27

The JDBC API

The java.sql package contains the entire JDBC API. It first became part of the core Java libraries with the 1.1 release. Classes new as of JDK 1.2 are indicated by the "Availability" header. Deprecated methods are preceded by a hash (#) mark. New JDK 1.2 methods in old JDK 1.1 classes are shown in bold. Figure 27-1 shows the entire java.sql package.

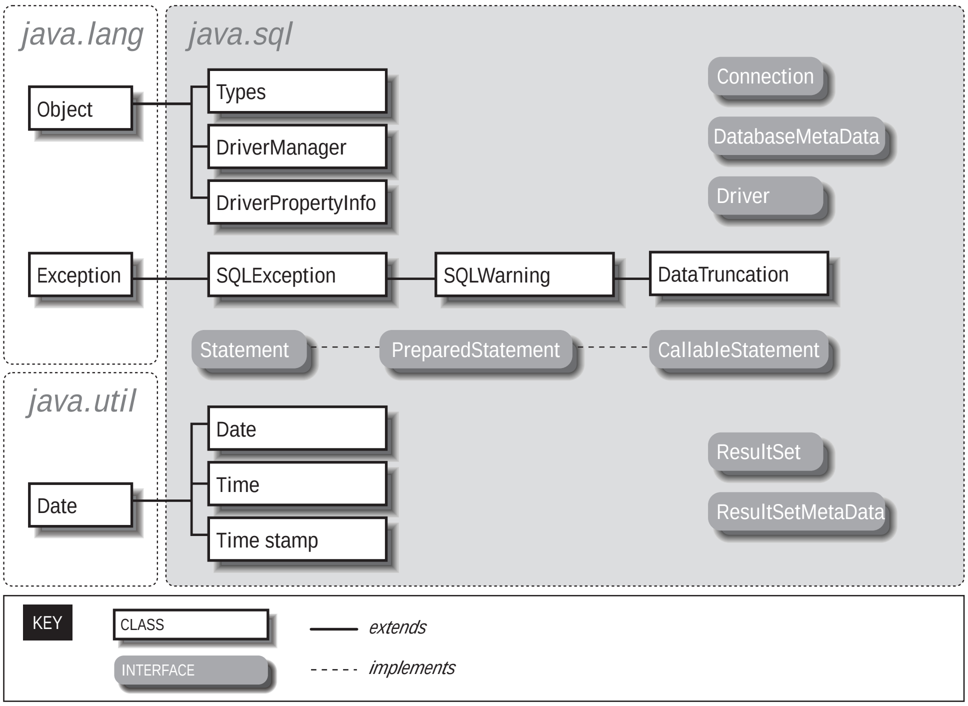

Figure 27-1. The classes and interfaces of the java.sql package

Array

Synopsis

Class Name:java.sql.Array
Superclass:None
Immediate Subclasses: None
Interfaces Implemented: None
Availability: New as of JDK 1.2

Description

The Array interface is a new addition to JDBC that supports SQL3 array objects. The default duration of a reference to a SQL array is for the life of the transaction in which it was created.

Class Summary

```
public interface Array {
  Object getArray() throws SQLException;
  Object getArray(Map map) throws SQLException;
  Object getArray(long index, int count)
    throws SQLException;
  Object getArray(long index, int count, Map map)
    throws SQLException;
  int getBaseType() throws SQLException;
  String getBaseTypeName() throws SQLException;
  ResultSet getResultSet() throws SQLException;
  ResultSet getResultSet(Map map) throws SQLException;
  ResultSet getResultSet(long index, int count)
    throws SQLException;
  ResultSet getResultSet(long index, int count,
                  Map map) throws SQLException
}
```

Object Methods

getArray( )

```
public Object getArray() throws SQLException
public Object getArray(Map map) throws SQLException
public Object getArray(long index, int count)
  throws SQLException
public Object getArray(long index, int count, Map map)
  throws SQLException
```

Description: Place the contents of this SQL array into a Java language array or, instead, into the Java type specified by a provided Map. If a map is specified but no match is found in there, then the default mapping to a Java array is used. The two versions that accept an array index and element count enable you to place a subset of the elements in the array.

getBaseType( )

```
public int getBaseType() throws SQLException
```

Description: Provides the JDBC type of the elements of this array.

getBaseTypeName( )

```
public String getBaseTypeName() throws SQLException
```

Description: Provides the SQL type name for the elements of this array.

getResultSet( )

```
public ResultSet getResultSet() throws SQLException
public ResultSet getResultSet(Map map)
  throws SQLException
public ResultSet getResultSet(long index, int count)
  throws SQLException
public ResultSet getResultSet(long index, int count,
                              Map map)
  throws SQLException
```

Description: Provides a result set that contains the array's elements as rows. If appropriate, the elements are mapped using the type map for the connection, or the specified type map if you pass one. Each row contains two columns: the first column is the index number (starting with 1), and the second column is the actual value.

Blob

Synopsis

Class Name: java.sql.Blob
Superclass: None
Immediate Subclasses: None
Interfaces Implemented: None
Availability: New as of JDK 1.2

Description

The JDBC Blob interface represents a SQL BLOB. BLOB stands for "binary large object" and is a relational database representation of a large piece of binary data. The value of using a BLOB is that you can manipulate the BLOB as a Java object without retrieving all of the data behind the BLOB from the database. A BLOB object is only valid for the duration of the transaction in which it was created.

Class Summary

```
public interface Blob {
  InputStream getBinaryStream() throws SQLException;
  byte[] getBytes(long pos, int count)
    throws SQLException;
  long length() throws SQLException;
  long position(byte[] pattern, long start)
```

```
        throws SQLException;
    long position(Blob pattern, long start)
        throws SQLException;
}
```

Object Methods

getBinaryStream( )

```
public InputStream getBinaryStream() throws SQLException
```

Description: Retrieves the data that makes up the binary object as a stream from the database.

getBytes( )

```
public byte[] getBytes(long pos, int count)
    throws SQLException
```

Description: Returns the data that makes up the underlying binary object in part or in whole as an array of bytes. You can get a subset of the binary data by specifying a nonzero starting index or by specifying a number of bytes less than the object's length.

length( )

```
public long length() throws SQLException
```

Description: Provides the number of bytes that make up the BLOB.

position( )

```
public long position(byte[] pattern, long start)
    throws SQLException
public long position(Blob pattern, long start)
    throws SQLException
```

Description: Searches this Blob for the specified pattern and returns the byte at which the specified pattern occurs within this Blob. If the pattern does not occur, then this method will return -1.

CallableStatement

Synopsis

Class Name: java.sql.CallableStatement
Superclass: java.sql.PreparedStatement
Immediate Subclasses: None
Interfaces Implemented: None
Availability: JDK 1.1

Description

The CallableStatement is an extension of the PreparedStatement interface that provides support for SQL stored procedures. It specifies methods that handle the binding of output parameters. JDBC prescribes a standard form in which stored procedures should appear independent of the DBMS being used. The format is:

```
{? = call …}
{call …}
```

Each question mark is a place holder for an input or output parameter. The first syntax provides for a single result parameter. The second syntax has no result parameters. The parameters are referred to sequentially with the first question mark holding the place for parameter 1.

Before executing a stored procedure, all output parameters should be registered using the registerOutParameter() method. You then bind the input parameters using the various set methods, and then execute the stored procedure.

Class Summary

```
public interface CallableStatement extends PreparedStatement {
  Array getArray(int index) throws SQLException;
  BigDecimal getBigDecimal(int index)
    throws SQLException;
  #BigDecimal getBigDecimal(int index, int scale)
    throws SQLException;
  Blob getBlob(int index) throws SQLException;
  boolean getBoolean(int index) throws SQLException;
  byte getByte(int index) throws SQLException;
  byte[] getBytes(int index) throws SQLException;
  Clob getClob(int index) throws SQLException;
  java.sql.Date getDate(int index, Calendar cal)
    throws SQLException;
  java.sql.Date getDate(int index) throws SQLException;
  double getDouble(int index) throws SQLException;
  float getFloat(int index) throws SQLException;
  int getInt(int index) throws SQLException;
  long getLong(int index) throws SQLException;
  Object getObject(int index) throws SQLException;
  Object getObject(int index, Map map)
    throws SQLException;
  Ref getRef(int index) throws SQLException;
  short getShort(int index) throws SQLException;
  String getString(int index) throws SQLException;
  java.sql.Time getTime(int index) throws SQLException;
  java.sql.Time getTime(int index, Calendar cal)
    throws SQLException;
  java.sql.Timestamp getTimestamp(int index)
    throws SQLException;
  java.sql.Timestamp getTimestamp(int index,
                                  Calendar cal)
    throws SQLException;
  void registerOutParameter(int index, int type)
    throws SQLException;
  void registerOutParameter(int index, int type,
                            int scale)
```

```
        throws SQLException;
    void registerOutParameter(int index, int type,
                    String typename)
        throws SQLException;
    boolean wasNull() throws SQLException;
}
```

Object Methods

getBigDecimal( )

```
public BigDecimal getBigDecimal(int index)
    throws SQLException
#public BigDecimal getBigDecimal(int index, int scale)
    throws SQLException
```

Description: Returns the value of the parameter specified by the index parameter as a Java BigDecimal with a scale specified by the scale argument. The scale is a nonnegative number representing the number of digits to the right of the decimal. Parameter indices start at 1; parameter 1 is thus index 1.

getArray( ), getBlob( ), getBoolean( ), getByte( ), getBytes( ), getClob( ), getDouble( ), getFloat( ), getInt( ), getLong( ), getRef( ), getShort( ), and getString( )

```
public Array getArray(int index)
    throws SQLException
public Blob getBlob(int index) throws SQLException
public boolean getBoolean(int index) throws SQLException
public byte getByte(int index) throws SQLException
public byte[] getBytes(int index) throws SQLException
public Clob getClob(int index) throws SQLException
public double getDouble(int index) throws SQLException
public float getFloat(int index) throws SQLException
public int getInt(int index) throws SQLException
public long getLong(int index) throws SQLException
public Ref getRef(int index) throws SQLException
public short getShort(int index) throws SQLException
public String getString(int index) throws SQLException
```

Description: Returns the value of the parameter specified by the index argument as the Java datatype indicated by the method name.

getDate( ), getTime( ), and getTimestamp( )

```
public Date getDate(int index) throws SQLException
public Date getDate(int index, Calendar cal)
    throws SQLException
public Time getTime(int index) throws SQLException
public Time getTime(int index, Calendar cal)
    throws SQLException
public Timestamp getTimestamp(int index)
    throws SQLException
public Timestamp getTimestamp(int index, Calendar cal)
    throws SQLException
```

Description: JDBC provides refinements on the basic java.util.Date object more suitable to database programming. These methods provide ways to access return values from a CallableStatement as a Date, Time, or Timestamp object. The new JDK 1.2 variants allow you to specify a Calendar.

getObject( )

```
public Object getObject(int index) throws SQLException
public Object getObject(int index, Map map)
  throws SQLException
```

Description: Like the other getXXX() methods, this method returns the value of the specified output parameter. In the case of getObject(), however, the JDBC driver chooses the Java class that corresponds to the SQL type registered for this parameter using registerOutParameter() or according to the specified type map.

registerOutParameter( )

```
public void registerOutParameter(int index, int type)
  throws SQLException
public void registerOutParameter(int index, int type,
                     int scale)
  throws SQLException
public void registerOutParameter(int index, int type,
                     String typename)
  throws SQLException
```

Description: Before executing any stored procedure using a CallableStatement, you must register each of the output parameters. This method registers the java.sql.Type of an output parameter for a stored procedure. The first parameter specifies the output parameter being registered and the second the java.sql.Type to register. The three-argument version of this method is for BigDecimal types that require a scale. You later read the output parameters using the corresponding getXXX() method or getObject(). The third version of this method is new to JDK 1.2 and provides a way to map REF SQL types or custom SQL types.

wasNull( )

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

Description: If the last value you read using a getXXX() call was SQL NULL, this method will return true.

Clob

Synopsis

Class Name: java.sql.Clob
Superclass: None
Immediate Subclasses: None
Interfaces Implemented: None
Availability: New as of JDK 1.2

Description

CLOB is a SQL3 type that stands for "character large object." Like a BLOB, a CLOB represents a very large chunk of data in the database. Unlike a BLOB, a CLOB represents text stored using some sort of character encoding. The point of a CLOB type as opposed to a CHAR or VARCHAR type is that CLOB data, like BLOB data, can be retrieved as a stream instead of all at once.

Class Summary

```
public interface Clob {
  InputStream getAsciiStream() throws SQLException;
  Reader getCharacterStream() throws SQLException;
  String getSubString(long pos, int count)
    throws SQLException;
  long length() throws SQLException;
  long position(String pattern, long start)
    throws SQLException;
  long position(Clob pattern, long start)
    throws SQLException;
}
```

Object Methods

getAsciiStream( )

```
public InputStream getAsciiStream() throws SQLException
```

Description: Provides access to the data that makes up this Clob via an ASCII stream.

getCharacterStream( )

```
public Reader getCharacterStream() throws SQLException
```

Description: Provides access to the data that makes up this Clob via a Unicode stream.

getSubString( )

```
public String getSubString(long pos, int count)
  throws SQLException
```

Description: Returns a substring of the Clob starting at the named position up to the number of character specified by the count value.

length( )

```
public long length() throws SQLException
```

Description: Provides the number of characters that make up the Clob.

position( )

```
public long position(String pattern, long start)
  throws SQLException;
public long position(Clob pattern, long start)
  throws SQLException;
```

Description: Searches the Clob for the specified pattern starting at the specified start point. If the pattern is found within the Clob, the index at which the pattern first occurs is returned. If it does not exist within the Clob, then this method returns -1.

Connection

Synopsis

Class Name: java.sql.Connection
Superclass: None
Immediate Subclasses: None
Interfaces Implemented: None
Availability: JDK 1.1

Description

The Connection class is the JDBC representation of a database session. It provides an application with Statement objects (and its subclasses) for that session. It also handles the transaction management for those statements. By default, each statement is committed immediately upon execution. You can use the Connection object to turn off this Autocommit feature for the session. In that event, you must expressly send commits, or any statements executed will be lost.

Class Summary

```
public interface Connection {
  static public final int TRANSACTION_NONE;
  static public final int TRANSACTION_READ_UNCOMMITTED;
  static public final int TRANSACTION_READ_COMMITTED;
  static public final int TRANSACTION_REPEATABLE_READ;
  static public final int TRANSACTION_SERIALIZABLE;

  void clearWarnings() throws SQLException;
  void close() throws SQLException;
  void commit() throws SQLException;
  Statement createStatement() throws SQLException;
  Statement createStatement(int type, int concur)
    throws SQLException;
  boolean getAutoCommit() throws SQLException;
  String getCatalog() throws SQLException;
  Map getTypeMap() throws SQLException;
  DatabaseMetaData getMetaData() throws SQLException;
  int getTransactionIsolation() throws SQLException;
  SQLWarning getWarnings() throws SQLException;
  boolean isClosed() throws SQLException;
  boolean isReadOnly() throws SQLException;
  String nativeSQL(String sql) throws SQLException;
  CallableStatement prepareCall(String sql)
    throws SQLException;
  CallableStatement prepareCall(String sql, int type,
                     int concur)
    throws SQLException;
  PreparedStatement prepareStatement(String sql)
    throws SQLException;
  PreparedStatement prepareStatement(String sql,
                                    int type,
                                    int concur)
    throws SQLException;
  void rollback() throws SQLException;
  void setAutoCommit(boolean ac) throws SQLException;
  void setCatalog(String catalog) throws SQLException;
  void setReadOnly(boolean ro) throws SQLException;
  void setTransactionIsolation(int level)
    throws SQLException;
  void setTypeMap(Map map) throws SQLException;
}
```

Class Attributes

TRANSACTION_NONE

```
static public final int TRANSACTION_NONE
```

Description: Transactions are not supported.

TRANSACTION_READ_UNCOMMITTED

```
static public final int TRANSACTION_READ_UNCOMMITTED
```

Description: This transaction isolation level allows uncommitted changes by one transaction to be readable by other transactions.

TRANSACTION_READ_COMMITTED

```
static public final int TRANSACTION_READ_COMMITTED
```

Description: This transaction isolation level prevents dirty reads from occurring. In other words, changes by a TRANSACTION_READ_COMMITTED transaction are invisible to other transactions until the transaction making the change commits those changes.

TRANSACTION_REPEATABLE_READ

```
static public final int TRANSACTION_REPEATABLE_READ
```

Description: This transaction isolation level prevents dirty reads and nonrepeatable reads. A nonrepeatable read is one where one transaction reads a row, a second transaction alters the row, and the first transaction rereads the row, getting different values the second time.

Object Methods

clearWarnings( )

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

Description: Clears out all the warnings associated with this Connection so that getWarnings() will return null until a new warning is reported.

close( )

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

Description: This method manually releases all resources (such as network connections and database locks) associated with a given JDBC Connection. This method is automatically called when garbage collection occurs; however, it is best to manually close a Connection once you are done with it.

commit( )

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

Description: This method makes permanent the changes created by all statements associated with this Connection since the last commit or rollback was issued. It should only be used when Autocommit is off. It does not commit changes made by statements associated with other Connection objects.

createStatement( )

```
public Statement createStatement() throws SQLException
public Statement createStatement(int type, int concur)
  throws SQLException
```

Description: This method creates a Statement object associated with this Connection session. The no argument version of this method creates a Statement whose ResultSet instances are type forward-only and read-only concurrency.

getAutoCommit( ) and setAutoCommit( )

```
public boolean getAutoCommit() throws SQLException
public void setAutoCommit(boolean ac)
  throws SQLException
```

Description: By default, all Connection objects are in Autocommit mode. With Autocommit mode on, each statement is committed as it is executed. An application may instead choose to manually commit a series of statements together as a single transaction. In this case, you use the setAutoCommit() method to turn Autocommit off. You then follow your statements with a call to commit() or rollback() depending on the success or failure of the transaction.

When in Autocommit mode, a statement is committed either when the statement completes or when the next statement is executed, whichever is first. For statements returning a ResultSet, the statement completes when the last row has been retrieved or the ResultSet has been closed. If a statement returns multiple result sets, the commit occurs when the last row of the last ResultSet object has been retrieved.

getCatalog( ) and setCatalog( )

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

Description: If a driver supports catalogs, then you use setCatalog() to select a subspace of the database with the specified catalog name. If the driver does not support catalogs, it will ignore this request.

getMetaData( )

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

Description: The DatabaseMetaData class provides methods that describe a database's tables, SQL support, stored procedures, and other information relating to the database and this Connection, which are not directly related to executing statements and retrieving result sets. This method provides an instance of the DatabaseMetaData class for this Connection.

getTransactionIsolation( ) and setTransactionIsolation( )

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

Description: Sets the Connection object's current transaction isolation level using one of the class attributes for the Connection interface. Those levels are called TRANSACTION_NONE, TRANSACTION_READ_UNCOMMITTED, TRANSACTION_READ_COMMITTED, and TRANSACTION_REPEATABLE_READ.

getTypeMap( ) and setTypeMap( )

```
public Map getTypeMap() throws SQLException
public void setTypeMap(Map map) throws SQLException
```

Description: You can use these methods to define or retrieve a custom mapping for SQL structured types and distinct types for all statements associated with this connection.

getWarnings( )

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

Description: Returns the first warning in the chain of warnings associated with this Connection object.

isClosed( )

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

Description: Returns true if the Connection has been closed.

isReadOnly( ) and setReadOnly( )

```
public boolean isReadOnly() throws SQLException
public void setReadOnly(boolean ro) throws SQLException
```

Description: Some databases can optimize for read-only database access. The setReadOnly() method provides you with a way to put a Connection into read-only mode so that those optimizations occur. You cannot call setReadOnly() while in the middle of a transaction.

nativeSQL( )

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

Description: Many databases may not actually support the same SQL required by JDBC. This method allows an application to see the native SQL for a given JDBC SQL string.

prepareCall( )

```
public CallableStatement prepareCall(String sql)
```

```
   throws SQLException
public CallableStatement prepareCall(String sql,
                         int type,
                         int concur)
   throws SQLException
```

Description: Given a particular SQL string, this method creates a CallableStatement object associated with this Connection session. This is the preferred way of handling stored procedures. The default (no argument) version of this method provides a CallableStatement whose ResultSet instances are type forward-only and read-only concurrency.

prepareStatement( )

```
public PreparedStatement prepareStatement(String sql)
   throws SQLException
public PreparedStatement prepareStatement(String sql,
                          int type,
                          int concur)
   throws SQLException
```

Description: Provides a PreparedStatement object to be associated with this Connection session. This is the preferred way of handling precompiled SQL statements. The default (no argument) version of this method provides a PreparedStatement whose ResultSet instances are type forward-only and read-only concurrency.

rollback( )

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

Description: Aborts all changes made by statements associated with this Connection since the last time a commit or rollback was issued. If you want to make those changes at a later time, your application will have to reexecute the statements that made those changes. This should be used only when auto-commit is off.

DatabaseMetaData

Synopsis

Class Name: java.sql.DatabaseMetaData
Superclass: None
Immediate Subclasses: None
Interfaces Implemented: None
Availability: New as of JDK 1.1

Description

This class provides a lot of information about the database to which a Connection object is connected. In many cases, it returns this information in the form of JDBC

ResultSet objects. For databases that do not support a particular kind of metadata, DatabaseMetaData will throw an SQLException.

DatabaseMetaData methods take string patterns as arguments where specific tokens within the String are interpreted to have a certain meaning. % matches any substring of 0 or more characters and _ matches any one character. You can pass null to methods in place of string pattern arguments; this means that the argument's criteria should be dropped from the search.

Class Summary

```
public interface DatabaseMetaData {
  static public final int bestRowTemporary;
  static public final int bestRowTransaction;
  static public final int bestRowSession;
  static public final int bestRowUnknown;
  static public final int bestRowNotPseudo;
  static public final int bestRowPseudo;
  static public final int columnNoNulls;
  static public final int columnNullable;
  static public final int columnNullableUnknown;
  static public final int importedKeyCascade;
  static public final int importedKeyRestrict;
  static public final int importedKeySetNull;
  static public final int importedKeyNoAction;
  static public final int importedKeySetDefault;
  static public final int importedKeyInitiallyDeferred;
  static public final int importedKeyInitiallyImmediate;
  static public final int importedKeyNotDeferrable;
  static public final int procedureResultUnknown;
  static public final int procedureNoResult;
  static public final int procedureReturnsResult;
  static public final int procedureColumnUnknown;
  static public final int procedureColumnIn;
  static public final int procedureColumnOut;
  static public final int procedureColumnReturn;
  static public final int procedureColumnResult;
  static public final int procedureNoNulls;
  static public final int procedureNullable;
  static public final int procedureNullableUnknown;
  static public final short tableIndexStatistic;
  static public final short tableIndexClustered;
  static public final short tableIndexHashed;
  static public final short tableIndexOther;
  static public final int typeNoNulls;
  static public final int typeNullable;
  static public final int typeNullableUnknown;
  static public final int typePredNone;
  static public final int typePredChar;
  static public final int typePredBasic;
  static public final int typeSearchable;
  static public final int versionColumnUnknown;
  static public final int versionColumnNotPseudo;
  static public final int versionColumnPseudo;

  boolean allProceduresAreCallable()
     throws SQLException;
  boolean allTablesAreSelectable() throws SQLException;
  boolean dataDefinitionCausesTransactionCommit()
     throws SQLException;
  boolean dataDefinitionIgnoredInTransactions()
     throws SQLException;
  ResultSet getBestRowIdentifier(String catalog,
     String schema, String table, int scope,
     boolean nullable)
     throws SQLException;
  ResultSet getCatalogs() throws SQLException;
  String getCatalogSeparator() throws SQLException;
  String getCatalogTerm() throws SQLException;
  ResultSet getColumnPriveleges(String catalog,
        String spat, String table,
        String cpat) throws SQLException;
  ResultSet getColumns(String catalog,
        String spat, String tpat,
        String cpat) throws SQLException;
  ResultSet getCrossReference(String primaryCatalog,
        String primarySchema, String primaryTable,
        String foreignCatalog, String foreignSchema,
        String foreignTable) throws SQLException;
  String getDatabaseProductName() throws SQLException;
  String getDatabaseProductVersion()
     throws SQLException;
  int getDefaultTransactionIsolation()
     throws SQLException;
  int getDriverMajorVersion();
  int getDriverMinorVersion();
  String getDriverName() throws SQLException;
  String getDriverVersion() throws SQLException;
  ResultSet getExportedKeys(String catalog,
     String schema, String table)
     throws SQLException;
  String getExtraNameCharacters() throws SQLException;
  String getIdentifierQuoteString() throws SQLException;
  ResultSet getImportedKeys(String catalog,
     String schema, String table) throws SQLException;
  ResultSet getIndexInfo(String catalog,
     String schema, String table, boolean unique,
     boolean approximate) throws SQLException;
  int getMaxBinaryLiteralLength() throws SQLException;
  int getMaxCatalogNameLength() throws SQLException;
  int getMaxCharLiteralLength() throws SQLException;
  int getMaxcnameLength() throws SQLException;
  int getMaxColumnsInGroupBy() throws SQLException;
  int getMaxColumnsInIndex() throws SQLException;
  int getMaxColumnsInOrderBy() throws SQLException;
  int getMaxColumnsInSelect() throws SQLException;
  int getMaxColumnsInTable() throws SQLException;
  int getMaxConnections() throws SQLException;
  int getMaxIndexLength() throws SQLException;
  int getMaxProcedureNameLength()
     throws SQLException;
  int getMaxRowSize() throws SQLException;
  int getMaxRowSizeIncludeBlobs()
     throws SQLException;
  int getMaxSchemaNameLength() throws SQLException;
  int getMaxStatementLength() throws SQLException;
  int getMaxStatements() throws SQLException;
  int getMaxTableNameLength() throws SQLException;
  int getMaxTablesInSelect() throws SQLException;
  int getMaxUserNameLength() throws SQLException;
  String getNumericFunctions() throws SQLException;
  ResultSet getPrimaryKeys(String catalog,
     String schema, String table) throws SQLException;
  ResultSet getProcedureColumns(String catalog,
     String schemePattern, String procedureNamePattern,
    String cnamePattern) throws SQLException;
  String getProcedureTerm() throws SQLException;
  ResultSet getProcedures(String catalog,
     String schemaPattern, String procedureNamePattern)
     throws SQLException;
  public abstract ResultSet getSchemas() throws SQLException;
  public abstract String getSchemaTerm() throws SQLException;
  String getSearchStringEscape() throws SQLException;
  String getSQLKeywords() throws SQLException;
  String getStringFunctions() throws  SQLException;
  String getSystemFunctions() throws SQLException;
  ResultSet getTablePriveleges(String catalog,
     String schemaPattern, String tableNamePattern)
     throws SQLException;
  ResultSet getTableTypes() throws SQLException;
  ResultSet getTables(String catalog,
     String schemaPattern, String tableNamePattern,
     String types[]) throws SQLException;
  String getTimeDateFunctions() throws SQLException;
  ResultSet getTypeInfo() throws SQLException;
  String getURL() throws SQLException;
  String getUserName() throws SQLException;
  ResultSet getVersionColumns(String catalog,
     String schema, String table) throws SQLException;
  boolean isCatalogAtStart() throws SQLException;
  boolean isReadOnly() throws SQLException;
  boolean nullPlusNonNullIsNull() throws SQLException;
  boolean nullsAreSortedHigh() throws SQLException;
  boolean nullsAreSortedLow() throws SQLException;
  boolean nullsAreSortedAtStart() throws SQLException;
  boolean nullsAreSortedAtEnd() throws SQLException;
  boolean storesLowerCaseIdentifiers()
     throws SQLException;
  boolean storesLowerCaseQuotedIdentifiers()
     throws SQLException;
  boolean storesMixedCaseIdentifiers()
```

```
   throws SQLException;
boolean storesMixedCaseQuotedIdentifiers()
   throws SQLException;
boolean storesUpperCaseIdentifiers()
   throws SQLException;
boolean storesUpperCaseQuotedIdentifiers()
   throws SQLException;
boolean supportsAlterTableWithAddColumn()
   throws SQLException;
boolean supportsAlterTableWithDropColumn()
   throws SQLException;
boolean supportsANSI92FullSQL() throws SQLException;
boolean supportsANSI92IntermediateSQL()
   throws SQLException;
boolean supportsCatalogsInDataManipulation()
   throws SQLException;
boolean suppportsCatalogsInIndexDefinitions()
   throws SQLException;
boolean supportsCatalogsInPrivelegeDefinitions()
   throws SQLException;
boolean supportsCatalogsInProcedureCalls()
   throws SQLException;
boolean supportsCatalogsInTableDefinitions()
   throws SQLException;
boolean supportsColumnAliasing() throws SQLException;
boolean supportsConvert() throws SQLException;
boolean supportsConvert(int fromType, int toType)
   throws SQLException;
boolean supportsCoreSQLGrammar() throws SQLException;
boolean supportsCorrelatedSubqueries()
   throws SQLException;
boolean
supportsDataDefinitionAndDataManipulationTransactions()
   throws SQLException;
boolean supportsDataManipulationTransactionsOnly()
   throws SQLException;
boolean supportsDifferentTableCorrelationNames()
   throws SQLException;
boolean supportsExpressionsInOrderBy()
   throws SQLException;
boolean supportsExtendedSQLGrammar()
   throws SQLException;
boolean supportsFullOuterJoins() throws SQLException;
boolean supportsGroupBy() throws SQLException;
boolean supportsGroupByBeyondSelect()
   throws SQLException;
boolean supportsGroupByUnrelated()
   throws SQLException;
boolean supportsIntegrityEnhancementFacility()
   throws SQLException;
boolean supportsLikeEscapeClause()
   throws SQLException;
boolean supportsLimitedOuterJoins()
   throws SQLException;
```

```
boolean supportsMinimumSQLGrammar()
   throws SQLException;
boolean supportsMixedCaseIdentifiers()
   throws SQLException;
boolean supportsMixedCaseQuotedIdenfitiers()
   throws SQLException;
boolean supportsMultipleResultSets()
   throws SQLException;
boolean supportsMultipleTransactions()
   throws SQLException;
boolean supportsNonNullableColumns()
   throws SQLException;
boolean supportsOpenCursorsAcrossCommit()
   throws SQLException;
boolean supportsOpenCursorsAcrossRollback()
   throws SQLException;
boolean supportsOpenStatementsAcrossCommit()
   throws SQLException;
boolean supportsOpenStatementsAcrossRollback()
   throws SQLException;
boolean supportsOrderByUnrelated()
   throws SQLException;
boolean supportsOuterJoins() throws SQLException;
boolean supportsPositionedDelete()
   throws SQLException;
boolean supportsPositionedUpdate()
   throws SQLException;
boolean supportsSchemasInDataManipulation()
   throws SQLException;
boolean supportsSchemasInIndexDefinitions()
   throws SQLException;
boolean supportsSchemasInPrivelegeDefinitions()
   throws SQLException;
boolean supportsSchemasInProcedureCalls()
   throws SQLException;
boolean supportsSchemasInTableDefinitions()
   throws SQLException;
boolean supportsSelectForUpdate()
   throws SQLException;
boolean supportsStoredProcedures()
   throws SQLException;
boolean supportsSubqueriesInComparisons()
   throws SQLException;
boolean supportsSubqueriesInExists()
   throws SQLException;
boolean supportsSubqueriesInIns()
   throws SQLException;
boolean supportsSubqueriesInQuantifieds()
   throws SQLException;
boolean supportsTableCorrelationNames()
   throws SQLException;
boolean supportsTransactionIsolationLevel(int level)
   throws SQLException;
boolean supportsTransactions() throws SQLException;
```

```
    boolean supportsUnion() throws SQLException;
    boolean supportsUnionAll() throws SQLException;
    boolean usesLocalFilePerTable()
        throws SQLException;
    boolean usesLocalFiles() throws SQLException;
}
```

Date

Synopsis

Class Name: java.sql.Date
Superclass: java.util.Date
Immediate Subclasses: None
Interfaces Implemented: None
Availability: JDK 1.1

Description

This class deals with a subset of functionality found in the java.util.Date class. It specifically worries only about days and ignores hours, minutes, and seconds.

Class Summary

```
public class Date extends java.util.Date {
    static public Date valueOf(String s);
    #public Date(int year, int month, int day);
    public Date(long date);
    public void setTime(long date);
    public String toString();
}
```

Class Methods

valueOf( )

```
static public Date valueOf(String s)
```

Description: Given a String in the form of yyyy-mm-dd, this will return a corresponding instance of the Date class representing that date.

Object Constructors

Date( )

```
public Date(long date)
#public Date(int year, int month, int day)
```

Description: Constructs a new Date instance. The proper way to construct a Date is to use the new JDK 1.2 Date(long) constructor. The date argument specifies the number of milliseconds since 1 January 1970 00:00:00 GMT. A negative number represents the milliseconds before that date. The second, deprecated constructor naturally should never be used since it is ambiguous with respect to calendar and time zone.

Object Methods

setTime( )

```
public void setTime(long date)
```

Description: Sets the time represented by this Date object to the specified number of milliseconds since 1 January 1970 00:00:00 GMT. A negative number represents the milliseconds before that date.

toString( )

```
public String toString()
```

Description: Provides a String representing this Date in the form yyyy-mm-dd.

Driver

Synopsis

Class Name: java.sql.Driver
Superclass: None
Immediate Subclasses: None
Interfaces Implemented: None
Availability: JDK 1.1

Description

This class represents a specific JDBC implementation. When a Driver is loaded, it should create an instance of itself and register that instance with the DriverManager class. This allows applications to create instances of it using the Class.forName() call to load a driver.

The Driver object then provides the ability for an application to connect to one or more databases. When a request for a specific database comes through, the DriverManager will pass the data source request to each Driver registered as a URL. The first Driver to connect to the data source using that URL will be used.

Class Summary

```
public interface Driver {
  boolean acceptsURL(String url) throws SQLException;
  Connection connect(String url, Properties info)
    throws SQLException;
  int getMajorVersion();
  int getMinorVersion();
  DriverPropertyInfo[] getPropertyInfo(String url,
                                       Properties info)
    throws SQLException;
  boolean jdbcCompliant();
}
```

Object Methods

acceptsURL( )

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

Description: Returns true if the specified URL matches the URL subprotocol used by this driver.

connect( )

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

Description: This method attempts a connect using the specified URL and Property information (usually containing the user name and password). If the URL is not right for this driver, connect() simply returns null. If it is the right URL, but an error occurs during the connection process, an SQLException should be thrown.

getMajorVersion( )

```
public int getMajorVersion()
```

Description: Returns the major version number for the driver.

getMinorVersion( )

```
public int getMinorVersion()
```

Description: Returns the minor version number for the driver.

getPropertyInfo( )

```
public DriverPropertyInfo[] getPropertyInfo(String url,
                                            Properties info)
  throws SQLException;
```

Description: This method allows GUI-based RAD environments to find out which properties the driver needs on connect so that it can prompt a user to enter values for those properties.

jdbcCompliant( )

```
public boolean jdbcCompliant()
```

Description: A Driver can return true here only if it passes the JDBC compliance tests. This means that the driver implementation supports the full JDBC API and full SQL 92 Entry Level.

DriverManager

Synopsis

Class Name: java.sql.DriverManager
Superclass: java.lang.Object
Immediate Subclasses: None
Interfaces Implemented: None
Availability: JDK 1.1

Description

The DriverManager holds the master list of registered JDBC drivers for the system. Upon initialization, it loads all classes specified in the jdbc.drivers property. You can thus specify any runtime information about the database being used by an application on the command line.

During program execution, other drivers may register themselves with the DriverManager by calling the registerDriver() method. The DriverManager uses a JDBC URL to find an application's desired driver choice when requests are made through getConnection().

The DriverManager class is likely to disappear one day as the new JDBC 2.0 Standard Extension provides a much more application-friendly way of getting a database connection.

Class Summary

```
public class DriverManager {
  static void deregisterDriver(Driver driver)
    throws SQLException;
  static public synchronized Connection getConnection(String url,
    Properties info) throws SQLException;
  static public synchronized Connection getConnection(String url,
    String user, String password) throws SQLException;
  static public synchronized Connection getConnection(String url)
    throws SQLException;
  static public Driver getDriver(String url) throws SQLException;
  static public Enumeration getDrivers();
  static public int getLoginTimeout();
  #static public PrintStream getLogStream();
  static public PrintWriter getLogWriter();

  static public void println(String message);
  static public synchronized void registerDriver(Driver driver)
      throws SQLException;
  #static public void setLogStream(PrintStream out);
  static public void setLogWriter(PrintWriter out);
  static public void setLoginTimeout(int seconds);
}
```

Class Methods

deregisterDriver()

```
static public void deregisterDriver(Driver driver) throws SQLException
```

Description: Removes a Driver from the list of registered drivers.

getConnection()

```
static public synchronized Connection getConnection(String url,
    Properties info) throws SQLException
static public synchronized Connection getConnection(String url,
    String user, String password) throws SQLException
static public synchronized Connection getConnection(String url)
```

```
    throws SQLException
```

Description: Establishes a connection to the data store represented by the URL given. The DriverManager then looks through its list of registered Driver instances for one that will handle the specified URL. If none is found, it throws an SQLException. Otherwise it returns the Connection instance from the connect() method in the Driver class.

getDriver( )

```
static public Driver getDriver(String url) throws SQLException
```

Description: Returns a driver than can handle the specified URL.

getDrivers( )

```
static public Enumeration getDrivers()
```

Description: Returns a list of all registered drivers.

getLoginTimeout( ) and setLoginTimeout( )

```
static public int getLoginTimeout()
static public int setLoginTimeout()
```

Description: The login timeout is the maximum time in seconds that a driver can wait in attempting to log in to a database.

getLogStream( ) and setLogStream( )

```
#static public PrintStream getLogStream()
#static public void setLogStream(PrintStream out)
static public PrintWriter getLogWriter()
static public void setLogWriter(PrintWriter out)
```

Description: Sets the stream used by the DriverManager and all drivers. The LogStream variant is the old JDK 1.1 version and should be avoided in favor of log writers

println( )

```
static public void println(String message)
```

Description: Prints a message to the current log stream.

registerDriver( )

```
static public synchronized void registerDriver(Driver driver)
    throws SQLException
```

Description: This method allows a newly loaded Driver to register itself with the DriverManager class.

DriverPropertyInfo

Synopsis

Class Name: java.sql.DriverPropertyInfo
Superclass: java.lang.Object
Immediate Subclasses: None
Interfaces Implemented: None
Availability: JDK 1.1

Description

This class provides information required by a driver in order to connect to a database. Only development tools are likely ever to require this class. It has no methods, simply a list of public attributes.

Class Summary

```
public class DriverPropertyInfo {
   public String[] choices;
   public String description;
   public String name;
   public boolean required;
   public String value;
   public DriverPropertyInfo(String name, String value);
}
```

Object Attributes

choices

```
public String[] choices
```

Description: A list of choices from which a user may be prompted to specify a value for this property. This value can be null.

description

```
public String description
```

Description: A brief description of the property or null.

name

```
public String name
```

Description: The name of the property.

required

```
public boolean required
```

Description: Indicates whether or not this property must be set in order to make a connection.

value

```
public String value
```

Description: The current value of the property or null if no current value is set.

Object Constructors

DriverPropertyInfo( )

public DriverPropertyInfo(String name, String value)

Description: Constructs a new DriverPropertyInfo object with the name and value attributes set to the specified parameters. All other values are set to their default values.

PreparedStatement

Synopsis

Class Name: java.sql.PreparedStatement
Superclass: java.sql.Statement
Immediate Subclasses: java.sql.CallableStatement
Interfaces Implemented: None
Availability: JDK 1.1

Description

This class represents a precompiled SQL statement.

Class Summary

```
public interface PreparedStatement extends Statement {
  void addBatch() throws SQLException;
  void clearParameters() throws SQLException;
  boolean execute() throws SQLException;
  ResultSet executeQuery() throws SQLException;
  int executeUpdate() throws SQLException;
  ResultSetMetaData getMetaData() throws SQLException;
  void setArray(int index, Array arr)
    throws SQLException;
  void setAsciiStream(int index, InputStream is,
    int length) throws SQLException;
  void setBigDecimal(int index, BigDecimal d)
    throws SQLException;
  void setBinaryStream(int index, InputStream is,
    int length) throws SQLException;
  void setBlob(int index, Blob b) throws SQLException;
  void setBoolean(int index, boolean b)
    throws SQLException;
  void setByte(int index, byte b) throws SQLException;
  void setBytes(int index, byte[] bts)
    throws SQLException;
  void setCharacterStream(int index, Reader rdr,
    int length) throws SQLException;
  void setClob(int index, Clob c) throws SQLException;
  void setDate(int index, Date d) throws SQLException;
  void setDate(int index, Date d, Calendar cal)
    throws SQLException;
  void setDouble(int index, double x)
```

```
        throws SQLException;
    void setFloat(int index, float f) throws SQLException;
    void setInt(int index, int x) throws SQLException;
    void setLong(int index, long x) throws SQLException;
    void setNull(int index, int type) throws SQLException;
    void setNull(int index, int type, String tname)
        throws SQLException;
    void setObject(int index, Object ob)
        throws SQLException;
    void setObject(int index, Object ob, int type)
        throws SQLException;
    void setObject(int index, Object ob, int type,
        int scale) throws SQLException;
    void setRef(int index, Ref ref) throws SQLException;
    void setShort(int index, short s) throws SQLException;
    void setString(int index, String str)
        throws SQLException;
    void setTime(int index, Time t) throws SQLException;
    void setTime(int index, Time t, Calendar cal)
        throws SQLException;
    void setTimestamp(int index, Timestamp ts)
        throws SQLException;
    void setTimestamp(int index, Timestamp ts, Calendar cal)
        throws SQLException;
    #void setUnicodeStream(int index, InputStream is,
        int length) throws SQLException;
}
```

Object Methods

addBatch( )

```
public void addBatch() throws SQLException
```

Description: Adds a set of parameters to the batch for batch processing.

clearParameters( )

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

Description: Once set, a parameter value remains bound until either a new value is set for the parameter or until clearParameters() is called. This method clears all parameters associated with the PreparedStatement.

execute( ), executeQuery( ), and executeUpdate( )

```
public abstract boolean execute() throws SQLException
public abstract ResultSet executeQuery() throws SQLException
public abstract int executeUpdate() throws SQLException
```

Description: Executes the PreparedStatement. The first method, execute(), allows you to execute the PreparedStatement when you do not know if it is a query or an update. It returns true if the statement has result sets to process.

The executeQuery() method is used for executing queries. It returns a result set for processing.

The executeUpdate() statement is used for executing updates. It returns the number of rows affected by the update.

getMetaData( )

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

Description: Retrieves the number, types, and properties of a ResultSet's columns.

setArray( ), setAsciiStream( ), setBigDecimal( ), setBinaryStream( ), setBlob( ), setBoolean( ), setByte( ), setBytes( ), setCharacterStream( ), setClob( ), setDate( ), setDouble( ), setFloat( ), setInt( ), setLong( ), setNull( ), setObject( ), setRef( ), setShort( ), setString( ), setTime( ), setTimestamp( ), and setUnicodeStream( )

```
public void setArray(int index, Array arr)
  throws SQLException
public void setAsciiStream(int index, InputStream is,
  int length) throws SQLException
public void setBigDecimal(int index, BigDecimal d)
  throws SQLException
public void setBinaryStream(int index, InputStream is,
  int length) throws SQLException
public void setBlob(int index, Blob b)
  throws SQLException
public void setBoolean(int index, boolean b)
  throws SQLException
public void setByte(int index, byte b)
  throws SQLException
public void setBytes(int index, byte[] bts)
  throws SQLException
public void setCharacterStream(int index, Reader rdr,
  int length) throws SQLException
public void setClob(int index, Clob c)
  throws SQLException
public void setDate(int index, Date d)
  throws SQLException
public void setDate(int index, Date d, Calendar cal)
  throws SQLException
public void setDouble(int index, double d)
  throws SQLException
public void setFloat(int index, float f)
  throws SQLException
public void setInt(int index, int x)
  throws SQLException
public void setLong(int index, long x)
  throws SQLException
public void setNull(int index, int type)
  throws SQLException
public void setNull(int index, int type, String tname)
  throws SQLException
public void setObject(int index, Object ob)
  throws SQLException
public void setObject(int index, Object ob, int type)
  throws SQLException
```

```
public void setObject(int index, Object ob, int type,
  int scale) throws SQLException
public void setRef(int index, Ref ref)
  throws SQLException
public void setShort(int index, short s)
  throws SQLException
public void setString(int index, String str)
  throws SQLException
public void setTime(int index, Time t)
  throws SQLException
public void setTime(int index, Time t, Calendar cal)
  throws SQLException
public void setTimestamp(int index, Timestamp ts)
  throws SQLException
public void setTimestamp(int index, Timestamp ts,
  Calendar cal) throws SQLException
#public void setUnicodeStream(int index, InputStream is,
  int length) throws SQLException
```

Description: Binds a value to the specified parameter.

Ref

Synopsis

Class Name: java.sql.Ref
Superclass: None
Immediate Subclasses: None
Interfaces Implemented: None
Availability: New as of JDK 1.2

Description

A Ref is a reference to a value of an SQL structured type in the database. You can dereference a Ref by passing it as a parameter to an SQL statement and executing the statement.

Class Summary

```
public interface Ref {
  String getBaseTypeName() throws SQLException;
}
```

Object Methods

getBaseTypeName( )

```
public String getBaseTypeName() throws SQLException
```

Description: Provides the SQL structured type name for the referenced item.

ResultSet

Synopsis

Class Name: java.sql.ResultSet
Superclass: None
Immediate Subclasses: None
Interfaces Implemented: None
Availability: JDK 1.1

Description

This class represents a database result set. It provides an application with access to database queries one row at a time. During query processing, a ResultSet maintains a pointer to the current row being manipulated. The application then moves through the results sequentially until all results have been processed or the ResultSet is closed. A ResultSet is automatically closed when the Statement that generated it is closed, reexecuted, or used to retrieve the next ResultSet in a multiple result set query.

Class Summary

```
public interface ResultSet {
  static public final int CONCUR_READ_ONLY;
  static public final int CONCUR_UPDATABLE;
  static public final int FETCH_FORWARD;
  static public final int FETCH_REVERSE;
  static public final int FETCH_UNKNOWN;
  static public final int TYPE_FORWARD_ONLY;
  static public final int TYPE_SCROLL_INSENSITIVE;
  static public final int TYPE_SCROLL_SENSITIVE;
  boolean absolute(int row) throws SQLException;
  void afterLast() throws SQLException;
  void beforeFirst() throws SQLException;
  void cancelRowUpdates() throws SQLException;
  void clearWarnings() throws SQLException;
  void close() throws SQLException;
  void deleteRow() throws SQLException;
  int findColumn(String cname) throws SQLException;
  boolean first() throws SQLException;
  Array getArray(int index) throws SQLException;
  Array getArray(String cname) throws SQLException;
  InputStream getAsciiStream(int index)
    throws SQLException;
  InputStream getAsciiStream(String cname)
    throws SQLException;
  InputStream getBinaryStream(int index)
    throws SQLException;
  InputStream getBinaryStream(String cname)
    throws SQLException;
  BigDecimal getBigDecimal(int index)
    throws SQLException;
```

```
#BigDecimal getBigDecimal(int index, int scale)
   throws SQLException;
BigDecimal getBigDecimal(String cname)
   throws SQLException;
#BigDecimal getBigDecimal(String cname, int scale)
   throws SQLException;
InputStream getBinaryStream(int index)
   throws SQLException;
InputStream getBinaryStream(String cname)
   throws SQLException;
Blob getBlob(int index) throws SQLException;
Blob getBlob(String cname) throws SQLException;
boolean getBoolean(int index) throws SQLException;
boolean getBoolean(String cname) throws SQLException;
byte getByte(int index) throws SQLException;
byte getByte(String cname) throws SQLException;
byte[] getBytes(int index) throws SQLException;
byte[] getBytes(String cname) throws SQLException;
Reader getCharacterStream(int index)
   throws SQLException;
Reader getCharacterStream(String cname)
   throws SQLException;
Clob getClob(int index) throws SQLException;
Clob getClob(String cname) throws SQLException;
int getConcurrency() throws SQLException;
String getCursorName() throws SQLException;
Date getDate(int index) throws SQLException;
Date getDate(int index, Calendar cal)
   throws SQLException;
Date getDate(String cname) throws SQLException;
Date getDate(String cname, Calendar cal)
   throws SQLException;
double getDouble(int index) throws SQLException;
double getDouble(String cname) throws SQLException;
int getFetchDirection() throws SQLException;
int getFetchSize() throws SQLException;
float getFloat(int index) throws SQLException;
float getFloat(String cname) throws SQLException;
int getInt(int index) throws SQLException;
int getInt(String cname) throws SQLException;
long getLong(int index) throws SQLException;
long getLong(String cname) throws SQLException;
ResultSetMetaData getMetaData() throws SQLException;
Object getObject(int index) throws SQLException;
Object getObject(int index, Map map)
   throws SQLException;
Object getObject(String cname) throws SQLException;
Object getObject(String cname, Map map)
   throws SQLException;
Ref getRef(int index) throws SQLException;
Ref getRef(String cname) throws SQLException;
int getRow() throws SQLException;
short getShort(int index) throws SQLException;
short getShort(String cname) throws SQLException;
```

```
Statement getStatement() throws SQLException;
String getString(int index) throws SQLException;
String getString(String cname) throws SQLException;
Time getTime(int index) throws SQLException;
Time getTime(int index, Calendar cal)
   throws SQLException;
Time getTime(String cname) throws SQLException;
Time getTime(String cname, Calendar cal)
   throws SQLException;
Timestamp getTimestamp(int index) throws SQLException;
Timestamp getTimestamp(int index, Calendar cal)
   throws SQLException;
Timestamp getTimestamp(String cname) throws SQLException;
Timestamp getTimestamp(String cname, Calendar cal)
   throws SQLException;
int getType() throws SQLException;
#InputStream getUnicodeStream(int index)
   throws SQLException;
#InputStream getUnicodeStream(String cname)
   throws SQLException;
SQLWarning getWarnings() throws SQLException;
void insertRow() throws SQLException;
boolean isAfterLast() throws SQLException;
boolean isBeforeFirst() throws SQLException;
boolean isFirst() throws SQLException;
boolean isLast() throws SQLException;
boolean last() throws SQLException;
void moveToCurrentRow() throws SQLException;
void moveToInsertRow() throws SQLException;
boolean next() throws SQLException;
boolean previous() throws SQLException;
void refreshRow() throws SQLException;
boolean relative(int rows) throws SQLException;
boolean rowDeleted() throws SQLException;
boolean rowInserted() throws SQLException;
boolean rowUpdated() throws SQLException;
void setFetchDirection(int dir) throws SQLException;
void setFetchSize(int rows) throws SQLException;
void updateAsciiStream(int index, InputStream is,
   int length) throws SQLException;
void updateAsciiStream(String cname, InputStream is,
   int length) throws SQLException;
void updateBigDecimal(int index, BigDecimal d)
   throws SQLException;
void updateBigDecimal(String cname, BigDecimal d)
   throws SQLException;
void updateBinaryStream(int index, InputStream is)
   throws SQLException;
void updateBinaryStream(String cname, InputStream is)
   throws SQLException;
void updateBoolean(int index, boolean b)
   throws SQLException;
void updateBoolean(String cname, boolean b)
   throws SQLException;
```

```
void updateByte(int index, byte b)
   throws SQLException;
void updateByte(String cname, byte b)
   throws SQLException;
void updateBytes(int index, byte[] bts)
   throws SQLException;
void updateBytes(String cname, byte[] bts)
   throws SQLException;
void updateCharacterStream(int index, Reader rdr,
   int length) throws SQLException;
void updateCharacterStream(String cname, Reader rdr,
   int length) throws SQLException;
void updateDate(int index, Date d)
   throws SQLException;
void updateDate(String cname, Date d)
   throws SQLException;
void updateDouble(int index, double d)
   throws SQLException;
void updateDouble(String cname, double d)
   throws SQLException;
void updateFloat(int index, float f)
   throws SQLException;
void updateFloat(String cname, float f)
   throws SQLException;
void updateInt(int index, int x) throws SQLException;
void updateInt(String cname, int x)
   throws SQLException;
void updateLong(int index, long x)
   throws SQLException;
void updateLong(String cname, long x)
   throws SQLException;
void updateNull(int index) throws SQLException;
void updateNull(String cname) throws SQLException;
void updateObject(int index, Object ob)
   throws SQLException;
void updateObject(int index, Object ob, int scale)
void updateObject(String cname, Object ob)
   throws SQLException;
void updateObject(String cname, Object ob, int scale)
   throws SQLException;
void updateRow() throws SQLException;
void updateShort(int index, short s)
   throws SQLException;
void updateShort(String cname, short s)
   throws SQLException;
void updateString(int index, String str)
   throws SQLException;
void updateString(String cname, String str)
   throws SQLException;
void updateTime(int index, Time t)
   throws SQLException;
void updateTime(String cname, Time t)
   throws SQLException;
void updateTimestamp(int index, Timestamp ts)
```

```
      throws SQLException;
    void updateTimestamp(String cname, Timestamp ts)
      throws SQLException;
    boolean wasNull() throws SQLException;
}
```

Class Attributes

CONCUR_READ_ONLY

```
static public final int CONCUR_READ_ONLY
```

Description: The concurrency mode that specifies that a result set may not be updated.

CONCUR_UPDATABLE

```
static public final int CONCUR_UPDATABLE
```

Description: The concurrency mode that specifies that a result set is updatable.

FETCH_FORWARD

```
static public final int FETCH_FORWARD
```

Description: This value specifies that a result set's fetch direction is in the forward direction, from first to last.

FETCH_REVERSE

```
static public final int FETCH_REVERSE
```

Description: This value specifies that a result set's fetch direction is in the reverse direction, from last to first.

FETCH_UNKNOWN

```
static public final int FETCH_UNKNOWN
```

Description: This value specifies that the order of result set processing is unknown.

TYPE_FORWARD_ONLY

```
static public final int TYPE_FORWARD_ONLY
```

Description: This result set type specifies that a result set can only be navigated in the forward direction.

TYPE_SCROLL_INSENSITIVE

```
static public final int TYPE_SCROLL_INSENSITIVE
```

Description: This result set type specifies that a result set may be navigated in any direction, but it is not sensitive to changes made by others.

TYPE_SCROLL_SENSITIVE

```
static public final int TYPE_SCROLL_SENSITIVE
```

Description: This result set type specifies that a result set may be navigated in any direction and that changes made by others will be seen in the result set.

Object Methods

absolute( )

public boolean absolute(int row) throws SQLException

Description: This method moves the cursor to the specified row number starting from the beginning for a positive number or from the end for a negative number.

afterLast( )

public void afterLast() throws SQLException

Description: This method moves the cursor to the end of the result set, after the last row.

beforeFirst( )

public void beforeFirst() throws SQLException

Description: Moves the cursor to the beginning of the result set, before the first row.

cancelRowUpdates( )

public void cancelRowUpdates() throws SQLException

Description: Cancels any updates made to this row.

clearWarnings( )

public void clearWarnings() throws SQLException

Description: Clears all warnings from the SQLWarning chain. Subsequent calls to getWarnings() then returns null until another warning occurs.

close( )

public void close() throws SQLException

Description: Performs an immediate, manual close of the ResultSet. This is generally never required, as the closure of the Statement associated with the ResultSet will automatically close the ResultSet.

deleteRow( )

public void deleteRow() throws SQLException

Description: Deletes the current row from this result set and from the database.

findColumn( )

public int findColumn(String cname) throws SQLException

Description: For the specified column name, this method will return the column number associated with it.

first( )

public boolean first() throws SQLException

Description: Moves the cursor to the first row of a result set.

getAsciiStream( ), getBinaryStream( ), getCharacterStream( ), and getUnicodeStream( )

```
public InputStream getAsciiStream(int index)
  throws SQLException
public InputStream getAsciiStream(String cname)
  throws SQLException
public InputStream getBinaryStream(int index)
  throws SQLException
public InputStream getBinaryStream(String cname)
  throws SQLException
public Reader getCharacterStream(int index)
  throws SQLException
public Reader getCharacterStream(String cname)
  throws SQLException
#public InputStream getUnicodeStream(int index)
  throws SQLException
#public InputStream getUnicodeStream(String cname)
  throws SQLException
```

Description: In some cases, it may make sense to retrieve large pieces of data from the database as a Java InputStream. These methods allow an application to retrieve the specified column from the current row in this manner. You should notice that the getUnicodeStream() method has been deprecated in favor of the new getCharacterStream() method.

getArray( ), getBlob( ), getBoolean( ), getByte( ), getBytes( ), getClob( ), getDate( ), getDouble( ), getFloat( ), getInt( ), getLong( ), getRef( ), getShort( ), getString( ), getTime( ), and getTimestamp( )

```
public Array getArray(int index) throws SQLException
public Array getArray(String cname) throws SQLException
public Blob getBlob(int index) throws SQLException
public Blob getBlob(String cname) throws SQLException
public boolean getBoolean(int index) throws SQLException
public boolean getBoolean(String cname) throws SQLException
public byte getByte(int index) throws SQLException
public byte getByte(String cname) throws SQLException
public byte[] getBytes(int index) throws SQLException
public byte[] getBytes(String cname) throws SQLException
public Clob getClob(int index) throws SQLException
public Clob getClob(String cname) throws SQLException
public Date getDate(int index) throws SQLException
public Date getDate(String cname) throws SQLException
public double getDouble(int index) throws SQLException
public double getDouble(String cname) throws SQLException
public float getFloat(int index) throws SQLException
public float getFloat(String cname) throws SQLException
public int getInt(int index) throws SQLException
public int getInt(String cname) throws SQLException
public long getLong(int index) throws SQLException
public long getLong(String cname) throws SQLException
public Ref getRef(int index) throws SQLException
public Ref getRef(String cname) throws SQLException
public short getShort(int index) throws SQLException
public short getShort(String cname) throws SQLException
public String getString(int index) throws SQLException
public String getString(String cname) throws SQLException
public Time getTime(int index) throws SQLException
public Time getTime(String cname) throws SQLException
public Timestamp getTimestamp(int index)
  throws SQLException
public Timestamp getTimestamp(String cname)
  throws SQLException
```

Description: These methods return the specified column value for the current row as the Java datatype that matches the method name.

getConcurrency( ), and setConcurrency( )

```
public int getConcurrency() throws SQLException
```

Description: These methods access the result set concurrency mode. It initially takes its value from the statement that generated this result set.

getCursorName( )

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

Description: Because some databases allow positioned updates, an application needs the cursor name associated with a ResultSet in order to perform those positioned updates. This method provides the cursor name.

getMetaData( )

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

Description: Provides the meta-data object for this ResultSet.

getFetchDirection( ), setFetchDirection( ), getFetchSize( ), and setFetchSize( )

```
public int getFetchDirection() throws SQLException
public void setFetchDirection(int dir) throws SQLException
public int getFetchSize() throws SQLException
public void setFetchSize(int rows) throws SQLException
```

Description: These methods provide optimization hints for the driver. The driver is free to ignore these hints. The fetch size is the suggested number of rows the driver should prefetch for each time it grabs data from the database. The direction is a hint to the driver about the direction in which you intend to work.

getObject( )

```
public Object getObject(int index) throws SQLException
public Object getObject(int index, Map map)
  throws SQLException
public Object getObject(String cname) throws SQLException
public Object getObject(String cname, Map map)
  throws SQLException
```

Description: Returns the specified column value for the current row as a Java object. The type returned will be the Java object that most closely matches the SQL type for the column. It is also useful for columns with database-specific datatypes.

getRow( )

public int getRow() throws SQLException

Description: Returns the current row number.

getStatement( )

public Statement getStatement() throws SQLException

Description: Returns the Statement instance that generated this result set.

getType( )

public int getType() throws SQLException

Description: Returns the result set type for this result set.

getWarnings( )

public SQLWarning getWarnings() throws SQLException

Description: Returns the first SQLWarning object in the warning chain.

insertRow( )

public void insertRow() throws SQLException

Description: Inserts the contents of the insert row into the result set and into the database.

isAfterLast( )

public boolean isAfterLast() throws SQLException

Description: Returns true if this result set is positioned after the last row in the result set.

isBeforeLast( )

public boolean isBeforeFirst() throws SQLException

Description: Returns true if this result set is positioned before the first row in the result set.

isFirst( )

public boolean isFirst() throws SQLException

Description: Returns true if the result set is positioned on the first row of the result set.

isLast( )

public boolean isLast() throws SQLException

Description: Returns true if result set is positioned after the last row in the result set.

last( )

public boolean last() throws SQLException

Description: Moves the cursor to the last row in the result set.

moveToCurrentRow( )

public void moveToCurrentRow() throws SQLException

Description: Moves the result set to the current row. This is used after you are done inserting a row.

moveToInsertRow( )

```
public void moveToInsertRow() throws SQLException
```

Description: Moves the result to a new insert row. You need to call moveToCurrentRow() to get back.

next( ) and previous( )

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

Description: These methods navigate one row forward or one row backward in the ResultSet. Under a newly created result set, the result set is positioned before the first row. The first call to next() would thus move the result set to the first row. These methods return true as long as there is a row to move to. If there are no further rows to process, it returns false. If an InputStream from the previous row is still open, it is closed. The SQLWarning chain is also cleared.

refreshRow( )

```
public void refreshRow() throws SQLException
```

Description: Refreshes the current row with its most recent value from the database.

relative( )

```
public boolean relative(int rows) throws SQLException
```

Description: Moves the cursor the specified number of rows forwards or backwards. A positive number indicates that the cursor should be moved forwards and a negative number indicates it should be moved backwards.

rowDeleted( ), rowInserted( ), and rowUpdated( )

```
public boolean rowDeleted() throws SQLException
public boolean rowInserted() throws SQLException
public boolean rowUpdated() throws SQLException
```

Description: Returns true if the current row has been deleted, inserted, or updated.

updateAsciiStream( ), updateBigDecimal( ), updateBinaryStream( ), updateBoolean( ), updateByte( ), updateBytes( ), updateCharacterStream( ), updateDate( ), updateDouble( ), updateFloat( ), updateInt( ), updateLong( ), updateNull( ), updateObject( ), updateShort( ), updateString( ), updateTime( ), and updateTimestamp( )

```
public void updateAsciiStream(int index, InputStream is,
  int length) throws SQLException
public void updateAsciiStream(String cname, InputStream is,
  int length) throws SQLException
public void updateBigDecimal(int index, BigDecimal d)
  throws SQLException
public void updateBigDecimal(String cname, BigDecimal d)
  throws SQLException
```

```
public void updateBinaryStream(int index, InputStream is)
   throws SQLException
public void updateBinaryStream(String cname, InputStream is)
   throws SQLException
public void updateBoolean(int index, boolean b)
   throws SQLException
public void updateBoolean(String cname, boolean b)
   throws SQLException
public void updateByte(int index, byte b)
   throws SQLException
public void updateByte(String cname, byte b)
   throws SQLException
public void updateBytes(int index, byte[] bts)
   throws SQLException
public void updateBytes(String cname, byte[] bts)
   throws SQLException
public void updateCharacterStream(int index, Reader rdr,
   int length) throws SQLException
public void updateCharacterStream(String cname, Reader rdr,
   int length) throws SQLException
public void updateDate(int index, Date d)
   throws SQLException
public void updateDate(String cname, Date d)
   throws SQLException
public void updateDouble(int index, double d)
   throws SQLException
public void updateDouble(String cname, double d)
   throws SQLException
public void updateFloat(int index, float f)
   throws SQLException
public void updateFloat(String cname, float f)
   throws SQLException
public void updateInt(int index, int x)
   throws SQLException
public void updateInt(String cname, int x)
   throws SQLException
public void updateLong(int index, long x)
   throws SQLException
public void updateLong(String cname, long x)
   throws SQLException
public void updateNull(int index) throws SQLException
public void updateNull(String cname) throws SQLException
public void updateObject(int index, Object ob)
   throws SQLException
public void updateObject(int index, Object ob, int scale)
   throws SQLException
public void updateObject(String cname, Object ob)
   throws SQLException
public void updateObject(String cname, Object ob, int scale)
   throws SQLException
public void updateShort(int index, short s)
   throws SQLException
public void updateShort(String cname, short s)
   throws SQLException
```

```
public void updateString(int index, String str)
   throws SQLException
public void updateString(String cname, String str)
   throws SQLException
public void updateTime(int index, Time t)
   throws SQLException
public void updateTime(String cname, Time t)
   throws SQLException
public void updateTimestamp(int index, Timestamp ts)
   throws SQLException
public void updateTimestamp(String cname, Timestamp ts)
   throws SQLException
```

Description: These methods update column by column in the current row of your result set as long as your result set supports updating. Once you are done modifying the row, you can call `insertRow()` or `updateRow()` to save the changes to the database.

updateRow( )

```
public void updateRow() throws SQLException
```

Description: Updates any changes made to the current row to the database.

wasNull( )

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

Description: This method returns `true` if the last column read was `null`; otherwise it returns `false`.

ResultSetMetaData

Synopsis

Class Name:`java.sql.ResultSetMetaData`
Superclass:None
Immediate Subclasses:None
Interfaces Implemented:None
Availability:JDK 1.1

Description

This class provides meta-information about the types and properties of the columns in a `ResultSet` instance.

Class Summary

```
public interface ResultSetMetaData {
   static public final int columnNoNulls;
   static public final int columnNullable;
   static public final int columnNullableUnknown;
   String getCatalogName(int index)
      throws SQLException;
   String getColumnClassName(int index)
```

```
        throws SQLException;
    public int getColumnCount() throws SQLException;
    public int getColumnDisplaySize(int index)
        throws SQLException;
    public String getColumnLabel(int index)
        throws SQLException;
    public String getColumnName(int index)
        throws SQLException;
    public int getColumnType(int index) throws SQLException;
    public String getColumnTypeName(int index)
        throws SQLException;
    public int getPrecision(int index) throws SQLException;
    public int getScale(int index) throws SQLException;
    public String getSchemaName(int index)
        throws SQLException;
    public String getTableName(int index)
        throws SQLException;
    public boolean isAutoIncrement(int index)
        throws SQLException;
    public isCaseSensitive(int index)
        throws SQLException;
    public boolean isCurrency(int index)
        throws SQLException;
    public boolean isDefinitelyWritable(int index)
        throws SQLException;
    public int isNullable(int index) throws SQLException;
    public boolean isReadOnly(int index)
        throws SQLException;
    public boolean isSearchable(int index)
        throws SQLException;
    public boolean isSigned(int index) throws SQLException;
    public boolean isWritable(int index)
        throws SQLException;
}
```

Class Attributes

columnNoNulls

```
static public final int columnNoNulls
```

Description: The column in question does not allow NULL values.

columnNullable

```
static public final int columnNullable
```

Description: The column in question allows NULL values.

columnNullableUnknown

```
static public final int columnNullableUnknown
```

Description: It is not known if the column in question can accept NULL values.

Object Methods

getCatalogName( )

```
public String getCatalogName(int index) throws SQLException
```

Description: Provides the catalog name associated with the specified column's table.

getColumnClassName( )

```
public String getColumnClassName(int index)
  throws SQLException
```

Description: Provides the fully-qualified name of the Java class that will be instantiated by a call to ResultSet.getObject() for this column.

getColumnCount( )

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

Description: Returns the number of columns in the result set.

getColumnDisplaySize( )

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

Description: Returns the maximum width for displaying the column's values.

getColumnLabel( )

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

Description: Returns the display name for the column.

getColumnName( )

```
public String getcname(int column) throws SQLException
```

Description: Returns the database name for the column.

getColumnType( )

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

Description: Returns the SQL type for the specified column as a value from java.sql.Types.

getColumnTypeName( )

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

Description: Returns the name of the SQL type for the specified column.

getPrecision( )

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

Description: Returns the number of decimal digits for the specified column.

getScale( )

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

Description: Returns the number of digits to the right of the decimal for this column.

getSchemaName( )

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

Description: Returns the schema for the table for the specified column.

getTableName( )

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

Description: Returns the name of the table for the specified column.

isAutoIncrement( )

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

Description: Returns true if the column is automatically numbered and therefore read-only.

isCaseSensitive( )

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

Description: Returns true if the column's case is important.

isCurrency( )

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

Description: Returns true if the value for the specified column represents a currency value.

isDefinitelyWritable( )

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

Description: Returns true if a write operation on the column will definitely succeed.

isNullable( )

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

Description: Returns true if null values are allowed for the column.

isReadOnly( )

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

Description: Returns true if the column is read-only.

isSearchable( )

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

Description: Returns true if the column may be used in a WHERE clause.

isSigned( )

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

Description: Returns true if the column contains a signed number.

isWritable( )

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

Description: Returns true if it is possible for a write on a column to succeed.

Statement

Synopsis

Class Name:java.sql.Statement
Superclass:None
Immediate Subclasses:java.sql.PreparedStatement
Interfaces Implemented:None
Availability:JDK 1.1

Description

This class represents an embedded SQL statement and is used by an application to perform database access. The closing of a Statement automatically closes any open ResultSet associated with the Statement.

Class Summary

```
public interface Statement {
  void addBatch(String sql) throws SQLException;
  void cancel() throws SQLException;
  void clearBatch() throws SQLException;
  void clearWarnings() throws SQLException;
  void close() throws SQLException;
  boolean execute(String sql) throws SQLException;
  int[] executeBatch() throws SQLException;
  ResultSet executeQuery(String sql)
    throws SQLException;
  int executeUpdate(String sql) throws SQLException;
  Connection getConnection() throws SQLException;
  int getFetchDirection() throws SQLException;
  int getFetchSize() throws SQLException;
  int getMaxFieldSize() throws SQLException;
  int getMaxRows() throws SQLException;
  boolean getMoreResults() throws SQLException;
  int getQueryTimeout() throws SQLException;
  ResultSet getResultSet() throws SQLException;
  int getResultSetConcurrency() throws SQLException;
  int getResultSetType() throws SQLException;
  int getUpdateCount() throws SQLException;
  SQLWarning getWarnings() throws SQLException;
  void setCursorName(String name) throws SQLException;
  void setEscapeProcessing(boolean enable)
    throws SQLException;
  void setFetchDirection(int dir) throws SQLException;
  void setFetchSize(int rows) throws SQLException;
  void setMaxFieldSize(int max) throws SQLException;
  void setMaxRows(int max) throws SQLException;
  void setQueryTimeout(int seconds)
    throws SQLException;
}
```

Object Methods

addBatch( )

```
public void addBatch(String sql) throws SQLException
```

Description: Adds the specified SQL statement to the current set of batch commands.

cancel( )

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

Description: In a multithreaded environment, you can use this method to indicate that any processing for this Statement should be canceled. In this respect, it is similar to the stop() method for Thread objects.

clearBatch( )

```
public void clearBatch() throws SQLException
```

Description: Clears out any batch statements.

clearWarnings( ) and getWarnings( )

```
public void clearWarnings() throws SQLException
public SQLWarning getWarnings() throws SQLException
```

Description: The clearWarnings() method allows you to clear all warnings from the warning chain associated with this class. The getWarnings() method retrieves the first warning on the chain. You can retrieve any subsequent warnings on the chain using that first warning.

close( )

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

Description: Manually closes the Statement. This is generally not required because a Statement is automatically closed whenever the Connection associated with it is closed.

execute( ), executeQuery( ), and executeUpdate( )

```
public boolean execute(String sql) throws SQLException
public ResultSet executeQuery(String sql) throws SQLException
public int executeUpdate(String sql) throws SQLException
```

Description: Executes the Statement by passing the specified SQL to the database. The first method, execute(), allows you to execute the Statement when you do not know if it is a query or an update. It will return true if the statement has result sets to process.

The executeQuery() method is used for executing queries. It returns a result set for processing.

The executeUpdate() statement is used for executing updates. It returns the number of rows affected by the update.

executeBatch( )

```
public int[] executeBatch(String sql) throws SQLException
```

Description: Submits the batched list of SQL statements to the database for execution. The return value is an array of numbers that describe the number of rows affected by each SQL statement.

getConnection( )

```
public Connection getConnection() throws SQLException
```

Description: Returns the Connection object associated with this Statement.

getFetchDirection( ), setFetchDirection( ), getFetchSize( ), and setFetchSize( )

```
public int getFetchDirection() throws SQLException
public void setFetchDirection(int dir) throws SQLException
public int getFetchSize() throws SQLException
public void setFetchSize(int rows) throws SQLException
```

Description: These methods provide optimization hints for the driver. The driver is free to ignore these hints. The fetch size is the suggested number of rows the driver should prefetch for each time it grabs data from the database. The direction is a hint to the driver about the direction in which you intend to work.

getMaxFieldSize( ) and setMaxFieldize( )

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

Description: These methods support the maximum field size attribute that determines the maximum amount of data for any BINARY, VARBINARY, LONGVARBINARY, CHAR, VARCHAR, and LONGVARCHAR column value. If the limit is exceeded, the excess is silently discarded.

getMaxRows( ) and setMaxRows( )

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

Description: This attribute represents the maximum number of rows a ResultSet can contain. If this number is exceeded, then any excess rows are silently discarded.

getMoreResults( )

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

Description: This method moves to the next result and returns true if that result is a ResultSet. Any previously open ResultSet for this Statement is then implicitly closed. If the next result is not a ResultSet or if there are no more results, this method will return false. You can test explicitly for no more results using:

```
(!getMoreResults() && (getUpdateCount() == -1)
```

getQueryTimeout( ) and setQueryTimeout( )

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

Description: This attribute is the amount of time a driver will wait for a Statement to execute. If the limit is exceeded, an SQLException is thrown.

getResultSet( )

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

Description: This method returns the current ResultSet. You should call this only once per result. You never need to call this for executeQuery() calls that return a single result.

getResultSetConcurrency( )

```
public int getResultSetConcurrency() throws SQLException
```

Description: Returns the concurrency for the result sets generated by this Statement.

getResultSetType( )

```
public int getResultSetType() throws SQLException
```

Description: Returns the result set type for any result sets generated by this Statement.

getUpdateCount( )

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

Description: If the current result was an update, this method returns the number of rows affected by the update. If the result is a ResultSet or if there are no more results, –1 is returned. As with getResultSet(), this method should only be called once per result.

getWarnings( )

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

Description: Retrieves the first warning associated with this object.

setCursorName( )

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

Description: This method specifies the cursor name to be used by subsequent Statement executions. For databases that support positioned updates and deletes, you can then use this cursor name in coordination with any ResultSet objects returned by your execute() or executeQuery() calls to identify the current row for a positioned update or delete. You must use a different Statement object to perform those updates or deletes. This method does nothing for databases that do not support positioned updates or deletes.

setEscapeProcessing( )

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

Description: Escape processing is on by default. When enabled, the driver will perform escape substitution before sending SQL to the database.

Struct

Synopsis

Class Name:java.sql.Struct
Superclass:None
Immediate Subclasses:None
Interfaces Implemented:None
Availability:New as of JDK 1.2

Description

This class maps to a SQL3 structured type. A Struct instance has values that map to each of the attributes in its associated structured value in the database.

Class Summary

```
public interface Struct {
   Object[] getAttributes() throws SQLException;
   Object[] getAttributes(Map map) throws SQLException;
   String getSQLTypeName() throws SQLException;
}
```

Object Methods

getAttributes( )

```
public Object[] getAttributes() throws SQLException
public Object[] getAttributes(Map map) throws SQLException
```

Description: Provides the values for the attributes in the SQL structured type in order. If you pass a type map, it will use that type map to construct the Java values.

getSQLTypeName( )

```
public String getSQLTypeName() throws SQLException
```

Description: Provides the SQL type name for this structured type.

Time

Synopsis

Class Name:java.sql.Time
Superclass:java.util.Date
Immediate Subclasses:None
Interfaces Implemented:None
Availability:JDK 1.1

Description

This version of the java.util.Date class maps to an SQL TIME datatype.

Class Summary

```
public class Time extends java.util.Date {
   static public Time valueOf(String s);
   public Time(int hour, int minute, int second);
   public Time(long time);
   #public int getDate();
   #public int getDay();
   #public int getMonth();
   #public int getYear();
   #public int setDate(int i);
   #public int setMonth(int i);
   public void setTime(long time);
   #public void setYear(int i);
   public String toString();
}
```

Object Constructors

Time( )

```
public Timestamp(int hour, int minute, intsecond)
public Timestamp(long time)
```

Description: Constructs a new Time object. The first prototype constructs a Time for the hour, minute, and seconds specified. The second constructs one based on the number of seconds since 12:00:00 January 1, 1970 GMT.

Object Methods

getDate( ), setDate( ), getDay( ), getMonth( ), setMonth( ), getYear( ), and setYear( )

```
#public int getDate()
#public int getDay()
#public int getMonth()
#public int getYear()
#public int setDate(int i)
#public int setMonth(int i)
#public void setYear(int i)
```

Description: These attributes represent the individual segments of a Time object.

setTime( )

```
public void setTime(long time)
```

Description: This method sets the Time object to the specified time as the number of seconds since 12:00:00 January 1, 1970 GMT.

toString( )

```
public String toString()
```

Description: Formats the Time into a String in the form of hh:mm:ss.

valueOf( )

```
static public Timestamp valueOf(String s)
```

Description: Create a new Time based on a String in the form of hh:mm:ss.

Timestamp

Synopsis

Class Name:java.sql.Timestamp
Superclass:java.util.Date
Immediate Subclasses:None
Interfaces Implemented:None
Availability:JDK 1.1

Description

This class serves as an SQL representation of the Java Date class specifically designed to serve as an SQL TIMESTAMP. It also provides the ability to hold nanoseconds as required by SQL TIMESTAMP values. You should keep in mind that this class uses the java.util.Date version of hashcode(). This means that two timestamps that differ only by nanoseconds will have identical hashcode() return values.

Class Summary

```
public class Timestamp extends java.util.Date {
   static public Timestamp valueOf(String s);
   #public Timestamp(int year, int month, int date,
      int hour, int minute, int second, int nano);
   public Timestamp(long time);
   public boolean after(Timestamp t);
   public boolean before(Timestamp t);
   public boolean equals(Timestamp t);
   public int getNanos();
   public void setNanos(int n);
   public String toString();
}
```

Object Constructors

Timestamp( )

```
#public Timestamp(int year, int month, int date, int hour, int minute,
      int second, int nano)
public Timestamp(long time)
```

Description: Constructs a new Timestamp object. The first prototype constructs a Timestamp for the year, month, date, hour, minute, seconds, and nanoseconds specified. The second prototype constructs one based on the number of seconds since 12:00:00 January 1, 1970 GMT.

Object Methods

after( )

```
public boolean after(Timestamp t)
```

Description: Returns true if this Timestamp is later than the argument.

before( )

```
public boolean before(Timestamp t)
```

Description: Returns true if this Timestamp is earlier than the argument.

equals( )

```
public boolean equals(Timestamp t)
```

Description: Returns true if the two timestamps are equivalent.

getNanos( ) and setNanos( )

```
public int getNanos()
public void setNanos(int n)
```

Description: This attribute represents the number of nanoseconds for this Timestamp.

toString( )

```
public String toString()
```

Description: Formats the Timestamp into a String in the form of yyyy-mm-dd hh:mm:ss.fffffffff.

valueOf( )

```
static public Timestamp valueOf(String s)
```

Description: Creates a new Timestamp based on a String in the form of yyyy-mm-dd hh:mm:ss.fffffffff.

Types

Synopsis

Class Name:java.sql.Types
Superclass:java.lang.Object
Immediate Subclasses:None
Interfaces Implemented:None
Availability:JDK 1.1

Description

This class holds static attributes representing SQL data types. These values are the actual constant values defined in the XOPEN specification.

Class Summary

```
public class Types {
  static public final int ARRAY;
  static public final int BIGINT;
  static public final int BINARY;
  static public final int BIT;
  static public final int BLOB;
  static public final int CHAR;
  static public final int CLOB;
```

```
    static public final int DATE;
    static public final int DECIMAL;
    static public final int DISTINCT;
    static public final int DOUBLE;
    static public final int FLOAT;
    static public final int INTEGER;
    static public final int JAVA_OBJECT;
    static public final int LONGVARBINARY;
    static public final int LONGVARCHAR;
    static public final int NULL;
    static public final int NUMERIC;
    static public final int OTHER;
    static public final int REAL;
    static public final int REF;
    static public final int SMALLINT;
    static public final int STRUCT;
    static public final int TIME;
    static public final int TIMESTAMP;
    static public final int TINYINT;
    static public final int VARBINARY;
    static public final int VARCHAR;
}
```