UNION, GROUP BY or is a JOIN.

Returns:
true if this `DataSet` is read-only; `false` otherwise

Throws:
`SQLRuntimeException` - if an error occurs accessing the `DataSet`

Since:
1.6

---

sync

`void sync()`

Propagates modifications made to a disconnected `DataSet` to the underlying data store. This method works in the same manner as the `CachedRowSet` method `acceptChanges`.

Note: The `sync()` can only be used by a `DataSet` that is disconnected and whose Query Object instance was created by calling `DataSource.createQueryObject`.

Throws:
`SQLDataSetSyncException` - if an error occurs writing the changes back to the data source; the `DataSet` is connected to the underlying data store; or the Query Object instance from which this `DataSet` was derived from, was not created from a `DataSource`

Since:
1.6

See Also:
`javax.sql.rowset.CachedRowSet#acceptChanges`, `DataSource.createQueryObject(java.lang.Class)`

---

sync

`void sync(Connection con)`

Propagates modifications made to a disconnected `DataSet` to the underlying data store using the specified `Connection` object. This method works in the same manner as the `CachedRowSet` method `acceptChanges(Connection)`.

Note: The `sync()` can only be used by a `DataSet` that is disconnected.

Parameters:
`con` - The connection object to use to synchronize the changes back to the underlying data store.

Throws:
`SQLDataSetSyncException` - if an error occurs writing the changes back to the data source; or the `DataSet` is connected to the underlying data store

Since:
1.6

See Also:
`javax.sql.rowset.CachedRowSet#acceptChanges(Connection)`

---

isConnected

`boolean isConnected()`

Indicates whether this `DataSet` is connected to the underlying data store . Developers may configure a `DataSet` instance as connected by setting the `connected` annotation element to `true` for the `Query` annotation whose method returned this `DataSet`.

Returns:
true if this `DataSet` is connected to the underly data source; `false` otherwise

Throws:
`SQLRuntimeException` - if an error occurs accessing the `DataSet`

Since:
1.6

---

isScrollable

`boolean isScrollable()`

Indicates whether a `DataSet` is scrollable. If the `DataSet` is connected and is scrollable, it has a `ResultSet` type of `TYPE_SCROLL_INSENSITVE`. If the `DataSet` is not scrollable, the `ResultSet` type is specified to be `TYPE_FORWARD_ONLY`. A disconnected `DataSet` is considered scrollable as it is implemented as a `CachedRowSet`.

Returns:
`true` if the connected `DataSet` is scrollable; `false` otherwise.

Throws:
`SQLRuntimeException` - if an error occurs accessing the `DataSet`

Since:
1.6

---

isClosed

`boolean isClosed()`

Retrieves whether this `DataSet` object has been closed. A `DataSet` is closed if the method close has been called on it, or if it is automatically closed.

Returns:
true if this `DataSet` object is closed; false if it is still open

Throws:
`SQLRuntimeException` - if a database access error occurs

Since:
1.6

---

getWarnings

`SQLWarning getWarnings()`

Retrieves the first warning reported by invoking methods on this `DataSet` object. Subsequent `DataSet` object warnings will be chained to this `SQLWarning` object.

The warning chain is automatically cleared each time a new row is read. This method may not be called on a closed `DataSet` object; doing so will cause an `SQLRuntimeException` to be thrown.

Returns:
the first `SQLWarning` object or `null` if there are no warnings

Throws:
`SQLRuntimeException` - if a database access error occurs or this method is called on a closed `DataSet` Object

Since:
1.6

---

clearWarnings

```
void clearWarnings()
```

Clears all warnings reported on this `DataSet` object. After this method is called, the method `getWarnings` returns `null` until a new warning is reported for this `DataSet` object.

Throws:
`SQLRuntimeException` - if a database access error occurs

Since:
1.6
