The JDBC(tm) API Version 1.20

January 3, 1997

Part 1 - Interfaces

This document contains a paper copy of the JDBC API online documentation that is distributed with the JDBC package and is also available on http://splash.javasoft.com/jdbc.

It takes the place of the source code comments that were originally included as part of the JDBC specification.

---

package java.sql

Interface Index

Class Index

Exception Index

Interface java.sql.CallableStatement

public interface **CallableStatement**
extends Object

extends PreparedStatement

CallableStatement is used to execute SQL stored procedures.

JDBC provides a stored procedure SQL escape that allows stored procedures to be called in a standard way for all RDBMS's. This escape syntax has one form that includes a result parameter and one that does not. If used, the result parameter must be registered as an OUT parameter. The other parameters may be used for input, output or both. Parameters are refered to sequentially, by number. The first parameter is 1.

```
{?= call [,, ...]}
{call [,, ...]}
```

IN parameter values are set using the set methods inherited from PreparedStatement. The type of all OUT parameters must be registered prior to executing the stored procedure; their values are retrieved after execution via the get methods provided here.

A Callable statement may return a ResultSet or multiple ResultSets. Multiple ResultSets are handled using operations inherited from Statement.

For maximum portability, a call's ResultSets and update counts should be processed prior to getting the values of output parameters.

See Also:
prepareCall, ResultSet

Method Index

- **getBigDecimal**(int, int)
  Get the value of a NUMERIC parameter as a java.math.BigDecimal object.
- **getBoolean**(int)
  Get the value of a BIT parameter as a Java boolean.
- **getByte**(int)
  Get the value of a TINYINT parameter as a Java byte.
- **getBytes**(int)
  Get the value of a SQL BINARY or VARBINARY parameter as a Java byte[]
- **getDate**(int)
  Get the value of a SQL DATE parameter as a java.sql.Date object
- **getDouble**(int)
  Get the value of a DOUBLE parameter as a Java double.
- **getFloat**(int)
  Get the value of a FLOAT parameter as a Java float.
- **getInt**(int)
  Get the value of an INTEGER parameter as a Java int.
- **getLong**(int)

Get the value of a BIGINT parameter as a Java long.

- **getObject**(int)
  Get the value of a parameter as a Java object.
- **getShort**(int)
  Get the value of a SMALLINT parameter as a Java short.
- **getString**(int)
  Get the value of a CHAR, VARCHAR, or LONGVARCHAR parameter as a Java String.
- **getTime**(int)
  Get the value of a SQL TIME parameter as a java.sql.Time object.
- **getTimestamp**(int)
  Get the value of a SQL TIMESTAMP parameter as a java.sql.Timestamp object.
- **registerOutParameter**(int, int)
  Before executing a stored procedure call, you must explicitly call registerOutParameter to register the java.sql.Type of each out parameter.
- **registerOutParameter**(int, int, int)
  Use this version of registerOutParameter for registering Numeric or Decimal out parameters.
- **wasNull**()
  An OUT parameter may have the value of SQL NULL; wasNull reports whether the last value read has this special value.

registerOutParameter

```
public abstract void registerOutParameter(int parameterIndex,
                                          int sqlType) throws SQLException
```

Before executing a stored procedure call, you must explicitly call registerOutParameter to register the java.sql.Type of each out parameter.

Note: When reading the value of an out parameter, you must use the getXXX method whose Java type XXX corresponds to the parameter's registered SQL type.

Parameters:

parameterIndex - the first parameter is 1, the second is 2,...
sqlType - SQL type code defined by java.sql.Types; for parameters of type Numeric or Decimal use the version of registerOutParameter that accepts a scale value

See Also:

Type

registerOutParameter

```
public abstract void registerOutParameter(int parameterIndex,
                                          int sqlType,
                                          int scale) throws SQLException
```

Use this version of registerOutParameter for registering Numeric or Decimal out parameters.

Note: When reading the value of an out parameter, you must use the getXXX method whose Java type XXX corresponds to the parameter's registered SQL type.

Parameters:
parameterIndex - the first parameter is 1, the second is 2, ...
sqlType - use either java.sql.Type.NUMERIC or java.sql.Type.DECIMAL
scale - a value greater than or equal to zero representing the desired number of digits to the right of the decimal point

See Also:
Type

- **wasNull**

```
public abstract boolean wasNull() throws SQLException
```

An OUT parameter may have the value of SQL NULL; wasNull reports whether the last value read has this special value.

Note: You must first call getXXX on a parameter to read its value and then call wasNull() to see if the value was SQL NULL.

Returns:
true if the last parameter read was SQL NULL

- **getString**

```
public abstract String getString(int parameterIndex) throws SQLException
```

Get the value of a CHAR, VARCHAR, or LONGVARCHAR parameter as a Java String.

Parameters:
parameterIndex - the first parameter is 1, the second is 2, ...

Returns:
the parameter value; if the value is SQL NULL, the result is null

- **getBoolean**

```
public abstract boolean getBoolean(int parameterIndex) throws SQLException
```

Get the value of a BIT parameter as a Java boolean.

Parameters:
parameterIndex - the first parameter is 1, the second is 2, ...

Returns:
the parameter value; if the value is SQL NULL, the result is false

- **getByte**

```
public abstract byte getByte(int parameterIndex) throws SQLException
```

Get the value of a TINYINT parameter as a Java byte.

Parameters:
parameterIndex - the first parameter is 1, the second is 2, ...
Returns:
the parameter value; if the value is SQL NULL, the result is 0

- **getShort**

```
public abstract short getShort(int parameterIndex) throws SQLException
```

Get the value of a SMALLINT parameter as a Java short.
Parameters:
parameterIndex - the first parameter is 1, the second is 2, ...
Returns:
the parameter value; if the value is SQL NULL, the result is 0

- **getInt**

```
public abstract int getInt(int parameterIndex) throws SQLException
```

Get the value of an INTEGER parameter as a Java int.
Parameters:
parameterIndex - the first parameter is 1, the second is 2, ...
Returns:
the parameter value; if the value is SQL NULL, the result is 0

- **getLong**

```
public abstract long getLong(int parameterIndex) throws SQLException
```

Get the value of a BIGINT parameter as a Java long.
Parameters:
parameterIndex - the first parameter is 1, the second is 2, ...
Returns:
the parameter value; if the value is SQL NULL, the result is 0

- **getFloat**

```
public abstract float getFloat(int parameterIndex) throws SQLException
```

Get the value of a FLOAT parameter as a Java float.
Parameters:
parameterIndex - the first parameter is 1, the second is 2, ...
Returns:
the parameter value; if the value is SQL NULL, the result is 0

- **getDouble**

```
public abstract double getDouble(int parameterIndex) throws SQLException
```

Get the value of a DOUBLE parameter as a Java double.

Parameters:
parameterIndex - the first parameter is 1, the second is 2, ...
Returns:
the parameter value; if the value is SQL NULL, the result is 0

- **getBigDecimal**

```
public abstract BigDecimal getBigDecimal(int parameterIndex,
                                         int scale) throws SQLException
```

Get the value of a NUMERIC parameter as a java.math.BigDecimal object.
Parameters:
parameterIndex - the first parameter is 1, the second is 2, ...
scale - a value greater than or equal to zero representing the desired number of digits to the right of the decimal point
Returns:
the parameter value; if the value is SQL NULL, the result is null

- **getBytes**

```
public abstract byte[] getBytes(int parameterIndex) throws SQLException
```

Get the value of a SQL BINARY or VARBINARY parameter as a Java byte[]
Parameters:
parameterIndex - the first parameter is 1, the second is 2, ...
Returns:
the parameter value; if the value is SQL NULL, the result is null

- **getDate**

```
public abstract Date getDate(int parameterIndex) throws SQLException
```

Get the value of a SQL DATE parameter as a java.sql.Date object
Parameters:
parameterIndex - the first parameter is 1, the second is 2, ...
Returns:
the parameter value; if the value is SQL NULL, the result is null

- **getTime**

```
public abstract Time getTime(int parameterIndex) throws SQLException
```

Get the value of a SQL TIME parameter as a java.sql.Time object.
Parameters:
parameterIndex - the first parameter is 1, the second is 2, ...
Returns:
the parameter value; if the value is SQL NULL, the result is null

- **getTimestamp**

```
public abstract Timestamp getTimestamp(int parameterIndex) throws SQLException
```

Get the value of a SQL TIMESTAMP parameter as a java.sql.Timestamp object.

Parameters:
parameterIndex - the first parameter is 1, the second is 2, ...

Returns:
the parameter value; if the value is SQL NULL, the result is null

- **getObject**

```
public abstract Object getObject(int parameterIndex) throws SQLException
```

Get the value of a parameter as a Java object.

This method returns a Java object whose type coresponds to the SQL type that was registered for this parameter using registerOutParameter.

Note that this method may be used to read datatabase-specific, abstract data types. This is done by specifying a targetSqlType of java.sql.types.OTHER, which allows the driver to return a database-specific Java type.

Parameters:
parameterIndex - The first parameter is 1, the second is 2, ...

Returns:
A java.lang.Object holding the OUT parameter value.

See Also:
Types

Interface java.sql.Connection

public interface **Connection**
extends Object

A Connection represents a session with a specific database. Within the context of a Connection, SQL statements are executed and results are returned.

A Connection's database is able to provide information describing its tables, its supported SQL grammar, its stored procedures, the capabilities of this connection, etc. This information is obtained with the getMetaData method.

Note: By default the Connection automatically commits changes after executing each statement. If auto commit has been disabled, an explicit commit must be done or database changes will not be saved.

See Also:
getConnection, Statement, ResultSet, DatabaseMetaData

Variable Index

- **TRANSACTION_NONE**
  Transactions are not supported.
- **TRANSACTION_READ_COMMITTED**
  Dirty reads are prevented; non-repeatable reads and phantom reads can occur.
- **TRANSACTION_READ_UNCOMMITTED**
  Dirty reads, non-repeatable reads and phantom reads can occur.
- **TRANSACTION_REPEATABLE_READ**
  Dirty reads and non-repeatable reads are prevented; phantom reads can occur.
- **TRANSACTION_SERIALIZABLE**
  Dirty reads, non-repeatable reads and phantom reads are prevented.

Method Index

- **clearWarnings**()
  After this call, getWarnings returns null until a new warning is reported for this Connection.
- **close**()
  In some cases, it is desirable to immediately release a Connection's database and JDBC resources instead of waiting for them to be automatically released; the close method provides this immediate release.
- **commit**()
  Commit makes all changes made since the previous commit/rollback permanent and releases any database locks currently held by the Connection.
- **createStatement**()
  SQL statements without parameters are normally executed using Statement objects.
- **getAutoCommit**()
  Get the current auto-commit state.
- **getCatalog**()
  Return the Connection's current catalog name.
- **getMetaData**()
  A Connection's database is able to provide information describing its tables, its supported SQL grammar, its stored procedures, the capabilities of this connection, etc.
- **getTransactionIsolation**()
  Get this Connection's current transaction isolation mode.
- **getWarnings**()
  The first warning reported by calls on this Connection is returned.
- **isClosed**()
  Tests to see if a Connection is closed.
- **isReadOnly**()
  Tests to see if the connection is in read-only mode.
- **nativeSQL**(String)
  A driver may convert the JDBC sql grammar into its system's native SQL grammar prior to

sending it; nativeSQL returns the native form of the statement that the driver would have sent.

- **prepareCall**(String)
  A SQL stored procedure call statement is handled by creating a CallableStatement for it.
- **prepareStatement**(String)
  A SQL statement with or without IN parameters can be pre-compiled and stored in a PreparedStatement object.
- **rollback**()
  Rollback drops all changes made since the previous commit/rollback and releases any database locks currently held by the Connection.
- **setAutoCommit**(boolean)
  If a connection is in auto-commit mode, then all its SQL statements will be executed and committed as individual transactions.
- **setCatalog**(String)
  A sub-space of this Connection's database may be selected by setting a catalog name.
- **setReadOnly**(boolean)
  You can put a connection in read-only mode as a hint to enable database optimizations.
- **setTransactionIsolation**(int)
  You can call this method to try to change the transaction isolation level using one of the TRANSACTION_* values.

Variables

- **TRANSACTION_NONE**

```
public final static int TRANSACTION_NONE
```

  Transactions are not supported.

- **TRANSACTION_READ_UNCOMMITTED**

```
public final static int TRANSACTION_READ_UNCOMMITTED
```

  Dirty reads, non-repeatable reads and phantom reads can occur.

- **TRANSACTION_READ_COMMITTED**

```
public final static int TRANSACTION_READ_COMMITTED
```

  Dirty reads are prevented; non-repeatable reads and phantom reads can occur.

- **TRANSACTION_REPEATABLE_READ**

```
public final static int TRANSACTION_REPEATABLE_READ
```

  Dirty reads and non-repeatable reads are prevented; phantom reads can occur.

- **TRANSACTION_SERIALIZABLE**

```
public final static int TRANSACTION_SERIALIZABLE
```

Dirty reads, non-repeatable reads and phantom reads are prevented.

Methods

- **createStatement**

```
public abstract Statement createStatement() throws SQLException
```

SQL statements without parameters are normally executed using Statement objects. If the same SQL statement is executed many times, it is more efficient to use a PreparedStatement

Returns:

a new Statement object

- **prepareStatement**

```
public abstract PreparedStatement prepareStatement(String sql) throws SQLException
```

A SQL statement with or without IN parameters can be pre-compiled and stored in a PreparedStatement object. This object can then be used to efficiently execute this statement multiple times.

Note: This method is optimized for handling parametric SQL statements that benefit from precompilation. If the driver supports precompilation, prepareStatement will send the statement to the database for precompilation. Some drivers may not support precompilation. In this case, the statement may not be sent to the database until the PreparedStatement is executed. This has no direct affect on users; however, it does affect which method throws certain SQLExceptions.

Parameters:

sql - a SQL statement that may contain one or more '?' IN parameter placeholders

Returns:

a new PreparedStatement object containing the pre-compiled statement

- **prepareCall**

```
public abstract CallableStatement prepareCall(String sql) throws SQLException
```

A SQL stored procedure call statement is handled by creating a CallableStatement for it. The CallableStatement provides methods for setting up its IN and OUT parameters, and methods for executing it.

Note: This method is optimized for handling stored procedure call statements. Some drivers may send the call statement to the database when the prepareCall is done; others may wait until the CallableStatement is executed. This has no direct affect on users; however, it does affect which method throws certain SQLExceptions.

Parameters:

sql - a SQL statement that may contain one or more '?' parameter placeholders. Typically this statement is a JDBC function call escape string.

Returns:

a new CallableStatement object containing the pre-compiled SQL statement

- **nativeSQL**

```
public abstract String nativeSQL(String sql) throws SQLException
```

A driver may convert the JDBC sql grammar into its system's native SQL grammar prior to sending it; nativeSQL returns the native form of the statement that the driver would have sent.

Parameters:

sql - a SQL statement that may contain one or more '?' parameter placeholders

Returns:

the native form of this statement

- **setAutoCommit**

```
public abstract void setAutoCommit(boolean autoCommit) throws SQLException
```

If a connection is in auto-commit mode, then all its SQL statements will be executed and committed as individual transactions. Otherwise, its SQL statements are grouped into transactions that are terminated by either commit() or rollback(). By default, new connections are in auto-commit mode. The commit occurs when the statement completes or the next execute occurs, whichever comes first. In the case of statements returning a ResultSet, the statement completes when the last row of the ResultSet has been retrieved or the ResultSet has been closed. In advanced cases, a single statement may return multiple results as well as output parameter values. Here the commit occurs when all results and output param values have been retrieved.

Parameters:

autoCommit - true enables auto-commit; false disables auto-commit.

- **getAutoCommit**

```
public abstract boolean getAutoCommit() throws SQLException
```

Get the current auto-commit state.

Returns:

Current state of auto-commit mode.

See Also:

setAutoCommit

- **commit**

```
public abstract void commit() throws SQLException
```

Commit makes all changes made since the previous commit/rollback permanent and releases any database locks currently held by the Connection. This method should only be used when auto commit has been disabled.

See Also:

setAutoCommit

- **rollback**

```
public abstract void rollback() throws SQLException
```

Rollback drops all changes made since the previous commit/rollback and releases any database locks currently held by the Connection. This method should only be used when auto commit has been disabled.

See Also:

setAutoCommit

- **close**

```
public abstract void close() throws SQLException
```

In some cases, it is desirable to immediately release a Connection's database and JDBC resources instead of waiting for them to be automatically released; the close method provides this immediate release.

Note: A Connection is automatically closed when it is garbage collected. Certain fatal errors also result in a closed Connection.

- **isClosed**

```
public abstract boolean isClosed() throws SQLException
```

Tests to see if a Connection is closed.

Returns:

true if the connection is closed; false if it's still open

- **getMetaData**

```
public abstract DatabaseMetaData getMetaData() throws SQLException
```

A Connection's database is able to provide information describing its tables, its supported SQL grammar, its stored procedures, the capabilities of this connection, etc. This information is made available through a DatabaseMetaData object.

Returns:

a DatabaseMetaData object for this Connection

- **setReadOnly**

```
public abstract void setReadOnly(boolean readOnly) throws SQLException
```

You can put a connection in read-only mode as a hint to enable database optimizations.

Note: setReadOnly cannot be called while in the middle of a transaction.

Parameters:

readOnly - true enables read-only mode; false disables read-only mode.

- **isReadOnly**

```
public abstract boolean isReadOnly() throws SQLException
```

Tests to see if the connection is in read-only mode.

Returns:

true if connection is read-only

- **setCatalog**

```
public abstract void setCatalog(String catalog) throws SQLException
```

A sub-space of this Connection's database may be selected by setting a catalog name. If the driver does not support catalogs it will silently ignore this request.

- **getCatalog**

```
public abstract String getCatalog() throws SQLException
```

Return the Connection's current catalog name.

Returns:

the current catalog name or null

- **setTransactionIsolation**

```
public abstract void setTransactionIsolation(int level) throws SQLException
```

You can call this method to try to change the transaction isolation level using one of the TRANSACTION_* values.

Note: setTransactionIsolation cannot be called while in the middle of a transaction.

Parameters:

level - one of the TRANSACTION_* isolation values with the exception of TRANSACTION_NONE; some databases may not support other values

See Also:

supportsTransactionIsolationLevel

- **getTransactionIsolation**

```
public abstract int getTransactionIsolation() throws SQLException
```

Get this Connection's current transaction isolation mode.

Returns:

the current TRANSACTION_* mode value

- **getWarnings**

```
public abstract SQLWarning getWarnings() throws SQLException
```

The first warning reported by calls on this Connection is returned.

Note: Subsequent warnings will be chained to this SQLWarning.

Returns:
the first SQLWarning or null

- **clearWarnings**

```
public abstract void clearWarnings() throws SQLException
```

After this call, getWarnings returns null until a new warning is reported for this Connection.

Interface java.sql.DatabaseMetaData

public interface **DatabaseMetaData**
extends Object

This class provides information about the database as a whole.

Many of the methods here return lists of information in ResultSets. You can use the normal ResultSet methods such as getString and getInt to retrieve the data from these ResultSets. If a given form of metadata is not available, these methods should throw a SQLException.

Some of these methods take arguments that are String patterns. These arguments all have names such as fooPattern. Within a pattern String, "%" means match any substring of 0 or more characters, and "_" means match any one character. Only metadata entries matching the search pattern are returned. If a search pattern argument is set to a null ref, it means that argument's criteria should be dropped from the search.

A SQLException will be thrown if a driver does not support a meta data method. In the case of methods that return a ResultSet, either a ResultSet (which may be empty) is returned or a SQLException is thrown.

Variable Index

- **bestRowNotPseudo**
  BEST ROW PSEUDO_COLUMN - is NOT a pseudo column.
- **bestRowPseudo**
  BEST ROW PSEUDO_COLUMN - is a pseudo column.

- **bestRowSession**
  BEST ROW SCOPE - valid for remainder of current session.
- **bestRowTemporary**
  BEST ROW SCOPE - very temporary, while using row.
- **bestRowTransaction**
  BEST ROW SCOPE - valid for remainder of current transaction.
- **bestRowUnknown**
  BEST ROW PSEUDO_COLUMN - may or may not be pseudo column.
- **columnNoNulls**
  COLUMN NULLABLE - might not allow NULL values.
- **columnNullable**
  COLUMN NULLABLE - definitely allows NULL values.
- **columnNullableUnknown**
  COLUMN NULLABLE - nullability unknown.
- **importedKeyCascade**
  IMPORT KEY UPDATE_RULE and DELETE_RULE - for update, change imported key to agree with primary key update; for delete, delete rows that import a deleted key.
- **importedKeyInitiallyDeferred**
  IMPORT KEY DEFERRABILITY - see SQL92 for definition
- **importedKeyInitiallyImmediate**
  IMPORT KEY DEFERRABILITY - see SQL92 for definition
- **importedKeyNoAction**
  IMPORT KEY UPDATE_RULE and DELETE_RULE - do not allow update or delete of primary key if it has been imported.
- **importedKeyNotDeferrable**
  IMPORT KEY DEFERRABILITY - see SQL92 for definition
- **importedKeyRestrict**
  IMPORT KEY UPDATE_RULE and DELETE_RULE - do not allow update or delete of primary key if it has been imported.
- **importedKeySetDefault**
  IMPORT KEY UPDATE_RULE and DELETE_RULE - change imported key to default values if its primary key has been updated or deleted.
- **importedKeySetNull**
  IMPORT KEY UPDATE_RULE and DELETE_RULE - change imported key to NULL if its primary key has been updated or deleted.
- **procedureColumnIn**
  COLUMN_TYPE - IN parameter.
- **procedureColumnInOut**
  COLUMN_TYPE - INOUT parameter.
- **procedureColumnOut**
  COLUMN_TYPE - OUT parameter.
- **procedureColumnResult**
  COLUMN_TYPE - result column in ResultSet.
- **procedureColumnReturn**
  COLUMN_TYPE - procedure return value.
- **procedureColumnUnknown**
  COLUMN_TYPE - nobody knows.

- **procedureNoNulls**
  TYPE NULLABLE - does not allow NULL values.
- **procedureNoResult**
  PROCEDURE_TYPE - Does not return a result.
- **procedureNullable**
  TYPE NULLABLE - allows NULL values.
- **procedureNullableUnknown**
  TYPE NULLABLE - nullability unknown.
- **procedureResultUnknown**
  PROCEDURE_TYPE - May return a result.
- **procedureReturnsResult**
  PROCEDURE_TYPE - Returns a result.
- **tableIndexClustered**
  INDEX INFO TYPE - this identifies a clustered index
- **tableIndexHashed**
  INDEX INFO TYPE - this identifies a hashed index
- **tableIndexOther**
  INDEX INFO TYPE - this identifies some other form of index
- **tableIndexStatistic**
  INDEX INFO TYPE - this identifies table statistics that are returned in conjuction with a table's index descriptions
- **typeNoNulls**
  TYPE NULLABLE - does not allow NULL values.
- **typeNullable**
  TYPE NULLABLE - allows NULL values.
- **typeNullableUnknown**
  TYPE NULLABLE - nullability unknown.
- **typePredBasic**
  TYPE INFO SEARCHABLE - Supported except for WHERE ..
- **typePredChar**
  TYPE INFO SEARCHABLE - Only supported with WHERE ..
- **typePredNone**
  TYPE INFO SEARCHABLE - No support.
- **typeSearchable**
  TYPE INFO SEARCHABLE - Supported for all WHERE ...
- **versionColumnNotPseudo**
  VERSION COLUMNS PSEUDO_COLUMN - is NOT a pseudo column.
- **versionColumnPseudo**
  VERSION COLUMNS PSEUDO_COLUMN - is a pseudo column.
- **versionColumnUnknown**
  VERSION COLUMNS PSEUDO_COLUMN - may or may not be pseudo column.

Method Index

- **allProceduresAreCallable**()
  Can all the procedures returned by getProcedures be called by the current user?

- **allTablesAreSelectable**()
  Can all the tables returned by getTable be SELECTed by the current user?
- **dataDefinitionCausesTransactionCommit**()
  Does a data definition statement within a transaction force the transaction to commit?
- **dataDefinitionIgnoredInTransactions**()
  Is a data definition statement within a transaction ignored?
- **doesMaxRowSizeIncludeBlobs**()
  Did getMaxRowSize() include LONGVARCHAR and LONGVARBINARY blobs?
- **getBestRowIdentifier**(String, String, String, int, boolean)
  Get a description of a table's optimal set of columns that uniquely identifies a row.
- **getCatalogs**()
  Get the catalog names available in this database.
- **getCatalogSeparator**()
  What's the separator between catalog and table name?
- **getCatalogTerm**()
  What's the database vendor's preferred term for "catalog"?
- **getColumnPrivileges**(String, String, String, String)
  Get a description of the access rights for a table's columns.
- **getColumns**(String, String, String, String)
  Get a description of table columns available in a catalog.
- **getCrossReference**(String, String, String, String, String, String)
  Get a description of the foreign key columns in the foreign key table that reference the primary key columns of the primary key table (describe how one table imports another's key.) This should normally return a single foreign key/primary key pair (most tables only import a foreign key from a table once.) They are ordered by FKTABLE_CAT, FKTABLE_SCHEM, FKTABLE_NAME, and KEY_SEQ.
- **getDatabaseProductName**()
  What's the name of this database product?
- **getDatabaseProductVersion**()
  What's the version of this database product?
- **getDefaultTransactionIsolation**()
  What's the database's default transaction isolation level? The values are defined in java.sql.Connection.
- **getDriverMajorVersion**()
  What's this JDBC driver's major version number?
- **getDriverMinorVersion**()
  What's this JDBC driver's minor version number?
- **getDriverName**()
  What's the name of this JDBC driver?
- **getDriverVersion**()
  What's the version of this JDBC driver?
- **getExportedKeys**(String, String, String)
  Get a description of the foreign key columns that reference a table's primary key columns (the foreign keys exported by a table).
- **getExtraNameCharacters**()
  Get all the "extra" characters that can be used in unquoted identifier names (those beyond a-z, A-Z, 0-9 and _).

- **getIdentifierQuoteString**()
  What's the string used to quote SQL identifiers? This returns a space " " if identifier quoting isn't supported.
- **getImportedKeys**(String, String, String)
  Get a description of the primary key columns that are referenced by a table's foreign key columns (the primary keys imported by a table).
- **getIndexInfo**(String, String, String, boolean, boolean)
  Get a description of a table's indices and statistics.
- **getMaxBinaryLiteralLength**()
  How many hex characters can you have in an inline binary literal?
- **getMaxCatalogNameLength**()
  What's the maximum length of a catalog name?
- **getMaxCharLiteralLength**()
  What's the max length for a character literal?
- **getMaxColumnNameLength**()
  What's the limit on column name length?
- **getMaxColumnsInGroupBy**()
  What's the maximum number of columns in a "GROUP BY" clause?
- **getMaxColumnsInIndex**()
  What's the maximum number of columns allowed in an index?
- **getMaxColumnsInOrderBy**()
  What's the maximum number of columns in an "ORDER BY" clause?
- **getMaxColumnsInSelect**()
  What's the maximum number of columns in a "SELECT" list?
- **getMaxColumnsInTable**()
  What's the maximum number of columns in a table?
- **getMaxConnections**()
  How many active connections can we have at a time to this database?
- **getMaxCursorNameLength**()
  What's the maximum cursor name length?
- **getMaxIndexLength**()
  What's the maximum length of an index (in bytes)?
- **getMaxProcedureNameLength**()
  What's the maximum length of a procedure name?
- **getMaxRowSize**()
  What's the maximum length of a single row?
- **getMaxSchemaNameLength**()
  What's the maximum length allowed for a schema name?
- **getMaxStatementLength**()
  What's the maximum length of a SQL statement?
- **getMaxStatements**()
  How many active statements can we have open at one time to this database?
- **getMaxTableNameLength**()
  What's the maximum length of a table name?
- **getMaxTablesInSelect**()
  What's the maximum number of tables in a SELECT?
- **getMaxUserNameLength**()

What's the maximum length of a user name?

- **getNumericFunctions**()
  Get a comma separated list of math functions.
- **getPrimaryKeys**(String, String, String)
  Get a description of a table's primary key columns.
- **getProcedureColumns**(String, String, String, String)
  Get a description of a catalog's stored procedure parameters and result columns.
- **getProcedures**(String, String, String)
  Get a description of stored procedures available in a catalog.
- **getProcedureTerm**()
  What's the database vendor's preferred term for "procedure"?
- **getSchemas**()
  Get the schema names available in this database.
- **getSchemaTerm**()
  What's the database vendor's preferred term for "schema"?
- **getSearchStringEscape**()
  This is the string that can be used to escape '_' or '%' in the string pattern style catalog search parameters.
- **getSQLKeywords**()
  Get a comma separated list of all a database's SQL keywords that are NOT also SQL92 keywords.
- **getStringFunctions**()
  Get a comma separated list of string functions.
- **getSystemFunctions**()
  Get a comma separated list of system functions.
- **getTablePrivileges**(String, String, String)
  Get a description of the access rights for each table available in a catalog.
- **getTables**(String, String, String, String[])
  Get a description of tables available in a catalog.
- **getTableTypes**()
  Get the table types available in this database.
- **getTimeDateFunctions**()
  Get a comma separated list of time and date functions.
- **getTypeInfo**()
  Get a description of all the standard SQL types supported by this database.
- **getURL**()
  What's the url for this database?
- **getUserName**()
  What's our user name as known to the database?
- **getVersionColumns**(String, String, String)
  Get a description of a table's columns that are automatically updated when any value in a row is updated.
- **isCatalogAtStart**()
  Does a catalog appear at the start of a qualified table name? (Otherwise it appears at the end)
- **isReadOnly**()
  Is the database in read-only mode?
- **nullPlusNonNullIsNull**()
  Are concatenations between NULL and non-NULL values NULL? A JDBC-Compliant driver

always returns true.

- **nullsAreSortedAtEnd**()
  Are NULL values sorted at the end regardless of sort order?
- **nullsAreSortedAtStart**()
  Are NULL values sorted at the start regardless of sort order?
- **nullsAreSortedHigh**()
  Are NULL values sorted high?
- **nullsAreSortedLow**()
  Are NULL values sorted low?
- **storesLowerCaseIdentifiers**()
  Does the database treat mixed case unquoted SQL identifiers as case insensitive and store them in lower case?
- **storesLowerCaseQuotedIdentifiers**()
  Does the database treat mixed case quoted SQL identifiers as case insensitive and store them in lower case?
- **storesMixedCaseIdentifiers**()
  Does the database treat mixed case unquoted SQL identifiers as case insensitive and store them in mixed case?
- **storesMixedCaseQuotedIdentifiers**()
  Does the database treat mixed case quoted SQL identifiers as case insensitive and store them in mixed case?
- **storesUpperCaseIdentifiers**()
  Does the database treat mixed case unquoted SQL identifiers as case insensitive and store them in upper case?
- **storesUpperCaseQuotedIdentifiers**()
  Does the database treat mixed case quoted SQL identifiers as case insensitive and store them in upper case?
- **supportsAlterTableWithAddColumn**()
  Is "ALTER TABLE" with add column supported?
- **supportsAlterTableWithDropColumn**()
  Is "ALTER TABLE" with drop column supported?
- **supportsANSI92EntryLevelSQL**()
  Is the ANSI92 entry level SQL grammar supported? All JDBC-Compliant drivers must return true.
- **supportsANSI92FullSQL**()
  Is the ANSI92 full SQL grammar supported?
- **supportsANSI92IntermediateSQL**()
  Is the ANSI92 intermediate SQL grammar supported?
- **supportsCatalogsInDataManipulation**()
  Can a catalog name be used in a data manipulation statement?
- **supportsCatalogsInIndexDefinitions**()
  Can a catalog name be used in an index definition statement?
- **supportsCatalogsInPrivilegeDefinitions**()
  Can a catalog name be used in a privilege definition statement?
- **supportsCatalogsInProcedureCalls**()
  Can a catalog name be used in a procedure call statement?
- **supportsCatalogsInTableDefinitions**()
  Can a catalog name be used in a table definition statement?

- **supportsColumnAliasing**()
  Is column aliasing supported?

  If so, the SQL AS clause can be used to provide names for computed columns or to provide alias names for columns as required.

- **supportsConvert**()
  Is the CONVERT function between SQL types supported?
- **supportsConvert**(int, int)
  Is CONVERT between the given SQL types supported?
- **supportsCoreSQLGrammar**()
  Is the ODBC Core SQL grammar supported?
- **supportsCorrelatedSubqueries**()
  Are correlated subqueries supported? A JDBC-Compliant driver always returns true.
- **supportsDataDefinitionAndDataManipulationTransactions**()
  Are both data definition and data manipulation statements within a transaction supported?
- **supportsDataManipulationTransactionsOnly**()
  Are only data manipulation statements within a transaction supported?
- **supportsDifferentTableCorrelationNames**()
  If table correlation names are supported, are they restricted to be different from the names of the tables?
- **supportsExpressionsInOrderBy**()
  Are expressions in "ORDER BY" lists supported?
- **supportsExtendedSQLGrammar**()
  Is the ODBC Extended SQL grammar supported?
- **supportsFullOuterJoins**()
  Are full nested outer joins supported?
- **supportsGroupBy**()
  Is some form of "GROUP BY" clause supported?
- **supportsGroupByBeyondSelect**()
  Can a "GROUP BY" clause add columns not in the SELECT provided it specifies all the columns in the SELECT?
- **supportsGroupByUnrelated**()
  Can a "GROUP BY" clause use columns not in the SELECT?
- **supportsIntegrityEnhancementFacility**()
  Is the SQL Integrity Enhancement Facility supported?
- **supportsLikeEscapeClause**()
  Is the escape character in "LIKE" clauses supported? A JDBC-Compliant driver always returns true.
- **supportsLimitedOuterJoins**()
  Is there limited support for outer joins? (This will be true if supportFullOuterJoins is true.)
- **supportsMinimumSQLGrammar**()
  Is the ODBC Minimum SQL grammar supported? All JDBC-Compliant drivers must return true.
- **supportsMixedCaseIdentifiers**()
  Does the database treat mixed case unquoted SQL identifiers as case sensitive and as a result store them in mixed case? A JDBC-Compliant driver will always return false.
- **supportsMixedCaseQuotedIdentifiers**()

Does the database treat mixed case quoted SQL identifiers as case sensitive and as a result store them in mixed case? A JDBC-Compliant driver will always return true.

- **supportsMultipleResultSets**()
  Are multiple ResultSets from a single execute supported?
- **supportsMultipleTransactions**()
  Can we have multiple transactions open at once (on different connections)?
- **supportsNonNullableColumns**()
  Can columns be defined as non-nullable? A JDBC-Compliant driver always returns true.
- **supportsOpenCursorsAcrossCommit**()
  Can cursors remain open across commits?
- **supportsOpenCursorsAcrossRollback**()
  Can cursors remain open across rollbacks?
- **supportsOpenStatementsAcrossCommit**()
  Can statements remain open across commits?
- **supportsOpenStatementsAcrossRollback**()
  Can statements remain open across rollbacks?
- **supportsOrderByUnrelated**()
  Can an "ORDER BY" clause use columns not in the SELECT?
- **supportsOuterJoins**()
  Is some form of outer join supported?
- **supportsPositionedDelete**()
  Is positioned DELETE supported?
- **supportsPositionedUpdate**()
  Is positioned UPDATE supported?
- **supportsSchemasInDataManipulation**()
  Can a schema name be used in a data manipulation statement?
- **supportsSchemasInIndexDefinitions**()
  Can a schema name be used in an index definition statement?
- **supportsSchemasInPrivilegeDefinitions**()
  Can a schema name be used in a privilege definition statement?
- **supportsSchemasInProcedureCalls**()
  Can a schema name be used in a procedure call statement?
- **supportsSchemasInTableDefinitions**()
  Can a schema name be used in a table definition statement?
- **supportsSelectForUpdate**()
  Is SELECT for UPDATE supported?
- **supportsStoredProcedures**()
  Are stored procedure calls using the stored procedure escape syntax supported?
- **supportsSubqueriesInComparisons**()
  Are subqueries in comparison expressions supported? A JDBC-Compliant driver always returns true.
- **supportsSubqueriesInExists**()
  Are subqueries in ’exists’ expressions supported? A JDBC-Compliant driver always returns true.
- **supportsSubqueriesInIns**()
  Are subqueries in ’in’ statements supported? A JDBC-Compliant driver always returns true.
- **supportsSubqueriesInQuantifieds**()
  Are subqueries in quantified expressions supported? A JDBC-Compliant driver always returns

true.

- **supportsTableCorrelationNames**()
  Are table correlation names supported? A JDBC-Compliant driver always returns true.
- **supportsTransactionIsolationLevel**(int)
  Does the database support the given transaction isolation level?
- **supportsTransactions**()
  Are transactions supported? If not, commit is a noop and the isolation level is TRANSACTION_NONE.
- **supportsUnion**()
  Is SQL UNION supported?
- **supportsUnionAll**()
  Is SQL UNION ALL supported?
- **usesLocalFilePerTable**()
  Does the database use a file for each table?
- **usesLocalFiles**()
  Does the database store tables in a local file?

Variables

- **procedureResultUnknown**

```
public final static int procedureResultUnknown
```

  PROCEDURE_TYPE - May return a result.

- **procedureNoResult**

```
public final static int procedureNoResult
```

  PROCEDURE_TYPE - Does not return a result.

- **procedureReturnsResult**

```
public final static int procedureReturnsResult
```

  PROCEDURE_TYPE - Returns a result.

- **procedureColumnUnknown**

```
public final static int procedureColumnUnknown
```

  COLUMN_TYPE - nobody knows.

- **procedureColumnIn**

```
public final static int procedureColumnIn
```

  COLUMN_TYPE - IN parameter.

- **procedureColumnInOut**

  `public final static int procedureColumnInOut`

  COLUMN_TYPE - INOUT parameter.

- **procedureColumnOut**

  `public final static int procedureColumnOut`

  COLUMN_TYPE - OUT parameter.

- **procedureColumnReturn**

  `public final static int procedureColumnReturn`

  COLUMN_TYPE - procedure return value.

- **procedureColumnResult**

  `public final static int procedureColumnResult`

  COLUMN_TYPE - result column in ResultSet.

- **procedureNoNulls**

  `public final static int procedureNoNulls`

  TYPE NULLABLE - does not allow NULL values.

- **procedureNullable**

  `public final static int procedureNullable`

  TYPE NULLABLE - allows NULL values.

- **procedureNullableUnknown**

  `public final static int procedureNullableUnknown`

  TYPE NULLABLE - nullability unknown.

- **columnNoNulls**

  `public final static int columnNoNulls`

  COLUMN NULLABLE - might not allow NULL values.

- **columnNullable**

  `public final static int columnNullable`

COLUMN NULLABLE - definitely allows NULL values.

- **columnNullableUnknown**

```
public final static int columnNullableUnknown
```

COLUMN NULLABLE - nullability unknown.

- **bestRowTemporary**

```
public final static int bestRowTemporary
```

BEST ROW SCOPE - very temporary, while using row.

- **bestRowTransaction**

```
public final static int bestRowTransaction
```

BEST ROW SCOPE - valid for remainder of current transaction.

- **bestRowSession**

```
public final static int bestRowSession
```

BEST ROW SCOPE - valid for remainder of current session.

- **bestRowUnknown**

```
public final static int bestRowUnknown
```

BEST ROW PSEUDO_COLUMN - may or may not be pseudo column.

- **bestRowNotPseudo**

```
public final static int bestRowNotPseudo
```

BEST ROW PSEUDO_COLUMN - is NOT a pseudo column.

- **bestRowPseudo**

```
public final static int bestRowPseudo
```

BEST ROW PSEUDO_COLUMN - is a pseudo column.

- **versionColumnUnknown**

```
public final static int versionColumnUnknown
```

VERSION COLUMNS PSEUDO_COLUMN - may or may not be pseudo column.

- **versionColumnNotPseudo**

```
public final static int versionColumnNotPseudo
```

VERSION COLUMNS PSEUDO_COLUMN - is NOT a pseudo column.

- **versionColumnPseudo**

```
public final static int versionColumnPseudo
```

VERSION COLUMNS PSEUDO_COLUMN - is a pseudo column.

- **importedKeyCascade**

```
public final static int importedKeyCascade
```

IMPORT KEY UPDATE_RULE and DELETE_RULE - for update, change imported key to agree with primary key update; for delete, delete rows that import a deleted key.

- **importedKeyRestrict**

```
public final static int importedKeyRestrict
```

IMPORT KEY UPDATE_RULE and DELETE_RULE - do not allow update or delete of primary key if it has been imported.

- **importedKeySetNull**

```
public final static int importedKeySetNull
```

IMPORT KEY UPDATE_RULE and DELETE_RULE - change imported key to NULL if its primary key has been updated or deleted.

- **importedKeyNoAction**

```
public final static int importedKeyNoAction
```

IMPORT KEY UPDATE_RULE and DELETE_RULE - do not allow update or delete of primary key if it has been imported.

- **importedKeySetDefault**

```
public final static int importedKeySetDefault
```

IMPORT KEY UPDATE_RULE and DELETE_RULE - change imported key to default values if its primary key has been updated or deleted.

- **importedKeyInitiallyDeferred**

```
public final static int importedKeyInitiallyDeferred
```

IMPORT KEY DEFERRABILITY - see SQL92 for definition

- **importedKeyInitiallyImmediate**

  `public final static int importedKeyInitiallyImmediate`

  IMPORT KEY DEFERRABILITY - see SQL92 for definition

- **importedKeyNotDeferrable**

  `public final static int importedKeyNotDeferrable`

  IMPORT KEY DEFERRABILITY - see SQL92 for definition

- **typeNoNulls**

  `public final static int typeNoNulls`

  TYPE NULLABLE - does not allow NULL values.

- **typeNullable**

  `public final static int typeNullable`

  TYPE NULLABLE - allows NULL values.

- **typeNullableUnknown**

  `public final static int typeNullableUnknown`

  TYPE NULLABLE - nullability unknown.

- **typePredNone**

  `public final static int typePredNone`

  TYPE INFO SEARCHABLE - No support.

- **typePredChar**

  `public final static int typePredChar`

  TYPE INFO SEARCHABLE - Only supported with WHERE .. LIKE.

- **typePredBasic**

  `public final static int typePredBasic`

  TYPE INFO SEARCHABLE - Supported except for WHERE .. LIKE.

- **typeSearchable**

  `public final static int typeSearchable`

TYPE INFO SEARCHABLE - Supported for all WHERE ...

- **tableIndexStatistic**

```
public final static short tableIndexStatistic
```

INDEX INFO TYPE - this identifies table statistics that are returned in conjuction with a table's index descriptions

- **tableIndexClustered**

```
public final static short tableIndexClustered
```

INDEX INFO TYPE - this identifies a clustered index

- **tableIndexHashed**

```
public final static short tableIndexHashed
```

INDEX INFO TYPE - this identifies a hashed index

- **tableIndexOther**

```
public final static short tableIndexOther
```

INDEX INFO TYPE - this identifies some other form of index

- **allProceduresAreCallable**

```
public abstract boolean allProceduresAreCallable() throws SQLException
```

Can all the procedures returned by getProcedures be called by the current user?
Returns:
true if so

- **allTablesAreSelectable**

```
public abstract boolean allTablesAreSelectable() throws SQLException
```

Can all the tables returned by getTable be SELECTed by the current user?
Returns:
true if so

- **getURL**

```
public abstract String getURL() throws SQLException
```

What's the url for this database?

Returns:
the url or null if it can't be generated

- **getUserName**

```
public abstract String getUserName() throws SQLException
```

What's our user name as known to the database?
Returns:
our database user name

- **isReadOnly**

```
public abstract boolean isReadOnly() throws SQLException
```

Is the database in read-only mode?
Returns:
true if so

- **nullsAreSortedHigh**

```
public abstract boolean nullsAreSortedHigh() throws SQLException
```

Are NULL values sorted high?
Returns:
true if so

- **nullsAreSortedLow**

```
public abstract boolean nullsAreSortedLow() throws SQLException
```

Are NULL values sorted low?
Returns:
true if so

- **nullsAreSortedAtStart**

```
public abstract boolean nullsAreSortedAtStart() throws SQLException
```

Are NULL values sorted at the start regardless of sort order?
Returns:
true if so

- **nullsAreSortedAtEnd**

```
public abstract boolean nullsAreSortedAtEnd() throws SQLException
```

Are NULL values sorted at the end regardless of sort order?
Returns:
true if so

- **getDatabaseProductName**

```
public abstract String getDatabaseProductName() throws SQLException
```

  What's the name of this database product?
  Returns:
    database product name

- **getDatabaseProductVersion**

```
public abstract String getDatabaseProductVersion() throws SQLException
```

  What's the version of this database product?
  Returns:
    database version

- **getDriverName**

```
public abstract String getDriverName() throws SQLException
```

  What's the name of this JDBC driver?
  Returns:
    JDBC driver name

- **getDriverVersion**

```
public abstract String getDriverVersion() throws SQLException
```

  What's the version of this JDBC driver?
  Returns:
    JDBC driver version

- **getDriverMajorVersion**

```
public abstract int getDriverMajorVersion()
```

  What's this JDBC driver's major version number?
  Returns:
    JDBC driver major version

- **getDriverMinorVersion**

```
public abstract int getDriverMinorVersion()
```

  What's this JDBC driver's minor version number?
  Returns:
    JDBC driver minor version number

- **usesLocalFiles**

```
public abstract boolean usesLocalFiles() throws SQLException
```

Does the database store tables in a local file?
Returns:
true if so

- **usesLocalFilePerTable**

```
public abstract boolean usesLocalFilePerTable() throws SQLException
```

Does the database use a file for each table?
Returns:
true if the database uses a local file for each table

- **supportsMixedCaseIdentifiers**

```
public abstract boolean supportsMixedCaseIdentifiers() throws SQLException
```

Does the database treat mixed case unquoted SQL identifiers as case sensitive and as a result store them in mixed case? A JDBC-Compliant driver will always return false.
Returns:
true if so

- **storesUpperCaseIdentifiers**

```
public abstract boolean storesUpperCaseIdentifiers() throws SQLException
```

Does the database treat mixed case unquoted SQL identifiers as case insensitive and store them in upper case?
Returns:
true if so

- **storesLowerCaseIdentifiers**

```
public abstract boolean storesLowerCaseIdentifiers() throws SQLException
```

Does the database treat mixed case unquoted SQL identifiers as case insensitive and store them in lower case?
Returns:
true if so

- **storesMixedCaseIdentifiers**

```
public abstract boolean storesMixedCaseIdentifiers() throws SQLException
```

Does the database treat mixed case unquoted SQL identifiers as case insensitive and store them in mixed case?
Returns:
true if so

- **supportsMixedCaseQuotedIdentifiers**

```
public abstract boolean supportsMixedCaseQuotedIdentifiers() throws SQLException
```

  Does the database treat mixed case quoted SQL identifiers as case sensitive and as a result store them in mixed case? A JDBC-Compliant driver will always return true.
  Returns:
  true if so

- **storesUpperCaseQuotedIdentifiers**

```
public abstract boolean storesUpperCaseQuotedIdentifiers() throws SQLException
```

  Does the database treat mixed case quoted SQL identifiers as case insensitive and store them in upper case?
  Returns:
  true if so

- **storesLowerCaseQuotedIdentifiers**

```
public abstract boolean storesLowerCaseQuotedIdentifiers() throws SQLException
```

  Does the database treat mixed case quoted SQL identifiers as case insensitive and store them in lower case?
  Returns:
  true if so

- **storesMixedCaseQuotedIdentifiers**

```
public abstract boolean storesMixedCaseQuotedIdentifiers() throws SQLException
```

  Does the database treat mixed case quoted SQL identifiers as case insensitive and store them in mixed case?
  Returns:
  true if so

- **getIdentifierQuoteString**

```
public abstract String getIdentifierQuoteString() throws SQLException
```

  What's the string used to quote SQL identifiers? This returns a space " " if identifier quoting isn't supported. A JDBC-Compliant driver always uses a double quote character.
  Returns:
  the quoting string

- **getSQLKeywords**

```
public abstract String getSQLKeywords() throws SQLException
```

  Get a comma separated list of all a database's SQL keywords that are NOT also SQL92 keywords.

Returns:
the list

- **getNumericFunctions**

```
public abstract String getNumericFunctions() throws SQLException
```

Get a comma separated list of math functions.
Returns:
the list

- **getStringFunctions**

```
public abstract String getStringFunctions() throws SQLException
```

Get a comma separated list of string functions.
Returns:
the list

- **getSystemFunctions**

```
public abstract String getSystemFunctions() throws SQLException
```

Get a comma separated list of system functions.
Returns:
the list

- **getTimeDateFunctions**

```
public abstract String getTimeDateFunctions() throws SQLException
```

Get a comma separated list of time and date functions.
Returns:
the list

- **getSearchStringEscape**

```
public abstract String getSearchStringEscape() throws SQLException
```

This is the string that can be used to escape '_' or '%' in the string pattern style catalog search parameters.

The '_' character represents any single character.

The '%' character represents any sequence of zero or more characters.

Returns:
the string used to escape wildcard characters

- **getExtraNameCharacters**

```
public abstract String getExtraNameCharacters() throws SQLException
```

Get all the "extra" characters that can be used in unquoted identifier names (those beyond a-z, A-Z, 0-9 and _).

Returns:

the string containing the extra characters

- **supportsAlterTableWithAddColumn**

```
public abstract boolean supportsAlterTableWithAddColumn() throws SQLException
```

Is "ALTER TABLE" with add column supported?

Returns:

true if so

- **supportsAlterTableWithDropColumn**

```
public abstract boolean supportsAlterTableWithDropColumn() throws SQLException
```

Is "ALTER TABLE" with drop column supported?

Returns:

true if so

- **supportsColumnAliasing**

```
public abstract boolean supportsColumnAliasing() throws SQLException
```

Is column aliasing supported?

If so, the SQL AS clause can be used to provide names for computed columns or to provide alias names for columns as required. A JDBC-Compliant driver always returns true.

Returns:

true if so

- **nullPlusNonNullIsNull**

```
public abstract boolean nullPlusNonNullIsNull() throws SQLException
```

Are concatenations between NULL and non-NULL values NULL? A JDBC-Compliant driver always returns true.

Returns:

true if so

- **supportsConvert**

```
public abstract boolean supportsConvert() throws SQLException
```

Is the CONVERT function between SQL types supported?

Returns:

true if so

- **supportsConvert**

```
public abstract boolean supportsConvert(int fromType,
                                        int toType) throws SQLException
```

Is CONVERT between the given SQL types supported?

Parameters:

fromType - the type to convert from
toType - the type to convert to

Returns:

true if so

See Also:

Types

- **supportsTableCorrelationNames**

```
public abstract boolean supportsTableCorrelationNames() throws SQLException
```

Are table correlation names supported? A JDBC-Compliant driver always returns true.

Returns:

true if so

- **supportsDifferentTableCorrelationNames**

```
public abstract boolean supportsDifferentTableCorrelationNames() throws SQLExcepti
```

If table correlation names are supported, are they restricted to be different from the names of the tables?

Returns:

true if so

- **supportsExpressionsInOrderBy**

```
public abstract boolean supportsExpressionsInOrderBy() throws SQLException
```

Are expressions in "ORDER BY" lists supported?

Returns:

true if so

- **supportsOrderByUnrelated**

```
public abstract boolean supportsOrderByUnrelated() throws SQLException
```

Can an "ORDER BY" clause use columns not in the SELECT?

Returns:

true if so

- **supportsGroupBy**

```
public abstract boolean supportsGroupBy() throws SQLException
```

Is some form of "GROUP BY" clause supported?
Returns:
true if so

- **supportsGroupByUnrelated**

```
public abstract boolean supportsGroupByUnrelated() throws SQLException
```

Can a "GROUP BY" clause use columns not in the SELECT?
Returns:
true if so

- **supportsGroupByBeyondSelect**

```
public abstract boolean supportsGroupByBeyondSelect() throws SQLException
```

Can a "GROUP BY" clause add columns not in the SELECT provided it specifies all the columns in the SELECT?
Returns:
true if so

- **supportsLikeEscapeClause**

```
public abstract boolean supportsLikeEscapeClause() throws SQLException
```

Is the escape character in "LIKE" clauses supported? A JDBC-Compliant driver always returns true.
Returns:
true if so

- **supportsMultipleResultSets**

```
public abstract boolean supportsMultipleResultSets() throws SQLException
```

Are multiple ResultSets from a single execute supported?
Returns:
true if so

- **supportsMultipleTransactions**

```
public abstract boolean supportsMultipleTransactions() throws SQLException
```

Can we have multiple transactions open at once (on different connections)?
Returns:
true if so

- **supportsNonNullableColumns**

```
public abstract boolean supportsNonNullableColumns() throws SQLException
```

Can columns be defined as non-nullable? A JDBC-Compliant driver always returns true.
Returns:
true if so

- **supportsMinimumSQLGrammar**

```
public abstract boolean supportsMinimumSQLGrammar() throws SQLException
```

Is the ODBC Minimum SQL grammar supported? All JDBC-Compliant drivers must return true.
Returns:
true if so

- **supportsCoreSQLGrammar**

```
public abstract boolean supportsCoreSQLGrammar() throws SQLException
```

Is the ODBC Core SQL grammar supported?
Returns:
true if so

- **supportsExtendedSQLGrammar**

```
public abstract boolean supportsExtendedSQLGrammar() throws SQLException
```

Is the ODBC Extended SQL grammar supported?
Returns:
true if so

- **supportsANSI92EntryLevelSQL**

```
public abstract boolean supportsANSI92EntryLevelSQL() throws SQLException
```

Is the ANSI92 entry level SQL grammar supported? All JDBC-Compliant drivers must return true.
Returns:
true if so

- **supportsANSI92IntermediateSQL**

```
public abstract boolean supportsANSI92IntermediateSQL() throws SQLException
```

Is the ANSI92 intermediate SQL grammar supported?
Returns:
true if so

- **supportsANSI92FullSQL**

```
public abstract boolean supportsANSI92FullSQL() throws SQLException
```

Is the ANSI92 full SQL grammar supported?

Returns:
true if so

- **supportsIntegrityEnhancementFacility**

```
public abstract boolean supportsIntegrityEnhancementFacility() throws SQLException
```

Is the SQL Integrity Enhancement Facility supported?
Returns:
true if so

- **supportsOuterJoins**

```
public abstract boolean supportsOuterJoins() throws SQLException
```

Is some form of outer join supported?
Returns:
true if so

- **supportsFullOuterJoins**

```
public abstract boolean supportsFullOuterJoins() throws SQLException
```

Are full nested outer joins supported?
Returns:
true if so

- **supportsLimitedOuterJoins**

```
public abstract boolean supportsLimitedOuterJoins() throws SQLException
```

Is there limited support for outer joins? (This will be true if supportFullOuterJoins is true.)
Returns:
true if so

- **getSchemaTerm**

```
public abstract String getSchemaTerm() throws SQLException
```

What’s the database vendor’s preferred term for "schema"?
Returns:
the vendor term

- **getProcedureTerm**

```
public abstract String getProcedureTerm() throws SQLException
```

What’s the database vendor’s preferred term for "procedure"?
Returns:
the vendor term

- **getCatalogTerm**

```
public abstract String getCatalogTerm() throws SQLException
```

  What's the database vendor's preferred term for "catalog"?
  Returns:
    the vendor term

- **isCatalogAtStart**

```
public abstract boolean isCatalogAtStart() throws SQLException
```

  Does a catalog appear at the start of a qualified table name? (Otherwise it appears at the end)
  Returns:
    true if it appears at the start

- **getCatalogSeparator**

```
public abstract String getCatalogSeparator() throws SQLException
```

  What's the separator between catalog and table name?
  Returns:
    the separator string

- **supportsSchemasInDataManipulation**

```
public abstract boolean supportsSchemasInDataManipulation() throws SQLException
```

  Can a schema name be used in a data manipulation statement?
  Returns:
    true if so

- **supportsSchemasInProcedureCalls**

```
public abstract boolean supportsSchemasInProcedureCalls() throws SQLException
```

  Can a schema name be used in a procedure call statement?
  Returns:
    true if so

- **supportsSchemasInTableDefinitions**

```
public abstract boolean supportsSchemasInTableDefinitions() throws SQLException
```

  Can a schema name be used in a table definition statement?
  Returns:
    true if so

- **supportsSchemasInIndexDefinitions**

```
public abstract boolean supportsSchemasInIndexDefinitions() throws SQLException
```

Can a schema name be used in an index definition statement?
Returns:
true if so

- **supportsSchemasInPrivilegeDefinitions**

```
public abstract boolean supportsSchemasInPrivilegeDefinitions() throws SQLExceptio
```

Can a schema name be used in a privilege definition statement?
Returns:
true if so

- **supportsCatalogsInDataManipulation**

```
public abstract boolean supportsCatalogsInDataManipulation() throws SQLException
```

Can a catalog name be used in a data manipulation statement?
Returns:
true if so

- **supportsCatalogsInProcedureCalls**

```
public abstract boolean supportsCatalogsInProcedureCalls() throws SQLException
```

Can a catalog name be used in a procedure call statement?
Returns:
true if so

- **supportsCatalogsInTableDefinitions**

```
public abstract boolean supportsCatalogsInTableDefinitions() throws SQLException
```

Can a catalog name be used in a table definition statement?
Returns:
true if so

- **supportsCatalogsInIndexDefinitions**

```
public abstract boolean supportsCatalogsInIndexDefinitions() throws SQLException
```

Can a catalog name be used in an index definition statement?
Returns:
true if so

- **supportsCatalogsInPrivilegeDefinitions**

```
public abstract boolean supportsCatalogsInPrivilegeDefinitions() throws SQLExcepti
```

Can a catalog name be used in a privilege definition statement?

Returns:
true if so

- **supportsPositionedDelete**

```
public abstract boolean supportsPositionedDelete() throws SQLException
```

Is positioned DELETE supported?
Returns:
true if so

- **supportsPositionedUpdate**

```
public abstract boolean supportsPositionedUpdate() throws SQLException
```

Is positioned UPDATE supported?
Returns:
true if so

- **supportsSelectForUpdate**

```
public abstract boolean supportsSelectForUpdate() throws SQLException
```

Is SELECT for UPDATE supported?
Returns:
true if so

- **supportsStoredProcedures**

```
public abstract boolean supportsStoredProcedures() throws SQLException
```

Are stored procedure calls using the stored procedure escape syntax supported?
Returns:
true if so

- **supportsSubqueriesInComparisons**

```
public abstract boolean supportsSubqueriesInComparisons() throws SQLException
```

Are subqueries in comparison expressions supported? A JDBC-Compliant driver always returns true.
Returns:
true if so

- **supportsSubqueriesInExists**

```
public abstract boolean supportsSubqueriesInExists() throws SQLException
```

Are subqueries in ’exists’ expressions supported? A JDBC-Compliant driver always returns true.
Returns:

true if so

- **supportsSubqueriesInIns**

```
public abstract boolean supportsSubqueriesInIns() throws SQLException
```

Are subqueries in 'in' statements supported? A JDBC-Compliant driver always returns true.
Returns:
true if so

- **supportsSubqueriesInQuantifieds**

```
public abstract boolean supportsSubqueriesInQuantifieds() throws SQLException
```

Are subqueries in quantified expressions supported? A JDBC-Compliant driver always returns true.
Returns:
true if so

- **supportsCorrelatedSubqueries**

```
public abstract boolean supportsCorrelatedSubqueries() throws SQLException
```

Are correlated subqueries supported? A JDBC-Compliant driver always returns true.
Returns:
true if so

- **supportsUnion**

```
public abstract boolean supportsUnion() throws SQLException
```

Is SQL UNION supported?
Returns:
true if so

- **supportsUnionAll**

```
public abstract boolean supportsUnionAll() throws SQLException
```

Is SQL UNION ALL supported?
Returns:
true if so

- **supportsOpenCursorsAcrossCommit**

```
public abstract boolean supportsOpenCursorsAcrossCommit() throws SQLException
```

Can cursors remain open across commits?
Returns:
true if cursors always remain open; false if they might not remain open

- **supportsOpenCursorsAcrossRollback**

```
public abstract boolean supportsOpenCursorsAcrossRollback() throws SQLException
```

  Can cursors remain open across rollbacks?
  Returns:
  true if cursors always remain open; false if they might not remain open

- **supportsOpenStatementsAcrossCommit**

```
public abstract boolean supportsOpenStatementsAcrossCommit() throws SQLException
```

  Can statements remain open across commits?
  Returns:
  true if statements always remain open; false if they might not remain open

- **supportsOpenStatementsAcrossRollback**

```
public abstract boolean supportsOpenStatementsAcrossRollback() throws SQLException
```

  Can statements remain open across rollbacks?
  Returns:
  true if statements always remain open; false if they might not remain open

- **getMaxBinaryLiteralLength**

```
public abstract int getMaxBinaryLiteralLength() throws SQLException
```

  How many hex characters can you have in an inline binary literal?
  Returns:
  max literal length

- **getMaxCharLiteralLength**

```
public abstract int getMaxCharLiteralLength() throws SQLException
```

  What’s the max length for a character literal?
  Returns:
  max literal length

- **getMaxColumnNameLength**

```
public abstract int getMaxColumnNameLength() throws SQLException
```

  What’s the limit on column name length?
  Returns:
  max literal length

- **getMaxColumnsInGroupBy**

```
public abstract int getMaxColumnsInGroupBy() throws SQLException
```

What's the maximum number of columns in a "GROUP BY" clause?

Returns:

max number of columns

- **getMaxColumnsInIndex**

```
public abstract int getMaxColumnsInIndex() throws SQLException
```

What's the maximum number of columns allowed in an index?

Returns:

max columns

- **getMaxColumnsInOrderBy**

```
public abstract int getMaxColumnsInOrderBy() throws SQLException
```

What's the maximum number of columns in an "ORDER BY" clause?

Returns:

max columns

- **getMaxColumnsInSelect**

```
public abstract int getMaxColumnsInSelect() throws SQLException
```

What's the maximum number of columns in a "SELECT" list?

Returns:

max columns

- **getMaxColumnsInTable**

```
public abstract int getMaxColumnsInTable() throws SQLException
```

What's the maximum number of columns in a table?

Returns:

max columns

- **getMaxConnections**

```
public abstract int getMaxConnections() throws SQLException
```

How many active connections can we have at a time to this database?

Returns:

max connections

- **getMaxCursorNameLength**

```
public abstract int getMaxCursorNameLength() throws SQLException
```

What's the maximum cursor name length?

Returns:
max cursor name length in bytes

- **getMaxIndexLength**

```
public abstract int getMaxIndexLength() throws SQLException
```

What's the maximum length of an index (in bytes)?
Returns:
max index length in bytes

- **getMaxSchemaNameLength**

```
public abstract int getMaxSchemaNameLength() throws SQLException
```

What's the maximum length allowed for a schema name?
Returns:
max name length in bytes

- **getMaxProcedureNameLength**

```
public abstract int getMaxProcedureNameLength() throws SQLException
```

What's the maximum length of a procedure name?
Returns:
max name length in bytes

- **getMaxCatalogNameLength**

```
public abstract int getMaxCatalogNameLength() throws SQLException
```

What's the maximum length of a catalog name?
Returns:
max name length in bytes

- **getMaxRowSize**

```
public abstract int getMaxRowSize() throws SQLException
```

What's the maximum length of a single row?
Returns:
max row size in bytes

- **doesMaxRowSizeIncludeBlobs**

```
public abstract boolean doesMaxRowSizeIncludeBlobs() throws SQLException
```

Did getMaxRowSize() include LONGVARCHAR and LONGVARBINARY blobs?
Returns:
true if so

- **getMaxStatementLength**

```
public abstract int getMaxStatementLength() throws SQLException
```

  What's the maximum length of a SQL statement?
  Returns:
    max length in bytes

- **getMaxStatements**

```
public abstract int getMaxStatements() throws SQLException
```

  How many active statements can we have open at one time to this database?
  Returns:
    the maximum

- **getMaxTableNameLength**

```
public abstract int getMaxTableNameLength() throws SQLException
```

  What's the maximum length of a table name?
  Returns:
    max name length in bytes

- **getMaxTablesInSelect**

```
public abstract int getMaxTablesInSelect() throws SQLException
```

  What's the maximum number of tables in a SELECT?
  Returns:
    the maximum

- **getMaxUserNameLength**

```
public abstract int getMaxUserNameLength() throws SQLException
```

  What's the maximum length of a user name?
  Returns:
    max name length in bytes

- **getDefaultTransactionIsolation**

```
public abstract int getDefaultTransactionIsolation() throws SQLException
```

  What's the database's default transaction isolation level? The values are defined in java.sql.Connection.
  Returns:
    the default isolation level
  See Also:
    Connection

- **supportsTransactions**

```
public abstract boolean supportsTransactions() throws SQLException
```

  Are transactions supported? If not, commit is a noop and the isolation level is TRANSACTION_NONE.
  Returns:
  true if transactions are supported

- **supportsTransactionIsolationLevel**

```
public abstract boolean supportsTransactionIsolationLevel(int level) throws SQLExc
```

  Does the database support the given transaction isolation level?
  Parameters:
  level - the values are defined in java.sql.Connection
  Returns:
  true if so
  See Also:
  Connection

- **supportsDataDefinitionAndDataManipulationTransactions**

```
public abstract boolean supportsDataDefinitionAndDataManipulationTransactions() th
```

  Are both data definition and data manipulation statements within a transaction supported?
  Returns:
  true if so

- **supportsDataManipulationTransactionsOnly**

```
public abstract boolean supportsDataManipulationTransactionsOnly() throws SQLExcep
```

  Are only data manipulation statements within a transaction supported?
  Returns:
  true if so

- **dataDefinitionCausesTransactionCommit**

```
public abstract boolean dataDefinitionCausesTransactionCommit() throws SQLExceptio
```

  Does a data definition statement within a transaction force the transaction to commit?
  Returns:
  true if so

- **dataDefinitionIgnoredInTransactions**

```
public abstract boolean dataDefinitionIgnoredInTransactions() throws SQLException
```

  Is a data definition statement within a transaction ignored?

Returns:
true if so

- **getProcedures**

```
public abstract ResultSet getProcedures(String catalog,
                                        String schemaPattern,
                                        String procedureNamePattern) throws SQLExc
```

Get a description of stored procedures available in a catalog.

Only procedure descriptions matching the schema and procedure name criteria are returned. They are ordered by PROCEDURE_SCHEM, and PROCEDURE_NAME.

Each procedure description has the the following columns:

1. **PROCEDURE_CAT** String => procedure catalog (may be null)
2. **PROCEDURE_SCHEM** String => procedure schema (may be null)
3. **PROCEDURE_NAME** String => procedure name
4. reserved for future use
5. reserved for future use
6. reserved for future use
7. **REMARKS** String => explanatory comment on the procedure
8. **PROCEDURE_TYPE** short => kind of procedure:
   - procedureResultUnknown - May return a result
   - procedureNoResult - Does not return a result
   - procedureReturnsResult - Returns a result

Parameters:
catalog - a catalog name; "" retrieves those without a catalog; null means drop catalog name from the selection criteria
schemaPattern - a schema name pattern; "" retrieves those without a schema
procedureNamePattern - a procedure name pattern
Returns:
ResultSet - each row is a procedure description
See Also:
getSearchStringEscape

- **getProcedureColumns**

```
public abstract ResultSet getProcedureColumns(String catalog,
                                              String schemaPattern,
                                              String procedureNamePattern,
                                              String columnNamePattern) throws SQL
```

Get a description of a catalog's stored procedure parameters and result columns.

Only descriptions matching the schema, procedure and parameter name criteria are returned. They are ordered by PROCEDURE_SCHEM and PROCEDURE_NAME. Within this, the return value, if any, is first. Next are the parameter descriptions in call order. The column descriptions follow in column number order.

Each row in the ResultSet is a parameter description or column description with the following fields:

1. **PROCEDURE_CAT** String => procedure catalog (may be null)
2. **PROCEDURE_SCHEM** String => procedure schema (may be null)
3. **PROCEDURE_NAME** String => procedure name
4. **COLUMN_NAME** String => column/parameter name
5. **COLUMN_TYPE** Short => kind of column/parameter:
   - procedureColumnUnknown - nobody knows
   - procedureColumnIn - IN parameter
   - procedureColumnInOut - INOUT parameter
   - procedureColumnOut - OUT parameter
   - procedureColumnReturn - procedure return value
   - procedureColumnResult - result column in ResultSet
6. **DATA_TYPE** short => SQL type from java.sql.Types
7. **TYPE_NAME** String => SQL type name
8. **PRECISION** int => precision
9. **LENGTH** int => length in bytes of data
10. **SCALE** short => scale
11. **RADIX** short => radix
12. **NULLABLE** short => can it contain NULL?
    - procedureNoNulls - does not allow NULL values
    - procedureNullable - allows NULL values
    - procedureNullableUnknown - nullability unknown
13. **REMARKS** String => comment describing parameter/column

Note: Some databases may not return the column descriptions for a procedure. Additional columns beyond REMARKS can be defined by the database.

Parameters:

catalog - a catalog name; "" retrieves those without a catalog; null means drop catalog name from the selection criteria
schemaPattern - a schema name pattern; "" retrieves those without a schema
procedureNamePattern - a procedure name pattern
columnNamePattern - a column name pattern

Returns:

ResultSet - each row is a stored procedure parameter or column description

See Also:

getSearchStringEscape

getTables

```
public abstract ResultSet getTables(String catalog,
                                    String schemaPattern,
                                    String tableNamePattern,
                                    String types[]) throws SQLException
```

Get a description of tables available in a catalog.

Only table descriptions matching the catalog, schema, table name and type criteria are returned. They are ordered by TABLE_TYPE, TABLE_SCHEM and TABLE_NAME.

Each table description has the following columns:

1. **TABLE_CAT** String => table catalog (may be null)
2. **TABLE_SCHEM** String => table schema (may be null)
3. **TABLE_NAME** String => table name
4. **TABLE_TYPE** String => table type. Typical types are "TABLE", "VIEW", "SYSTEM TABLE", "GLOBAL TEMPORARY", "LOCAL TEMPORARY", "ALIAS", "SYNONYM".
5. **REMARKS** String => explanatory comment on the table

Note: Some databases may not return information for all tables.

Parameters:
catalog - a catalog name; "" retrieves those without a catalog; null means drop catalog name from the selection criteria
schemaPattern - a schema name pattern; "" retrieves those without a schema
tableNamePattern - a table name pattern
types - a list of table types to include; null returns all types

Returns:
ResultSet - each row is a table description

See Also:
getSearchStringEscape

- **getSchemas**

```
public abstract ResultSet getSchemas() throws SQLException
```

Get the schema names available in this database. The results are ordered by schema name.

The schema column is:

1. **TABLE_SCHEM** String => schema name

Returns:
ResultSet - each row has a single String column that is a schema name

- **getCatalogs**

```
public abstract ResultSet getCatalogs() throws SQLException
```

Get the catalog names available in this database. The results are ordered by catalog name.

The catalog column is:

1. **TABLE_CAT** String => catalog name

Returns:

ResultSet - each row has a single String column that is a catalog name

- **getTableTypes**

```
public abstract ResultSet getTableTypes() throws SQLException
```

Get the table types available in this database. The results are ordered by table type.

The table type is:

1. **TABLE_TYPE** String => table type. Typical types are "TABLE", "VIEW", "SYSTEM TABLE", "GLOBAL TEMPORARY", "LOCAL TEMPORARY", "ALIAS", "SYNONYM".

Returns:
ResultSet - each row has a single String column that is a table type

- **getColumns**

```
public abstract ResultSet getColumns(String catalog,
                                     String schemaPattern,
                                     String tableNamePattern,
                                     String columnNamePattern) throws SQLException
```

Get a description of table columns available in a catalog.

Only column descriptions matching the catalog, schema, table and column name criteria are returned. They are ordered by TABLE_SCHEM, TABLE_NAME and ORDINAL_POSITION.

Each column description has the following columns:

1. **TABLE_CAT** String => table catalog (may be null)
2. **TABLE_SCHEM** String => table schema (may be null)
3. **TABLE_NAME** String => table name
4. **COLUMN_NAME** String => column name
5. **DATA_TYPE** short => SQL type from java.sql.Types
6. **TYPE_NAME** String => Data source dependent type name
7. **COLUMN_SIZE** int => column size. For char or date types this is the maximum number of characters, for numeric or decimal types this is precision.
8. **BUFFER_LENGTH** is not used.
9. **DECIMAL_DIGITS** int => the number of fractional digits
10. **NUM_PREC_RADIX** int => Radix (typically either 10 or 2)
11. **NULLABLE** int => is NULL allowed?
    - columnNoNulls - might not allow NULL values
    - columnNullable - definitely allows NULL values
    - columnNullableUnknown - nullability unknown
12. **REMARKS** String => comment describing column (may be null)
13. **COLUMN_DEF** String => default value (may be null)
14. **SQL_DATA_TYPE** int => unused
15. **SQL_DATETIME_SUB** int => unused

16. **CHAR_OCTET_LENGTH** int => for char types the maximum number of bytes in the column
17. **ORDINAL_POSITION** int => index of column in table (starting at 1)
18. **IS_NULLABLE** String => "NO" means column definitely does not allow NULL values; "YES" means the column might allow NULL values. An empty string means nobody knows.

Parameters:

catalog - a catalog name; "" retrieves those without a catalog; null means drop catalog name from the selection criteria
schemaPattern - a schema name pattern; "" retrieves those without a schema
tableNamePattern - a table name pattern
columnNamePattern - a column name pattern

Returns:

ResultSet - each row is a column description

See Also:

getSearchStringEscape

- **getColumnPrivileges**

```
public abstract ResultSet getColumnPrivileges(String catalog,
                                              String schema,
                                              String table,
                                              String columnNamePattern) throws SQL
```

Get a description of the access rights for a table's columns.

Only privileges matching the column name criteria are returned. They are ordered by COLUMN_NAME and PRIVILEGE.

Each privilige description has the following columns:

1. **TABLE_CAT** String => table catalog (may be null)
2. **TABLE_SCHEM** String => table schema (may be null)
3. **TABLE_NAME** String => table name
4. **COLUMN_NAME** String => column name
5. **GRANTOR** => grantor of access (may be null)
6. **GRANTEE** String => grantee of access
7. **PRIVILEGE** String => name of access (SELECT, INSERT, UPDATE, REFRENCES, ...)
8. **IS_GRANTABLE** String => "YES" if grantee is permitted to grant to others; "NO" if not; null if unknown

Parameters:

catalog - a catalog name; "" retrieves those without a catalog; null means drop catalog name from the selection criteria
schema - a schema name; "" retrieves those without a schema
table - a table name
columnNamePattern - a column name pattern

Returns:

ResultSet - each row is a column privilege description

See Also:

getSearchStringEscape

- **getTablePrivileges**

```
public abstract ResultSet getTablePrivileges(String catalog,
                                             String schemaPattern,
                                             String tableNamePattern) throws SQLEx
```

Get a description of the access rights for each table available in a catalog. Note that a table privilege applies to one or more columns in the table. It would be wrong to assume that this priviledge applies to all columns (this may be true for some systems but is not true for all.)

Only privileges matching the schema and table name criteria are returned. They are ordered by TABLE_SCHEM, TABLE_NAME, and PRIVILEGE.

Each privilige description has the following columns:

1. **TABLE_CAT** String => table catalog (may be null)
2. **TABLE_SCHEM** String => table schema (may be null)
3. **TABLE_NAME** String => table name
4. **GRANTOR** => grantor of access (may be null)
5. **GRANTEE** String => grantee of access
6. **PRIVILEGE** String => name of access (SELECT, INSERT, UPDATE, REFRENCES, ...)
7. **IS_GRANTABLE** String => "YES" if grantee is permitted to grant to others; "NO" if not; null if unknown

Parameters:

catalog - a catalog name; "" retrieves those without a catalog; null means drop catalog name from the selection criteria
schemaPattern - a schema name pattern; "" retrieves those without a schema
tableNamePattern - a table name pattern

Returns:

ResultSet - each row is a table privilege description

See Also:

getSearchStringEscape

- **getBestRowIdentifier**

```
public abstract ResultSet getBestRowIdentifier(String catalog,
                                               String schema,
                                               String table,
                                               int scope,
                                               boolean nullable) throws SQLExcepti
```

Get a description of a table's optimal set of columns that uniquely identifies a row. They are ordered by SCOPE.

Each column description has the following columns:

1. **SCOPE** short => actual scope of result
   - bestRowTemporary - very temporary, while using row
   - bestRowTransaction - valid for remainder of current transaction

- bestRowSession - valid for remainder of current session

2. **COLUMN_NAME** String => column name
3. **DATA_TYPE** short => SQL data type from java.sql.Types
4. **TYPE_NAME** String => Data source dependent type name
5. **COLUMN_SIZE** int => precision
6. **BUFFER_LENGTH** int => not used
7. **DECIMAL_DIGITS** short => scale
8. **PSEUDO_COLUMN** short => is this a pseudo column like an Oracle ROWID
   - bestRowUnknown - may or may not be pseudo column
   - bestRowNotPseudo - is NOT a pseudo column
   - bestRowPseudo - is a pseudo column

Parameters:

catalog - a catalog name; "" retrieves those without a catalog; null means drop catalog name from the selection criteria
schema - a schema name; "" retrieves those without a schema
table - a table name
scope - the scope of interest; use same values as SCOPE
nullable - include columns that are nullable?

Returns:

ResultSet - each row is a column description

getVersionColumns

```
public abstract ResultSet getVersionColumns(String catalog,
                                            String schema,
                                            String table) throws SQLException
```

Get a description of a table's columns that are automatically updated when any value in a row is updated. They are unordered.

Each column description has the following columns:

1. **SCOPE** short => is not used
2. **COLUMN_NAME** String => column name
3. **DATA_TYPE** short => SQL data type from java.sql.Types
4. **TYPE_NAME** String => Data source dependent type name
5. **COLUMN_SIZE** int => precision
6. **BUFFER_LENGTH** int => length of column value in bytes
7. **DECIMAL_DIGITS** short => scale
8. **PSEUDO_COLUMN** short => is this a pseudo column like an Oracle ROWID
   - versionColumnUnknown - may or may not be pseudo column
   - versionColumnNotPseudo - is NOT a pseudo column
   - versionColumnPseudo - is a pseudo column

Parameters:

catalog - a catalog name; "" retrieves those without a catalog; null means drop catalog name from the selection criteria
schema - a schema name; "" retrieves those without a schema
table - a table name

Returns:

ResultSet - each row is a column description

- **getPrimaryKeys**

```
public abstract ResultSet getPrimaryKeys(String catalog,
                                         String schema,
                                         String table) throws SQLException
```

Get a description of a table's primary key columns. They are ordered by COLUMN_NAME.

Each primary key column description has the following columns:

1. **TABLE_CAT** String => table catalog (may be null)
2. **TABLE_SCHEM** String => table schema (may be null)
3. **TABLE_NAME** String => table name
4. **COLUMN_NAME** String => column name
5. **KEY_SEQ** short => sequence number within primary key
6. **PK_NAME** String => primary key name (may be null)

Parameters:

catalog - a catalog name; "" retrieves those without a catalog; null means drop catalog name from the selection criteria
schema - a schema name pattern; "" retrieves those without a schema
table - a table name

Returns:

ResultSet - each row is a primary key column description

- **getImportedKeys**

```
public abstract ResultSet getImportedKeys(String catalog,
                                          String schema,
                                          String table) throws SQLException
```

Get a description of the primary key columns that are referenced by a table's foreign key columns (the primary keys imported by a table). They are ordered by PKTABLE_CAT, PKTABLE_SCHEM, PKTABLE_NAME, and KEY_SEQ.

Each primary key column description has the following columns:

1. **PKTABLE_CAT** String => primary key table catalog being imported (may be null)
2. **PKTABLE_SCHEM** String => primary key table schema being imported (may be null)
3. **PKTABLE_NAME** String => primary key table name being imported
4. **PKCOLUMN_NAME** String => primary key column name being imported
5. **FKTABLE_CAT** String => foreign key table catalog (may be null)
6. **FKTABLE_SCHEM** String => foreign key table schema (may be null)
7. **FKTABLE_NAME** String => foreign key table name
8. **FKCOLUMN_NAME** String => foreign key column name
9. **KEY_SEQ** short => sequence number within foreign key
10. **UPDATE_RULE** short => What happens to foreign key when primary is updated:

- importedNoAction - do not allow update of primary key if it has been imported
- importedKeyCascade - change imported key to agree with primary key update
- importedKeySetNull - change imported key to NULL if its primary key has been updated
- importedKeySetDefault - change imported key to default values if its primary key has been updated
- importedKeyRestrict - same as importedKeyNoAction (for ODBC 2.x compatibility)

11. **DELETE_RULE** short => What happens to the foreign key when primary is deleted.
    - importedKeyNoAction - do not allow delete of primary key if it has been imported
    - importedKeyCascade - delete rows that import a deleted key
    - importedKeySetNull - change imported key to NULL if its primary key has been deleted
    - importedKeyRestrict - same as importedKeyNoAction (for ODBC 2.x compatibility)
    - importedKeySetDefault - change imported key to default if its primary key has been deleted
12. **FK_NAME** String => foreign key name (may be null)
13. **PK_NAME** String => primary key name (may be null)
14. **DEFERRABILITY** short => can the evaluation of foreign key constraints be deferred until commit
    - importedKeyInitiallyDeferred - see SQL92 for definition
    - importedKeyInitiallyImmediate - see SQL92 for definition
    - importedKeyNotDeferrable - see SQL92 for definition

Parameters:

catalog - a catalog name; "" retrieves those without a catalog; null means drop catalog name from the selection criteria
schema - a schema name pattern; "" retrieves those without a schema
table - a table name

Returns:

ResultSet - each row is a primary key column description

See Also:

getExportedKeys

getExportedKeys

```
public abstract ResultSet getExportedKeys(String catalog,
                                          String schema,
                                          String table) throws SQLException
```

Get a description of the foreign key columns that reference a table's primary key columns (the foreign keys exported by a table). They are ordered by FKTABLE_CAT, FKTABLE_SCHEM, FKTABLE_NAME, and KEY_SEQ.

Each foreign key column description has the following columns:

1. **PKTABLE_CAT** String => primary key table catalog (may be null)
2. **PKTABLE_SCHEM** String => primary key table schema (may be null)
3. **PKTABLE_NAME** String => primary key table name
4. **PKCOLUMN_NAME** String => primary key column name
5. **FKTABLE_CAT** String => foreign key table catalog (may be null) being exported (may be

null)

6. **FKTABLE_SCHEM** String => foreign key table schema (may be null) being exported (may be null)
7. **FKTABLE_NAME** String => foreign key table name being exported
8. **FKCOLUMN_NAME** String => foreign key column name being exported
9. **KEY_SEQ** short => sequence number within foreign key
10. **UPDATE_RULE** short => What happens to foreign key when primary is updated:
    - importedNoAction - do not allow update of primary key if it has been imported
    - importedKeyCascade - change imported key to agree with primary key update
    - importedKeySetNull - change imported key to NULL if its primary key has been updated
    - importedKeySetDefault - change imported key to default values if its primary key has been updated
    - importedKeyRestrict - same as importedKeyNoAction (for ODBC 2.x compatibility)
11. **DELETE_RULE** short => What happens to the foreign key when primary is deleted.
    - importedKeyNoAction - do not allow delete of primary key if it has been imported
    - importedKeyCascade - delete rows that import a deleted key
    - importedKeySetNull - change imported key to NULL if its primary key has been deleted
    - importedKeyRestrict - same as importedKeyNoAction (for ODBC 2.x compatibility)
    - importedKeySetDefault - change imported key to default if its primary key has been deleted
12. **FK_NAME** String => foreign key name (may be null)
13. **PK_NAME** String => primary key name (may be null)
14. **DEFERRABILITY** short => can the evaluation of foreign key constraints be deferred until commit
    - importedKeyInitiallyDeferred - see SQL92 for definition
    - importedKeyInitiallyImmediate - see SQL92 for definition
    - importedKeyNotDeferrable - see SQL92 for definition

Parameters:

catalog - a catalog name; "" retrieves those without a catalog; null means drop catalog name from the selection criteria
schema - a schema name pattern; "" retrieves those without a schema
table - a table name

Returns:

ResultSet - each row is a foreign key column description

See Also:

getImportedKeys

getCrossReference

```
public abstract ResultSet getCrossReference(String primaryCatalog,
                                            String primarySchema,
                                            String primaryTable,
                                            String foreignCatalog,
                                            String foreignSchema,
                                            String foreignTable) throws SQLExcepti
```

Get a description of the foreign key columns in the foreign key table that reference the primary

key columns of the primary key table (describe how one table imports another's key.) This should normally return a single foreign key/primary key pair (most tables only import a foreign key from a table once.) They are ordered by FKTABLE_CAT, FKTABLE_SCHEM, FKTABLE_NAME, and KEY_SEQ.

Each foreign key column description has the following columns:

1. **PKTABLE_CAT** String => primary key table catalog (may be null)
2. **PKTABLE_SCHEM** String => primary key table schema (may be null)
3. **PKTABLE_NAME** String => primary key table name
4. **PKCOLUMN_NAME** String => primary key column name
5. **FKTABLE_CAT** String => foreign key table catalog (may be null) being exported (may be null)
6. **FKTABLE_SCHEM** String => foreign key table schema (may be null) being exported (may be null)
7. **FKTABLE_NAME** String => foreign key table name being exported
8. **FKCOLUMN_NAME** String => foreign key column name being exported
9. **KEY_SEQ** short => sequence number within foreign key
10. **UPDATE_RULE** short => What happens to foreign key when primary is updated:
    - importedNoAction - do not allow update of primary key if it has been imported
    - importedKeyCascade - change imported key to agree with primary key update
    - importedKeySetNull - change imported key to NULL if its primary key has been updated
    - importedKeySetDefault - change imported key to default values if its primary key has been updated
    - importedKeyRestrict - same as importedKeyNoAction (for ODBC 2.x compatibility)
11. **DELETE_RULE** short => What happens to the foreign key when primary is deleted.
    - importedKeyNoAction - do not allow delete of primary key if it has been imported
    - importedKeyCascade - delete rows that import a deleted key
    - importedKeySetNull - change imported key to NULL if its primary key has been deleted
    - importedKeyRestrict - same as importedKeyNoAction (for ODBC 2.x compatibility)
    - importedKeySetDefault - change imported key to default if its primary key has been deleted
12. **FK_NAME** String => foreign key name (may be null)
13. **PK_NAME** String => primary key name (may be null)
14. **DEFERRABILITY** short => can the evaluation of foreign key constraints be deferred until commit
    - importedKeyInitiallyDeferred - see SQL92 for definition
    - importedKeyInitiallyImmediate - see SQL92 for definition
    - importedKeyNotDeferrable - see SQL92 for definition

Parameters:

primaryCatalog - a catalog name; "" retrieves those without a catalog; null means drop catalog name from the selection criteria
primarySchema - a schema name pattern; "" retrieves those without a schema
primaryTable - the table name that exports the key
foreignCatalog - a catalog name; "" retrieves those without a catalog; null means drop catalog name from the selection criteria

foreignSchema - a schema name pattern; "" retrieves those without a schema
foreignTable - the table name that imports the key

Returns:

ResultSet - each row is a foreign key column description

See Also:

getImportedKeys

- **getTypeInfo**

```
public abstract ResultSet getTypeInfo() throws SQLException
```

Get a description of all the standard SQL types supported by this database. They are ordered by DATA_TYPE and then by how closely the data type maps to the corresponding JDBC SQL type.

Each type description has the following columns:

1. **TYPE_NAME** String => Type name
2. **DATA_TYPE** short => SQL data type from java.sql.Types
3. **PRECISION** int => maximum precision
4. **LITERAL_PREFIX** String => prefix used to quote a literal (may be null)
5. **LITERAL_SUFFIX** String => suffix used to quote a literal (may be null)
6. **CREATE_PARAMS** String => parameters used in creating the type (may be null)
7. **NULLABLE** short => can you use NULL for this type?
   - typeNoNulls - does not allow NULL values
   - typeNullable - allows NULL values
   - typeNullableUnknown - nullability unknown
8. **CASE_SENSITIVE** boolean=> is it case sensitive?
9. **SEARCHABLE** short => can you use "WHERE" based on this type:
   - typePredNone - No support
   - typePredChar - Only supported with WHERE .. LIKE
   - typePredBasic - Supported except for WHERE .. LIKE
   - typeSearchable - Supported for all WHERE ..
10. **UNSIGNED_ATTRIBUTE** boolean => is it unsigned?
11. **FIXED_PREC_SCALE** boolean => can it be a money value?
12. **AUTO_INCREMENT** boolean => can it be used for an auto-increment value?
13. **LOCAL_TYPE_NAME** String => localized version of type name (may be null)
14. **MINIMUM_SCALE** short => minimum scale supported
15. **MAXIMUM_SCALE** short => maximum scale supported
16. **SQL_DATA_TYPE** int => unused
17. **SQL_DATETIME_SUB** int => unused
18. **NUM_PREC_RADIX** int => usually 2 or 10

Returns:

ResultSet - each row is a SQL type description

- **getIndexInfo**

```
public abstract ResultSet getIndexInfo(String catalog,
                                       String schema,
```

```
                                    String table,
                                    boolean unique,
                                    boolean approximate) throws SQLException
```

Get a description of a table's indices and statistics. They are ordered by NON_UNIQUE, TYPE, INDEX_NAME, and ORDINAL_POSITION.

Each index column description has the following columns:

1. **TABLE_CAT** String => table catalog (may be null)
2. **TABLE_SCHEM** String => table schema (may be null)
3. **TABLE_NAME** String => table name
4. **NON_UNIQUE** boolean => Can index values be non-unique? false when TYPE is tableIndexStatistic
5. **INDEX_QUALIFIER** String => index catalog (may be null); null when TYPE is tableIndexStatistic
6. **INDEX_NAME** String => index name; null when TYPE is tableIndexStatistic
7. **TYPE** short => index type:
   - tableIndexStatistic - this identifies table statistics that are returned in conjuction with a table's index descriptions
   - tableIndexClustered - this is a clustered index
   - tableIndexHashed - this is a hashed index
   - tableIndexOther - this is some other style of index
8. **ORDINAL_POSITION** short => column sequence number within index; zero when TYPE is tableIndexStatistic
9. **COLUMN_NAME** String => column name; null when TYPE is tableIndexStatistic
10. **ASC_OR_DESC** String => column sort sequence, "A" => ascending, "D" => descending, may be null if sort sequence is not supported; null when TYPE is tableIndexStatistic
11. **CARDINALITY** int => When TYPE is tableIndexStatistic, then this is the number of rows in the table; otherwise, it is the number of unique values in the index.
12. **PAGES** int => When TYPE is tableIndexStatisic then this is the number of pages used for the table, otherwise it is the number of pages used for the current index.
13. **FILTER_CONDITION** String => Filter condition, if any. (may be null)

Parameters:

catalog - a catalog name; "" retrieves those without a catalog; null means drop catalog name from the selection criteria
schema - a schema name pattern; "" retrieves those without a schema
table - a table name
unique - when true, return only indices for unique values; when false, return indices regardless of whether unique or not
approximate - when true, result is allowed to reflect approximate or out of data values; when false, results are requested to be accurate

Returns:

ResultSet - each row is an index column description

Interface java.sql.Driver

public interface **Driver**
extends Object

The Java SQL framework allows for multiple database drivers.

Each driver should supply a class that implements the Driver interface.

The DriverManager will try to load as many drivers as it can find and then for any given connection request, it will ask each driver in turn to try to connect to the target URL.

It is strongly recommended that each Driver class should be small and standalone so that the Driver class can be loaded and queried without bringing in vast quantities of supporting code.

When a Driver class is loaded, it should create an instance of itself and register it with the DriverManager. This means that a user can load and register a driver by doing Class.forName("foo.bah.Driver").

See Also:
DriverManager, Connection

Method Index

Methods

- **connect**

```
public abstract Connection connect(String url,
                                   Properties info) throws SQLException
```

Try to make a database connection to the given URL. The driver should return "null" if it realizes it is the wrong kind of driver to connect to the given URL. This will be common, as when the JDBC driver manager is asked to connect to a given URL it passes the URL to each loaded driver in turn.

The driver should raise a SQLException if it is the right driver to connect to the given URL, but has trouble connecting to the database.

The java.util.Properties argument can be used to passed arbitrary string tag/value pairs as connection arguments. Normally at least "user" and "password" properties should be included in the Properties.

Parameters:

url - The URL of the database to connect to
info - a list of arbitrary string tag/value pairs as connection arguments; normally at least a "user" and "password" property should be included

Returns:

a Connection to the URL

- **acceptsURL**

```
public abstract boolean acceptsURL(String url) throws SQLException
```

Returns true if the driver thinks that it can open a connection to the given URL. Typically drivers will return true if they understand the subprotocol specified in the URL and false if they don't.

Parameters:

url - The URL of the database.

Returns:

True if this driver can connect to the given URL.

- **getPropertyInfo**

```
public abstract DriverPropertyInfo[] getPropertyInfo(String url,
                                                     Properties info) throws SQLEx
```

The getPropertyInfo method is intended to allow a generic GUI tool to discover what properties it should prompt a human for in order to get enough information to connect to a database. Note that depending on the values the human has supplied so far, additional values may become necessary, so it may be necessary to iterate though several calls to getPropertyInfo.

Parameters:

url - The URL of the database to connect to.
info - A proposed list of tag/value pairs that will be sent on connect open.

Returns:

An array of DriverPropertyInfo objects describing possible properties. This array may be an empty array if no properties are required.

- **getMajorVersion**

```
public abstract int getMajorVersion()
```

Get the driver's major version number. Initially this should be 1.

- **getMinorVersion**

```
public abstract int getMinorVersion()
```

Get the driver's minor version number. Initially this should be 0.

- **jdbcCompliant**

```
public abstract boolean jdbcCompliant()
```

Report whether the Driver is a genuine JDBC COMPLIANT (tm) driver. A driver may only report "true" here if it passes the JDBC compliance tests, otherwise it is required to return false. JDBC compliance requires full support for the JDBC API and full support for SQL 92 Entry Level. It is expected that JDBC compliant drivers will be available for all the major commercial databases. This method is not intended to encourage the development of non-JDBC compliant drivers, but is a recognition of the fact that some vendors are interested in using the JDBC API and framework for lightweight databases that do not support full database functionality, or for special databases such as document information retrieval where a SQL implementation may not be feasible.

Interface java.sql.PreparedStatement

public interface **PreparedStatement**
extends Object
extends Statement

A SQL statement is pre-compiled and stored in a PreparedStatement object. This object can then be used to efficiently execute this statement multiple times.

Note: The setXXX methods for setting IN parameter values must specify types that are compatible with the defined SQL type of the input parameter. For instance, if the IN parameter has SQL type Integer then setInt should be used.

If arbitrary parameter type conversions are required then the setObject method should be used with a target SQL type.

See Also:
prepareStatement, ResultSet

Method Index

- **clearParameters**()
  In general, parameter values remain in force for repeated use of a Statement.
- **execute**()
  Some prepared statements return multiple results; the execute method handles these complex statements as well as the simpler form of statements handled by executeQuery and executeUpdate.
- **executeQuery**()
  A prepared SQL query is executed and its ResultSet is returned.
- **executeUpdate**()
  Execute a SQL INSERT, UPDATE or DELETE statement.
- **setAsciiStream**(int, InputStream, int)
  When a very large ASCII value is input to a LONGVARCHAR parameter, it may be more practical to send it via a java.io.InputStream.
- **setBigDecimal**(int, BigDecimal)
  Set a parameter to a java.lang.BigDecimal value.
- **setBinaryStream**(int, InputStream, int)
  When a very large binary value is input to a LONGVARBINARY parameter, it may be more practical to send it via a java.io.InputStream.
- **setBoolean**(int, boolean)
  Set a parameter to a Java boolean value.
- **setByte**(int, byte)
  Set a parameter to a Java byte value.
- **setBytes**(int, byte[])
  Set a parameter to a Java array of bytes.
- **setDate**(int, Date)
  Set a parameter to a java.sql.Date value.
- **setDouble**(int, double)
  Set a parameter to a Java double value.
- **setFloat**(int, float)
  Set a parameter to a Java float value.
- **setInt**(int, int)
  Set a parameter to a Java int value.
- **setLong**(int, long)
  Set a parameter to a Java long value.
- **setNull**(int, int)
  Set a parameter to SQL NULL.
- **setObject**(int, Object)
  Set the value of a parameter using an object; use the java.lang equivalent objects for integral values.
- **setObject**(int, Object, int)
  This method is like setObject above, but assumes a scale of zero.
- **setObject**(int, Object, int, int)
  Set the value of a parameter using an object; use the java.lang equivalent objects for integral values.
- **setShort**(int, short)

Set a parameter to a Java short value.

- **setString**(int, String)
  Set a parameter to a Java String value.
- **setTime**(int, Time)
  Set a parameter to a java.sql.Time value.
- **setTimestamp**(int, Timestamp)
  Set a parameter to a java.sql.Timestamp value.
- **setUnicodeStream**(int, InputStream, int)
  When a very large UNICODE value is input to a LONGVARCHAR parameter, it may be more practical to send it via a java.io.InputStream.

- **executeQuery**

```
public abstract ResultSet executeQuery() throws SQLException
```

  A prepared SQL query is executed and its ResultSet is returned.
  Returns:
    a ResultSet that contains the data produced by the query; never null

- **executeUpdate**

```
public abstract int executeUpdate() throws SQLException
```

  Execute a SQL INSERT, UPDATE or DELETE statement. In addition, SQL statements that return nothing such as SQL DDL statements can be executed.
  Returns:
    either the row count for INSERT, UPDATE or DELETE; or 0 for SQL statements that return nothing

- **setNull**

```
public abstract void setNull(int parameterIndex,
                             int sqlType) throws SQLException
```

  Set a parameter to SQL NULL.

  Note: You must specify the parameter's SQL type.

  Parameters:
    parameterIndex - the first parameter is 1, the second is 2, ...
    sqlType - SQL type code defined by java.sql.Types

- **setBoolean**

```
public abstract void setBoolean(int parameterIndex,
                                boolean x) throws SQLException
```

Set a parameter to a Java boolean value. The driver converts this to a SQL BIT value when it sends it to the database.

Parameters:

parameterIndex - the first parameter is 1, the second is 2, ...
x - the parameter value

- **setByte**

```
public abstract void setByte(int parameterIndex,
                             byte x) throws SQLException
```

Set a parameter to a Java byte value. The driver converts this to a SQL TINYINT value when it sends it to the database.

Parameters:

parameterIndex - the first parameter is 1, the second is 2, ...
x - the parameter value

- **setShort**

```
public abstract void setShort(int parameterIndex,
                              short x) throws SQLException
```

Set a parameter to a Java short value. The driver converts this to a SQL SMALLINT value when it sends it to the database.

Parameters:

parameterIndex - the first parameter is 1, the second is 2, ...
x - the parameter value

- **setInt**

```
public abstract void setInt(int parameterIndex,
                            int x) throws SQLException
```

Set a parameter to a Java int value. The driver converts this to a SQL INTEGER value when it sends it to the database.

Parameters:

parameterIndex - the first parameter is 1, the second is 2, ...
x - the parameter value

- **setLong**

```
public abstract void setLong(int parameterIndex,
                             long x) throws SQLException
```

Set a parameter to a Java long value. The driver converts this to a SQL BIGINT value when it sends it to the database.

Parameters:

parameterIndex - the first parameter is 1, the second is 2, ...
x - the parameter value

- **setFloat**

```
public abstract void setFloat(int parameterIndex,
                              float x) throws SQLException
```

Set a parameter to a Java float value. The driver converts this to a SQL FLOAT value when it sends it to the database.

Parameters:

parameterIndex - the first parameter is 1, the second is 2, ...
x - the parameter value

- **setDouble**

```
public abstract void setDouble(int parameterIndex,
                               double x) throws SQLException
```

Set a parameter to a Java double value. The driver converts this to a SQL DOUBLE value when it sends it to the database.

Parameters:

parameterIndex - the first parameter is 1, the second is 2, ...
x - the parameter value

- **setBigDecimal**

```
public abstract void setBigDecimal(int parameterIndex,
                                   BigDecimal x) throws SQLException
```

Set a parameter to a java.lang.BigDecimal value. The driver converts this to a SQL NUMERIC value when it sends it to the database.

Parameters:

parameterIndex - the first parameter is 1, the second is 2, ...
x - the parameter value

- **setString**

```
public abstract void setString(int parameterIndex,
                               String x) throws SQLException
```

Set a parameter to a Java String value. The driver converts this to a SQL VARCHAR or LONGVARCHAR value (depending on the arguments size relative to the driver's limits on VARCHARs) when it sends it to the database.

Parameters:

parameterIndex - the first parameter is 1, the second is 2, ...
x - the parameter value

- **setBytes**

```
public abstract void setBytes(int parameterIndex,
                              byte x[]) throws SQLException
```

Set a parameter to a Java array of bytes. The driver converts this to a SQL VARBINARY or

LONGVARBINARY (depending on the argument's size relative to the driver's limits on VARBINARYs) when it sends it to the database.

Parameters:

parameterIndex - the first parameter is 1, the second is 2, ...
x - the parameter value

- **setDate**

```
public abstract void setDate(int parameterIndex,
                             Date x) throws SQLException
```

Set a parameter to a java.sql.Date value. The driver converts this to a SQL DATE value when it sends it to the database.

Parameters:

parameterIndex - the first parameter is 1, the second is 2, ...
x - the parameter value

- **setTime**

```
public abstract void setTime(int parameterIndex,
                             Time x) throws SQLException
```

Set a parameter to a java.sql.Time value. The driver converts this to a SQL TIME value when it sends it to the database.

Parameters:

parameterIndex - the first parameter is 1, the second is 2, ...
x - the parameter value

- **setTimestamp**

```
public abstract void setTimestamp(int parameterIndex,
                                  Timestamp x) throws SQLException
```

Set a parameter to a java.sql.Timestamp value. The driver converts this to a SQL TIMESTAMP value when it sends it to the database.

Parameters:

parameterIndex - the first parameter is 1, the second is 2, ...
x - the parameter value

- **setAsciiStream**

```
public abstract void setAsciiStream(int parameterIndex,
                                    InputStream x,
                                    int length) throws SQLException
```

When a very large ASCII value is input to a LONGVARCHAR parameter, it may be more practical to send it via a java.io.InputStream. JDBC will read the data from the stream as needed, until it reaches end-of-file. The JDBC driver will do any necessary conversion from ASCII to the database char format.

Note: This stream object can either be a standard Java stream object or your own subclass that

implements the standard interface.

Parameters:

parameterIndex - the first parameter is 1, the second is 2, ...
x - the java input stream which contains the ASCII parameter value
length - the number of bytes in the stream

- **setUnicodeStream**

```
public abstract void setUnicodeStream(int parameterIndex,
                                      InputStream x,
                                      int length) throws SQLException
```

When a very large UNICODE value is input to a LONGVARCHAR parameter, it may be more practical to send it via a java.io.InputStream. JDBC will read the data from the stream as needed, until it reaches end-of-file. The JDBC driver will do any necessary conversion from UNICODE to the database char format.

Note: This stream object can either be a standard Java stream object or your own subclass that implements the standard interface.

Parameters:

parameterIndex - the first parameter is 1, the second is 2, ...
x - the java input stream which contains the UNICODE parameter value
length - the number of bytes in the stream

- **setBinaryStream**

```
public abstract void setBinaryStream(int parameterIndex,
                                     InputStream x,
                                     int length) throws SQLException
```

When a very large binary value is input to a LONGVARBINARY parameter, it may be more practical to send it via a java.io.InputStream. JDBC will read the data from the stream as needed, until it reaches end-of-file.

Note: This stream object can either be a standard Java stream object or your own subclass that implements the standard interface.

Parameters:

parameterIndex - the first parameter is 1, the second is 2, ...
x - the java input stream which contains the binary parameter value
length - the number of bytes in the stream

- **clearParameters**

```
public abstract void clearParameters() throws SQLException
```

In general, parameter values remain in force for repeated use of a Statement. Setting a parameter value automatically clears its previous value. However, in some cases it is useful to immediately

release the resources used by the current parameter values; this can be done by calling clearParameters.

- **setObject**

```
public abstract void setObject(int parameterIndex,
                               Object x,
                               int targetSqlType,
                               int scale) throws SQLException
```

Set the value of a parameter using an object; use the java.lang equivalent objects for integral values.

The given Java object will be converted to the targetSqlType before being sent to the database.

Note that this method may be used to pass datatabase- specific abstract data types. This is done by using a Driver- specific Java type and using a targetSqlType of java.sql.types.OTHER.

Parameters:

parameterIndex - The first parameter is 1, the second is 2, ...
x - The object containing the input parameter value
targetSqlType - The SQL type (as defined in java.sql.Types) to be sent to the database. The scale argument may further qualify this type.
scale - For java.sql.Types.DECIMAL or java.sql.Types.NUMERIC types this is the number of digits after the decimal. For all other types this value will be ignored,

See Also:

Types

- **setObject**

```
public abstract void setObject(int parameterIndex,
                               Object x,
                               int targetSqlType) throws SQLException
```

This method is like setObject above, but assumes a scale of zero.

- **setObject**

```
public abstract void setObject(int parameterIndex,
                               Object x) throws SQLException
```

Set the value of a parameter using an object; use the java.lang equivalent objects for integral values.

The JDBC specification specifies a standard mapping from Java Object types to SQL types. The given argument java object will be converted to the corresponding SQL type before being sent to the database.

Note that this method may be used to pass datatabase specific abstract data types, by using a Driver specific Java type.

Parameters:

parameterIndex - The first parameter is 1, the second is 2, ...
x - The object containing the input parameter value

execute

```
public abstract boolean execute() throws SQLException
```

Some prepared statements return multiple results; the execute method handles these complex statements as well as the simpler form of statements handled by executeQuery and executeUpdate.

See Also:

execute

Interface java.sql.ResultSet

public interface **ResultSet**
extends Object

A ResultSet provides access to a table of data generated by executing a Statement. The table rows are retrieved in sequence. Within a row its column values can be accessed in any order.

A ResultSet maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The 'next' method moves the cursor to the next row.

The getXXX methods retrieve column values for the current row. You can retrieve values either using the index number of the column, or by using the name of the column. In general using the column index will be more efficient. Columns are numbered from 1.

For maximum portability, ResultSet columns within each row should be read in left-to-right order and each column should be read only once.

For the getXXX methods, the JDBC driver attempts to convert the underlying data to the specified Java type and returns a suitable Java value. See the JDBC specification for allowable mappings from SQL types to Java types with the ResultSet.getXXX methods.

Column names used as input to getXXX methods are case insensitive. When performing a getXXX using a column name, if several columns have the same name, then the value of the first matching column will be returned. The column name option is designed to be used when column names are used in the SQL query. For columns that are NOT explicitly named in the query, it is best to use column numbers. If column names were used there is no way for the programmer to guarantee that they actually refer to the intended columns.

A ResultSet is automatically closed by the Statement that generated it when that Statement is closed, re-executed, or is used to retrieve the next result from a sequence of multiple results.

The number, types and properties of a ResultSet's columns are provided by the ResulSetMetaData object returned by the getMetaData method.

See Also:

executeQuery, getResultSet, ResultSetMetaData

Method Index

- **clearWarnings**()
  After this call getWarnings returns null until a new warning is reported for this ResultSet.
- **close**()
  In some cases, it is desirable to immediately release a ResultSet's database and JDBC resources instead of waiting for this to happen when it is automatically closed; the close method provides this immediate release.
- **findColumn**(String)
  Map a Resultset column name to a ResultSet column index.
- **getAsciiStream**(int)
  A column value can be retrieved as a stream of ASCII characters and then read in chunks from the stream.
- **getAsciiStream**(String)
  A column value can be retrieved as a stream of ASCII characters and then read in chunks from the stream.
- **getBigDecimal**(int, int)
  Get the value of a column in the current row as a java.lang.BigDecimal object.
- **getBigDecimal**(String, int)
  Get the value of a column in the current row as a java.lang.BigDecimal object.
- **getBinaryStream**(int)
  A column value can be retrieved as a stream of uninterpreted bytes and then read in chunks from the stream.
- **getBinaryStream**(String)
  A column value can be retrieved as a stream of uninterpreted bytes and then read in chunks from the stream.
- **getBoolean**(int)
  Get the value of a column in the current row as a Java boolean.
- **getBoolean**(String)
  Get the value of a column in the current row as a Java boolean.
- **getByte**(int)
  Get the value of a column in the current row as a Java byte.
- **getByte**(String)
  Get the value of a column in the current row as a Java byte.
- **getBytes**(int)
  Get the value of a column in the current row as a Java byte array.
- **getBytes**(String)
  Get the value of a column in the current row as a Java byte array.

- **getCursorName**()
  Get the name of the SQL cursor used by this ResultSet.
- **getDate**(int)
  Get the value of a column in the current row as a java.sql.Date object.
- **getDate**(String)
  Get the value of a column in the current row as a java.sql.Date object.
- **getDouble**(int)
  Get the value of a column in the current row as a Java double.
- **getDouble**(String)
  Get the value of a column in the current row as a Java double.
- **getFloat**(int)
  Get the value of a column in the current row as a Java float.
- **getFloat**(String)
  Get the value of a column in the current row as a Java float.
- **getInt**(int)
  Get the value of a column in the current row as a Java int.
- **getInt**(String)
  Get the value of a column in the current row as a Java int.
- **getLong**(int)
  Get the value of a column in the current row as a Java long.
- **getLong**(String)
  Get the value of a column in the current row as a Java long.
- **getMetaData**()
  The number, types and properties of a ResultSet's columns are provided by the getMetaData method.
- **getObject**(int)
  Get the value of a column in the current row as a Java object.
- **getObject**(String)
  Get the value of a column in the current row as a Java object.
- **getShort**(int)
  Get the value of a column in the current row as a Java short.
- **getShort**(String)
  Get the value of a column in the current row as a Java short.
- **getString**(int)
  Get the value of a column in the current row as a Java String.
- **getString**(String)
  Get the value of a column in the current row as a Java String.
- **getTime**(int)
  Get the value of a column in the current row as a java.sql.Time object.
- **getTime**(String)
  Get the value of a column in the current row as a java.sql.Time object.
- **getTimestamp**(int)
  Get the value of a column in the current row as a java.sql.Timestamp object.
- **getTimestamp**(String)
  Get the value of a column in the current row as a java.sql.Timestamp object.
- **getUnicodeStream**(int)
  A column value can be retrieved as a stream of Unicode characters and then read in chunks from

the stream.

- **getUnicodeStream**(String)
  A column value can be retrieved as a stream of Unicode characters and then read in chunks from the stream.
- **getWarnings**()
  The first warning reported by calls on this ResultSet is returned.
- **next**()
  A ResultSet is initially positioned before its first row; the first call to next makes the first row the current row; the second call makes the second row the current row, etc.
- **wasNull**()
  A column may have the value of SQL NULL; wasNull reports whether the last column read had this special value.

- **next**

```
public abstract boolean next() throws SQLException
```

  A ResultSet is initially positioned before its first row; the first call to next makes the first row the current row; the second call makes the second row the current row, etc.

  If an input stream from the previous row is open, it is implicitly closed. The ResultSet's warning chain is cleared when a new row is read.

  Returns:
  true if the new current row is valid; false if there are no more rows

- **close**

```
public abstract void close() throws SQLException
```

  In some cases, it is desirable to immediately release a ResultSet's database and JDBC resources instead of waiting for this to happen when it is automatically closed; the close method provides this immediate release.

  Note: A ResultSet is automatically closed by the Statement that generated it when that Statement is closed, re-executed, or is used to retrieve the next result from a sequence of multiple results. A ResultSet is also automatically closed when it is garbage collected.

- **wasNull**

```
public abstract boolean wasNull() throws SQLException
```

  A column may have the value of SQL NULL; wasNull reports whether the last column read had this special value. Note that you must first call getXXX on a column to try to read its value and then call wasNull() to find if the value was the SQL NULL.
  Returns:

true if last column read was SQL NULL

- **getString**

```
public abstract String getString(int columnIndex) throws SQLException
```

Get the value of a column in the current row as a Java String.
Parameters:
columnIndex - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL NULL, the result is null

- **getBoolean**

```
public abstract boolean getBoolean(int columnIndex) throws SQLException
```

Get the value of a column in the current row as a Java boolean.
Parameters:
columnIndex - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL NULL, the result is false

- **getByte**

```
public abstract byte getByte(int columnIndex) throws SQLException
```

Get the value of a column in the current row as a Java byte.
Parameters:
columnIndex - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL NULL, the result is 0

- **getShort**

```
public abstract short getShort(int columnIndex) throws SQLException
```

Get the value of a column in the current row as a Java short.
Parameters:
columnIndex - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL NULL, the result is 0

- **getInt**

```
public abstract int getInt(int columnIndex) throws SQLException
```

Get the value of a column in the current row as a Java int.
Parameters:
columnIndex - the first column is 1, the second is 2, ...
Returns:

the column value; if the value is SQL NULL, the result is 0

- **getLong**

```
public abstract long getLong(int columnIndex) throws SQLException
```

Get the value of a column in the current row as a Java long.
Parameters:
columnIndex - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL NULL, the result is 0

- **getFloat**

```
public abstract float getFloat(int columnIndex) throws SQLException
```

Get the value of a column in the current row as a Java float.
Parameters:
columnIndex - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL NULL, the result is 0

- **getDouble**

```
public abstract double getDouble(int columnIndex) throws SQLException
```

Get the value of a column in the current row as a Java double.
Parameters:
columnIndex - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL NULL, the result is 0

- **getBigDecimal**

```
public abstract BigDecimal getBigDecimal(int columnIndex,
                                         int scale) throws SQLException
```

Get the value of a column in the current row as a java.lang.BigDecimal object.
Parameters:
columnIndex - the first column is 1, the second is 2, ...
scale - the number of digits to the right of the decimal
Returns:
the column value; if the value is SQL NULL, the result is null

- **getBytes**

```
public abstract byte[] getBytes(int columnIndex) throws SQLException
```

Get the value of a column in the current row as a Java byte array. The bytes represent the raw values returned by the driver.

Parameters:
columnIndex - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL NULL, the result is null

- **getDate**

```
public abstract Date getDate(int columnIndex) throws SQLException
```

Get the value of a column in the current row as a java.sql.Date object.
Parameters:
columnIndex - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL NULL, the result is null

- **getTime**

```
public abstract Time getTime(int columnIndex) throws SQLException
```

Get the value of a column in the current row as a java.sql.Time object.
Parameters:
columnIndex - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL NULL, the result is null

- **getTimestamp**

```
public abstract Timestamp getTimestamp(int columnIndex) throws SQLException
```

Get the value of a column in the current row as a java.sql.Timestamp object.
Parameters:
columnIndex - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL NULL, the result is null

- **getAsciiStream**

```
public abstract InputStream getAsciiStream(int columnIndex) throws SQLException
```

A column value can be retrieved as a stream of ASCII characters and then read in chunks from the stream. This method is particularly suitable for retrieving large LONGVARCHAR values. The JDBC driver will do any necessary conversion from the database format into ASCII.

Note: All the data in the returned stream must be read prior to getting the value of any other column. The next call to a get method implicitly closes the stream. . Also, a stream may return 0 for available() whether there is data available or not.

Parameters:
columnIndex - the first column is 1, the second is 2, ...

Returns:

a Java input stream that delivers the database column value as a stream of one byte ASCII characters. If the value is SQL NULL then the result is null.

- **getUnicodeStream**

```
public abstract InputStream getUnicodeStream(int columnIndex) throws SQLException
```

A column value can be retrieved as a stream of Unicode characters and then read in chunks from the stream. This method is particularly suitable for retrieving large LONGVARCHAR values. The JDBC driver will do any necessary conversion from the database format into Unicode.

Note: All the data in the returned stream must be read prior to getting the value of any other column. The next call to a get method implicitly closes the stream. . Also, a stream may return 0 for available() whether there is data available or not.

Parameters:

columnIndex - the first column is 1, the second is 2, ...

Returns:

a Java input stream that delivers the database column value as a stream of two byte Unicode characters. If the value is SQL NULL then the result is null.

- **getBinaryStream**

```
public abstract InputStream getBinaryStream(int columnIndex) throws SQLException
```

A column value can be retrieved as a stream of uninterpreted bytes and then read in chunks from the stream. This method is particularly suitable for retrieving large LONGVARBINARY values.

Note: All the data in the returned stream must be read prior to getting the value of any other column. The next call to a get method implicitly closes the stream. Also, a stream may return 0 for available() whether there is data available or not.

Parameters:

columnIndex - the first column is 1, the second is 2, ...

Returns:

a Java input stream that delivers the database column value as a stream of uninterpreted bytes. If the value is SQL NULL then the result is null.

- **getString**

```
public abstract String getString(String columnName) throws SQLException
```

Get the value of a column in the current row as a Java String.

Parameters:

columnName - is the SQL name of the column

Returns:

the column value; if the value is SQL NULL, the result is null

- **getBoolean**

```
public abstract boolean getBoolean(String columnName) throws SQLException
```

Get the value of a column in the current row as a Java boolean.
Parameters:
columnName - is the SQL name of the column
Returns:
the column value; if the value is SQL NULL, the result is false

- **getByte**

```
public abstract byte getByte(String columnName) throws SQLException
```

Get the value of a column in the current row as a Java byte.
Parameters:
columnName - is the SQL name of the column
Returns:
the column value; if the value is SQL NULL, the result is 0

- **getShort**

```
public abstract short getShort(String columnName) throws SQLException
```

Get the value of a column in the current row as a Java short.
Parameters:
columnName - is the SQL name of the column
Returns:
the column value; if the value is SQL NULL, the result is 0

- **getInt**

```
public abstract int getInt(String columnName) throws SQLException
```

Get the value of a column in the current row as a Java int.
Parameters:
columnName - is the SQL name of the column
Returns:
the column value; if the value is SQL NULL, the result is 0

- **getLong**

```
public abstract long getLong(String columnName) throws SQLException
```

Get the value of a column in the current row as a Java long.
Parameters:
columnName - is the SQL name of the column
Returns:
the column value; if the value is SQL NULL, the result is 0

- **getFloat**

```
public abstract float getFloat(String columnName) throws SQLException
```

Get the value of a column in the current row as a Java float.
Parameters:
columnName - is the SQL name of the column
Returns:
the column value; if the value is SQL NULL, the result is 0

- **getDouble**

```
public abstract double getDouble(String columnName) throws SQLException
```

Get the value of a column in the current row as a Java double.
Parameters:
columnName - is the SQL name of the column
Returns:
the column value; if the value is SQL NULL, the result is 0

- **getBigDecimal**

```
public abstract BigDecimal getBigDecimal(String columnName,
                                         int scale) throws SQLException
```

Get the value of a column in the current row as a java.lang.BigDecimal object.
Parameters:
columnName - is the SQL name of the column
scale - the number of digits to the right of the decimal
Returns:
the column value; if the value is SQL NULL, the result is null

- **getBytes**

```
public abstract byte[] getBytes(String columnName) throws SQLException
```

Get the value of a column in the current row as a Java byte array. The bytes represent the raw values returned by the driver.
Parameters:
columnName - is the SQL name of the column
Returns:
the column value; if the value is SQL NULL, the result is null

- **getDate**

```
public abstract Date getDate(String columnName) throws SQLException
```

Get the value of a column in the current row as a java.sql.Date object.
Parameters:
columnName - is the SQL name of the column

Returns:
the column value; if the value is SQL NULL, the result is null

- **getTime**

```
public abstract Time getTime(String columnName) throws SQLException
```

Get the value of a column in the current row as a java.sql.Time object.
Parameters:
columnName - is the SQL name of the column
Returns:
the column value; if the value is SQL NULL, the result is null

- **getTimestamp**

```
public abstract Timestamp getTimestamp(String columnName) throws SQLException
```

Get the value of a column in the current row as a java.sql.Timestamp object.
Parameters:
columnName - is the SQL name of the column
Returns:
the column value; if the value is SQL NULL, the result is null

- **getAsciiStream**

```
public abstract InputStream getAsciiStream(String columnName) throws SQLException
```

A column value can be retrieved as a stream of ASCII characters and then read in chunks from the stream. This method is particularly suitable for retrieving large LONGVARCHAR values. The JDBC driver will do any necessary conversion from the database format into ASCII.

Note: All the data in the returned stream must be read prior to getting the value of any other column. The next call to a get method implicitly closes the stream.

Parameters:
columnName - is the SQL name of the column
Returns:
a Java input stream that delivers the database column value as a stream of one byte ASCII characters. If the value is SQL NULL then the result is null.

- **getUnicodeStream**

```
public abstract InputStream getUnicodeStream(String columnName) throws SQLExceptio
```

A column value can be retrieved as a stream of Unicode characters and then read in chunks from the stream. This method is particularly suitable for retrieving large LONGVARCHAR values. The JDBC driver will do any necessary conversion from the database format into Unicode.

Note: All the data in the returned stream must be read prior to getting the value of any other

column. The next call to a get method implicitly closes the stream.

Parameters:
columnName - is the SQL name of the column
Returns:
a Java input stream that delivers the database column value as a stream of two byte Unicode characters. If the value is SQL NULL then the result is null.

- **getBinaryStream**

```
public abstract InputStream getBinaryStream(String columnName) throws SQLException
```

A column value can be retrieved as a stream of uninterpreted bytes and then read in chunks from the stream. This method is particularly suitable for retrieving large LONGVARBINARY values.

Note: All the data in the returned stream must be read prior to getting the value of any other column. The next call to a get method implicitly closes the stream.

Parameters:
columnName - is the SQL name of the column
Returns:
a Java input stream that delivers the database column value as a stream of uninterpreted bytes. If the value is SQL NULL then the result is null.

- **getWarnings**

```
public abstract SQLWarning getWarnings() throws SQLException
```

The first warning reported by calls on this ResultSet is returned. Subsequent ResultSet warnings will be chained to this SQLWarning.

The warning chain is automatically cleared each time a new row is read.

Note: This warning chain only covers warnings caused by ResultSet methods. Any warning caused by statement methods (such as reading OUT parameters) will be chained on the Statement object.

Returns:
the first SQLWarning or null

- **clearWarnings**

```
public abstract void clearWarnings() throws SQLException
```

After this call getWarnings returns null until a new warning is reported for this ResultSet.

- **getCursorName**

```
public abstract String getCursorName() throws SQLException
```

Get the name of the SQL cursor used by this ResultSet.

In SQL, a result table is retrieved through a cursor that is named. The current row of a result can be updated or deleted using a positioned update/delete statement that references the cursor name.

JDBC supports this SQL feature by providing the name of the SQL cursor used by a ResultSet. The current row of a ResultSet is also the current row of this SQL cursor.

Note: If positioned update is not supported a SQLException is thrown

Returns:
the ResultSet's SQL cursor name

- **getMetaData**

```
public abstract ResultSetMetaData getMetaData() throws SQLException
```

The number, types and properties of a ResultSet's columns are provided by the getMetaData method.
Returns:
the description of a ResultSet's columns

- **getObject**

```
public abstract Object getObject(int columnIndex) throws SQLException
```

Get the value of a column in the current row as a Java object.

This method will return the value of the given column as a Java object. The type of the Java object will be the default Java Object type corresponding to the column's SQL type, following the mapping specified in the JDBC spec.

This method may also be used to read datatabase specific abstract data types.

Parameters:
columnIndex - the first column is 1, the second is 2, ...
Returns:
A java.lang.Object holding the column value.

- **getObject**

```
public abstract Object getObject(String columnName) throws SQLException
```

Get the value of a column in the current row as a Java object.

This method will return the value of the given column as a Java object. The type of the Java object will be the default Java Object type corresponding to the column's SQL type, following the mapping specified in the JDBC spec.

This method may also be used to read datatabase specific abstract data types.

Parameters:
columnName - is the SQL name of the column
Returns:
A java.lang.Object holding the column value.

- **findColumn**

```
public abstract int findColumn(String columnName) throws SQLException
```

Map a Resultset column name to a ResultSet column index.
Parameters:
columnName - the name of the column
Returns:
the column index

Interface java.sql.ResultSetMetaData

public interface **ResultSetMetaData**
extends Object

A ResultSetMetaData object can be used to find out about the types and properties of the columns in a ResultSet.

Variable Index

- **columnNoNulls**
  Does not allow NULL values.
- **columnNullable**
  Allows NULL values.
- **columnNullableUnknown**
  Nullability unknown.

Method Index

- **getCatalogName**(int)
  What's a column's table's catalog name?
- **getColumnCount**()
  What's the number of columns in the ResultSet?
- **getColumnDisplaySize**(int)

What's the column's normal max width in chars?

- **getColumnLabel**(int)
  What's the suggested column title for use in printouts and displays?
- **getColumnName**(int)
  What's a column's name?
- **getColumnType**(int)
  What's a column's SQL type?
- **getColumnTypeName**(int)
  What's a column's data source specific type name?
- **getPrecision**(int)
  What's a column's number of decimal digits?
- **getScale**(int)
  What's a column's number of digits to right of the decimal point?
- **getSchemaName**(int)
  What's a column's table's schema?
- **getTableName**(int)
  What's a column's table name?
- **isAutoIncrement**(int)
  Is the column automatically numbered, thus read-only?
- **isCaseSensitive**(int)
  Does a column's case matter?
- **isCurrency**(int)
  Is the column a cash value?
- **isDefinitelyWritable**(int)
  Will a write on the column definitely succeed?
- **isNullable**(int)
  Can you put a NULL in this column?
- **isReadOnly**(int)
  Is a column definitely not writable?
- **isSearchable**(int)
  Can the column be used in a where clause?
- **isSigned**(int)
  Is the column a signed number?
- **isWritable**(int)
  Is it possible for a write on the column to succeed?

Variables

- **columnNoNulls**

```
public final static int columnNoNulls
```

  Does not allow NULL values.

- **columnNullable**

```
public final static int columnNullable
```

Allows NULL values.

- **columnNullableUnknown**

```
public final static int columnNullableUnknown
```

Nullability unknown.

Methods

- **getColumnCount**

```
public abstract int getColumnCount() throws SQLException
```

What's the number of columns in the ResultSet?
Returns:
the number

- **isAutoIncrement**

```
public abstract boolean isAutoIncrement(int column) throws SQLException
```

Is the column automatically numbered, thus read-only?
Parameters:
column - the first column is 1, the second is 2, ...
Returns:
true if so

- **isCaseSensitive**

```
public abstract boolean isCaseSensitive(int column) throws SQLException
```

Does a column's case matter?
Parameters:
column - the first column is 1, the second is 2, ...
Returns:
true if so

- **isSearchable**

```
public abstract boolean isSearchable(int column) throws SQLException
```

Can the column be used in a where clause?
Parameters:
column - the first column is 1, the second is 2, ...
Returns:
true if so

- **isCurrency**

```
public abstract boolean isCurrency(int column) throws SQLException
```

  Is the column a cash value?
  Parameters:
    column - the first column is 1, the second is 2, ...
  Returns:
    true if so

- **isNullable**

```
public abstract int isNullable(int column) throws SQLException
```

  Can you put a NULL in this column?
  Parameters:
    column - the first column is 1, the second is 2, ...
  Returns:
    columnNoNulls, columnNullable or columnNullableUnknown

- **isSigned**

```
public abstract boolean isSigned(int column) throws SQLException
```

  Is the column a signed number?
  Parameters:
    column - the first column is 1, the second is 2, ...
  Returns:
    true if so

- **getColumnDisplaySize**

```
public abstract int getColumnDisplaySize(int column) throws SQLException
```

  What’s the column’s normal max width in chars?
  Parameters:
    column - the first column is 1, the second is 2, ...
  Returns:
    max width

- **getColumnLabel**

```
public abstract String getColumnLabel(int column) throws SQLException
```

  What’s the suggested column title for use in printouts and displays?
  Parameters:
    column - the first column is 1, the second is 2, ...
  Returns:
    true if so

- **getColumnName**

```
public abstract String getColumnName(int column) throws SQLException
```

  What's a column's name?
  Parameters:
    column - the first column is 1, the second is 2, ...
  Returns:
    column name

- **getSchemaName**

```
public abstract String getSchemaName(int column) throws SQLException
```

  What's a column's table's schema?
  Parameters:
    column - the first column is 1, the second is 2, ...
  Returns:
    schema name or "" if not applicable

- **getPrecision**

```
public abstract int getPrecision(int column) throws SQLException
```

  What's a column's number of decimal digits?
  Parameters:
    column - the first column is 1, the second is 2, ...
  Returns:
    precision

- **getScale**

```
public abstract int getScale(int column) throws SQLException
```

  What's a column's number of digits to right of the decimal point?
  Parameters:
    column - the first column is 1, the second is 2, ...
  Returns:
    scale

- **getTableName**

```
public abstract String getTableName(int column) throws SQLException
```

  What's a column's table name?
  Returns:
    table name or "" if not applicable

- **getCatalogName**

```
public abstract String getCatalogName(int column) throws SQLException
```

What's a column's table's catalog name?

Parameters:

column - the first column is 1, the second is 2, ...

Returns:

column name or "" if not applicable.

- **getColumnType**

```
public abstract int getColumnType(int column) throws SQLException
```

What's a column's SQL type?

Parameters:

column - the first column is 1, the second is 2, ...

Returns:

SQL type

See Also:

Types

- **getColumnTypeName**

```
public abstract String getColumnTypeName(int column) throws SQLException
```

What's a column's data source specific type name?

Parameters:

column - the first column is 1, the second is 2, ...

Returns:

type name

- **isReadOnly**

```
public abstract boolean isReadOnly(int column) throws SQLException
```

Is a column definitely not writable?

Parameters:

column - the first column is 1, the second is 2, ...

Returns:

true if so

- **isWritable**

```
public abstract boolean isWritable(int column) throws SQLException
```

Is it possible for a write on the column to succeed?

Parameters:

column - the first column is 1, the second is 2, ...

Returns:

true if so

- **isDefinitelyWritable**

```
public abstract boolean isDefinitelyWritable(int column) throws SQLException
```

Will a write on the column definitely succeed?

Parameters:

column - the first column is 1, the second is 2, ...

Returns:

true if so

Interface java.sql.Statement

public interface **Statement**
extends Object

A Statement object is used for executing a static SQL statement and obtaining the results produced by it.

Only one ResultSet per Statement can be open at any point in time. Therefore, if the reading of one ResultSet is interleaved with the reading of another, each must have been generated by different Statements. All statement execute methods implicitly close a statment's current ResultSet if an open one exists.

See Also:

createStatement, ResultSet

Method Index

The maxFieldSize limit (in bytes) is the maximum amount of data returned for any column value; it only applies to BINARY, VARBINARY, LONGVARBINARY, CHAR, VARCHAR, and LONGVARCHAR columns.

- **getMaxRows**()
  The maxRows limit is the maximum number of rows that a ResultSet can contain.
- **getMoreResults**()
  getMoreResults moves to a Statement's next result.
- **getQueryTimeout**()
  The queryTimeout limit is the number of seconds the driver will wait for a Statement to execute.
- **getResultSet**()
  getResultSet returns the current result as a ResultSet.
- **getUpdateCount**()
  getUpdateCount returns the current result as an update count; if the result is a ResultSet or there are no more results, -1 is returned.
- **getWarnings**()
  The first warning reported by calls on this Statement is returned.
- **setCursorName**(String)
  setCursorname defines the SQL cursor name that will be used by subsequent Statement execute methods.
- **setEscapeProcessing**(boolean)
  If escape scanning is on (the default), the driver will do escape substitution before sending the SQL to the database.
- **setMaxFieldSize**(int)
  The maxFieldSize limit (in bytes) is set to limit the size of data that can be returned for any column value; it only applies to BINARY, VARBINARY, LONGVARBINARY, CHAR, VARCHAR, and LONGVARCHAR fields.
- **setMaxRows**(int)
  The maxRows limit is set to limit the number of rows that any ResultSet can contain.
- **setQueryTimeout**(int)
  The queryTimeout limit is the number of seconds the driver will wait for a Statement to execute.

Methods

- **executeQuery**

```
public abstract ResultSet executeQuery(String sql) throws SQLException
```

Execute a SQL statement that returns a single ResultSet.

Parameters:

sql - typically this is a static SQL SELECT statement

Returns:

a ResultSet that contains the data produced by the query; never null

- **executeUpdate**

```
public abstract int executeUpdate(String sql) throws SQLException
```

Execute a SQL INSERT, UPDATE or DELETE statement. In addition, SQL statements that return nothing such as SQL DDL statements can be executed.

Parameters:

sql - a SQL INSERT, UPDATE or DELETE statement or a SQL statement that returns nothing

Returns:

either the row count for INSERT, UPDATE or DELETE or 0 for SQL statements that return nothing

- **close**

```
public abstract void close() throws SQLException
```

In many cases, it is desirable to immediately release a Statements's database and JDBC resources instead of waiting for this to happen when it is automatically closed; the close method provides this immediate release.

Note: A Statement is automatically closed when it is garbage collected. When a Statement is closed, its current ResultSet, if one exists, is also closed.

- **getMaxFieldSize**

```
public abstract int getMaxFieldSize() throws SQLException
```

The maxFieldSize limit (in bytes) is the maximum amount of data returned for any column value; it only applies to BINARY, VARBINARY, LONGVARBINARY, CHAR, VARCHAR, and LONGVARCHAR columns. If the limit is exceeded, the excess data is silently discarded.

Returns:

the current max column size limit; zero means unlimited

- **setMaxFieldSize**

```
public abstract void setMaxFieldSize(int max) throws SQLException
```

The maxFieldSize limit (in bytes) is set to limit the size of data that can be returned for any column value; it only applies to BINARY, VARBINARY, LONGVARBINARY, CHAR, VARCHAR, and LONGVARCHAR fields. If the limit is exceeded, the excess data is silently discarded. For maximum portability use values greater than 256.

Parameters:

max - the new max column size limit; zero means unlimited

- **getMaxRows**

```
public abstract int getMaxRows() throws SQLException
```

The maxRows limit is the maximum number of rows that a ResultSet can contain. If the limit is exceeded, the excess rows are silently dropped.

Returns:

the current max row limit; zero means unlimited

- **setMaxRows**

```
public abstract void setMaxRows(int max) throws SQLException
```

The maxRows limit is set to limit the number of rows that any ResultSet can contain. If the limit is exceeded, the excess rows are silently dropped.

Parameters:

max - the new max rows limit; zero means unlimited

- **setEscapeProcessing**

```
public abstract void setEscapeProcessing(boolean enable) throws SQLException
```

If escape scanning is on (the default), the driver will do escape substitution before sending the SQL to the database. Note: Since prepared statements have usually been parsed prior to making this call, disabling escape processing for prepared statements will like have no affect.

Parameters:

enable - true to enable; false to disable

- **getQueryTimeout**

```
public abstract int getQueryTimeout() throws SQLException
```

The queryTimeout limit is the number of seconds the driver will wait for a Statement to execute. If the limit is exceeded, a SQLException is thrown.

Returns:

the current query timeout limit in seconds; zero means unlimited

- **setQueryTimeout**

```
public abstract void setQueryTimeout(int seconds) throws SQLException
```

The queryTimeout limit is the number of seconds the driver will wait for a Statement to execute. If the limit is exceeded, a SQLException is thrown.

Parameters:

seconds - the new query timeout limit in seconds; zero means unlimited

- **cancel**

```
public abstract void cancel() throws SQLException
```

Cancel can be used by one thread to cancel a statement that is being executed by another thread.

- **getWarnings**

```
public abstract SQLWarning getWarnings() throws SQLException
```

The first warning reported by calls on this Statement is returned. A Statment’s execute methods clear its SQLWarning chain. Subsequent Statement warnings will be chained to this SQLWarning.

The warning chain is automatically cleared each time a statement is (re)executed.

Note: If you are processing a ResultSet then any warnings associated with ResultSet reads will be chained on the ResultSet object.

Returns:
the first SQLWarning or null

- **clearWarnings**

```
public abstract void clearWarnings() throws SQLException
```

After this call, getWarnings returns null until a new warning is reported for this Statement.

- **setCursorName**

```
public abstract void setCursorName(String name) throws SQLException
```

setCursorname defines the SQL cursor name that will be used by subsequent Statement execute methods. This name can then be used in SQL positioned update/delete statements to identify the current row in the ResultSet generated by this statement. If the database doesn't support positioned update/delete, this method is a noop.

Note: By definition, positioned update/delete execution must be done by a different Statement than the one which generated the ResultSet being used for positioning. Also, cursor names must be unique within a Connection.

Parameters:
name - the new cursor name.

- **execute**

```
public abstract boolean execute(String sql) throws SQLException
```

Execute a SQL statement that may return multiple results. Under some (uncommon) situations a single SQL statement may return multiple result sets and/or update counts. Normally you can ignore this, unless you're executing a stored procedure that you know may return multiple results, or unless you're dynamically executing an unknown SQL string. The "execute", "getMoreResults", "getResultSet" and "getUpdateCount" methods let you navigate through multiple results. The "execute" method executes a SQL statement and indicates the form of the first result. You can then use getResultSet or getUpdateCount to retrieve the result, and getMoreResults to move to any subsequent result(s).

Parameters:
sql - any SQL statement

Returns:
true if the next result is a ResultSet; false if it is an update count or there are no more results

See Also:
getResultSet, getUpdateCount, getMoreResults

- **getResultSet**

```
public abstract ResultSet getResultSet() throws SQLException
```

getResultSet returns the current result as a ResultSet. It should only be called once per result.

Returns:

the current result as a ResultSet; null if the result is an update count or there are no more results

See Also:

execute

- **getUpdateCount**

```
public abstract int getUpdateCount() throws SQLException
```

getUpdateCount returns the current result as an update count; if the result is a ResultSet or there are no more results, -1 is returned. It should only be called once per result.

Returns:

the current result as an update count; -1 if it is a ResultSet or there are no more results

See Also:

execute

- **getMoreResults**

```
public abstract boolean getMoreResults() throws SQLException
```

getMoreResults moves to a Statement's next result. It returns true if this result is a ResultSet. getMoreResults also implicitly closes any current ResultSet obtained with getResultSet. There are no more results when (!getMoreResults() && (getUpdateCount() == -1)

Returns:

true if the next result is a ResultSet; false if it is an update count or there are no more results

See Also:

execute
