

User name:

Book: SQL in a Nutshell, 2nd Edition

No part of any chapter or book may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher of the book or chapter. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates these Terms of Service is strictly prohibited. Violators will be prosecuted to the full extent of U.S. Federal and Massachusetts laws.

5.6. Retrieving Data

A typical database program retrieves data from a database server and processes it. These programs will execute *SELECT* statements following a process similar to that described in the previous section. The difference between executing a *SELECT* statement as opposed to a statement that does not return results is that with *SELECT* you must execute additional code to process the results returned by *SELECT*.

5.6.1. Retrieving Data Using ADO.NET

The following C# code fragment executes a SQL *SELECT* statement that returns author names from the **authors** table and then iterates through the results one row at a time, printing out each author's name. The code is explained in the detailed steps that follow:

```
{Odbc|OleDb|Sql}Command statement = connection.CreateCommand( );
statement.CommandText = "SELECT au_fname, au_lname FROM authors";
{Odbc|OleDb|Sql}DataReader resultSet = statement.ExecuteReader( );
while( resultSet.Read( ) )
{
    String fname = "NULL";
    String lname = "NULL";
    if( !resultSet.IsDBNull( 0 ) ) fname = resultSet.GetString( 0 );
    if( !resultSet.IsDBNull( 1 ) ) lname = resultSet.GetString( 1 );
    System.Console.WriteLine( lname + ", " + fname );
}
resultSet.Close( );
statement.Close( );
```



Column ordinals are all zero-based (the first column is 0, the second is 1, etc.) in ADO.NET, which is different from the one-based ordinals used by JDBC.

To execute a SQL query and process the results using ADO.NET, take the following steps:

1. Create the `Command` object that will be used to execute the *SELECT* statement and attach a *SELECT* statement to it:

```
{Odbc|OleDb|Sql}Command statement = connection.CreateCommand( );
statement.CommandText = "SELECT au_fname, au_lname FROM AUTHORS";
```

2. Invoke the `ExecuteReader` method on the `Command` object, creating a new `DataReader` object.

```
{Odbc|OleDb|Sql}DataReader resultSet = statement.ExecuteReader( );
```

3. Iterate through each row in the result set. After execution of the *SELECT* statement, retrieve rows using the `DataReader`'s `Read` method. If you expect a multiple row result, you should invoke the `Read` method from within a `while` loop:

```
while( resultSet.Read( ) )
{
```

4. Having fetched a row of data into the `DataReader` object, the column data can be extracted using the `DataReader`'s `Get` methods. Before calling the `Get` methods, we check to see if the data is NULL using the `DataReader`'s `IsDBNull` method. If the value returned by the `IsDBNull` method is true, then the value of the string will be NULL. There are many `Get` methods for each type of column. Using the appropriate datatype for the column is important, because a conversion is not always possible. For a list of available `Get` methods and the ANSI SQL types they should be used with, please see [Table 5-4](#).

```
String fname = "NULL";
String lname = "NULL";
if( !resultSet.IsDBNull( 0 ) ) fname = resultSet.GetString( 0 );
if( !resultSet.IsDBNull( 1 ) ) lname = resultSet.GetString( 1 );
```

When creating an ADO.NET program that fetches data that can contain NULL values, it is always safest to check the return value of the `DataReader` object's `IsDBNull` method prior to extracting a value.

5. The `Read` method of `DataReader` returns false when all the data has been read from the database server. At that point, the `Close` methods on the `Command` object and `DataReader` should be invoked to free up the resources used internally to process the statement.

```
resultSet.Close( );
statement.Close( );
```

Table 5-4. ADO.NET DataReader Get methods

Method name	Description
<code>GetBoolean(int i)</code>	Returns the value from the <i>i</i> th column as a Boolean value, where <i>i</i> is the zero-based column number.
<code>GetByte(int i)</code>	Returns the value from the <i>i</i> th column as a single-byte value, where <i>i</i> is the zero-based column number. No conversion will be done if the data in the column exceeds one byte and an <code>InvalidCastException</code> object will be thrown.
<code>GetBytes(int i, long dataIndex, byte[] buffer, int bufferIndex, int length)</code>	Returns the value from the <i>i</i> th column as a binary value, where <i>i</i> is the zero-based column number, <i>dataIndex</i> is the offset in the column value to start reading, <i>buffer</i> is the <code>byte</code> array to copy the data into, <i>bufferIndex</i> is the offset into <i>buffer</i> at which to start copying data, and <i>length</i> is the maximum length to copy into <i>buffer</i> .
<code>GetChar(int i)</code>	Returns a single <code>char</code> value from the <i>i</i> th column. For use with character type columns, where <i>i</i> is the zero-based column number.
<code>GetChars(int i, long dataIndex, char[] buffer, int bufferIndex, int length)</code>	Returns a string of characters from the <i>i</i> th column. For use with character type columns, where <i>i</i> is the zero-based column number, <i>dataIndex</i> is the offset in the column value to start reading, <i>buffer</i> is the <code>char</code> array to copy the data into, <i>bufferIndex</i> is the offset into <i>buffer</i> at which to start copying data, and <i>length</i> is the maximum length to copy into <i>buffer</i> .
<code>GetDataTypeName(int i)</code>	Gets a <code>String</code> value that contains the name of the column's datatype, where <i>i</i> is the zero-based column number.
<code>GetDateTime(int i)</code>	Returns a <code>DateTime</code> value from the <i>i</i> th column. For use with temporal type columns, where <i>i</i> is the zero-based column number.
<code>GetDecimal(int i)</code>	Returns a single <code>Decimal</code> value from the <i>i</i> th column. For use with numeric type columns, where <i>i</i> is the zero-based column number.
<code>GetDouble(int i)</code>	Returns a single <code>double</code> value from the <i>i</i> th column. For use with double precision type columns, where <i>i</i> is the zero-based column number.
<code>GetFloat(int i)</code>	Returns a single <code>float</code> value from the <i>i</i> th column. For use with floating-point columns, where <i>i</i> is the zero-based column number.
<code>GetInt{16,32,64}(int i)</code>	Retrieves data from integer columns. The precision of the return value is encoded within the function names. Use <code>GetInt16</code> for a 16-bit signed <code>short</code> integer, <code>GetInt32</code> for a 32-bit signed <code>int</code> , and <code>GetInt64</code> for a 64-bit signed <code>long</code> .
<code>GetName(int i)</code>	Returns a <code>String</code> value containing the name of the <i>i</i> th column, where <i>i</i> is the zero-based column number.
<code>GetOrdinal(string name)</code>	Returns an <code>int</code> value containing the ordinal value of the column with name matching the <i>name</i> argument.
<code>GetString(int i)</code>	Returns a single <code>String</code> value from the <i>i</i> th column. For use with character type columns, where <i>i</i> is the zero-based column number.

Beyond the `Get` methods, there are three other methods on the `DataReader` type that are frequently used when processing data from a query. Those three methods are listed in Table 5-5.

Table 5-5. Frequently used ADO.NET DataReader methods

Method name	Description
<code>Close()</code>	Closes the <code>DataReader</code> object, freeing up resources held by the instance.
<code>IsDBNull(int i)</code>	Returns true if the specified column is NULL, otherwise it returns false, where <i>i</i> is the zero-based column number.
<code>Read()</code>	Fetch the next row if one is available, and return true, otherwise return false.

5.6.2. Retrieving Data Using JDBC

The following Java code fragment executes a SQL *SELECT* statement that returns author names from the **authors** table and then iterates through the results one row at a time, printing out each author's name:

```
java.sql.Statement statement = connection.createStatement( );
java.sql.ResultSet result =
    statement.executeQuery("SELECT au_fname, au_lname FROM authors" );
while( result.next( ) ) {
    String fname = result.getString( 1 );
    if( result.isNull( ) ) fname = "NULL";
    String lname = result.getString( 2 );
    if( result.isNull( ) ) lname = "NULL";
    System.out.println( lname + ", " + fname );
}
result.close( );
statement.close( );
```



Column ordinals are all one-based (first column is 1, second is 2, etc.) in JDBC, which is different from the zero-based ordinals used by ADO.NET.

5.6.2.1. Use the following steps to execute query statements in JDBC:

1. Create a JDBC `Statement` object by invoking the `createStatement` method on a valid `Connection` object:

```
java.sql.Statement statement = connection.createStatement( );
```

2. The query is executed by invoking one of the `execute` methods on the `Statement` object. Result sets from query statements are processed by JDBC `ResultSet` objects, which are returned from the `executeQuery` method of JDBC `Statement` objects.

```
java.sql.ResultSet result =
    statement.executeQuery("SELECT au_fname, au_lname FROM authors" );
```

3. After creating a `ResultSet` object, you can iterate through one row at a time by invoking the `next` method. The `next` method returns a Boolean value, true for each row in a result set and false after all rows have been iterated through. It is common to invoke the `next` method within a `while` loop to process the rows one-by-one. The `ResultSet` does not begin on the first row of the result, so you must invoke the `next` method to advance to the first row before calling any of the `get` methods.

```
while( result.next( ) ) {
```

4. To retrieve column data from the rows within a result set, invoke the appropriate `get` method on the `ResultSet` object. For a list of the most common `get` methods, check [Table 5-6](#). Note that the column data is checked for a NULL value after the `get` method has been invoked, since the nullness of a value can't be determined from the value returned by the `get` methods.

```
String fname = result.getString("au_fname");
if( result.isNull( ) ) fname = "NULL";
String lname = result.getString("au_lname");
if( result.isNull( ) ) lname = "NULL";
```

When creating a JDBC program that fetches data from nullable columns, it is always safest to check the value returned from a `ResultSet` object for a NULL value using the `wasNull` method.



The JDBC `get` methods that return object types will return Java NULL values when returning database NULL values. However, this does not apply to non-object types such as `getInt()`, which returns zero in the case of a database NULL. For this reason, this chapter uses the verbose `wasNull()` method to test the value for a NULL.

5. Free the resources held by the result set and statement objects. When finished with the `ResultSet` and `Statement` objects, invoke their `close` methods so that database resources can be freed.

```
result.close();  
statement.close();
```

Table 5-6. JDBC ResultSet get methods

Method name	Description
<code>getBlob({int i String name})</code>	Retrieves a <code>Blob</code> value from <i>BLOB</i> type columns, where <i>i</i> is the one-based column ordinal and <i>name</i> is the name of the column.
<code>getBoolean({int i String name})</code>	Retrieves a <code>boolean</code> value from <i>BOOLEAN</i> type columns, where <i>i</i> is the one-based column ordinal and <i>name</i> is the name of the column.
<code>getByte({int i String name})</code>	Retrieves a <code>byte</code> value from <i>CHARACTER</i> or <i>BINARY</i> type columns, where <i>i</i> is the one-based column ordinal and <i>name</i> is the name of the column.
<code>getBytes({int i String name})</code>	Retrieves a <code>byte[]</code> value from <i>BINARY</i> type columns, where <i>i</i> is the one-based column ordinal and <i>name</i> is the name of the column.
<code>getClob({int i String name})</code>	Retrieves a <code>Clob</code> value from <i>CLOB</i> type columns, where <i>i</i> is the one-based column ordinal and <i>name</i> is the name of the column.
<code>getDate({int i String name})</code>	Retrieves a <code>Date</code> value from <i>TEMPORAL</i> type columns, where <i>i</i> is the one-based column ordinal and <i>name</i> is the name of the column.
<code>getDouble({int i String name})</code>	Retrieves a <code>double</code> value from <i>DOUBLE PRECISION</i> type columns, where <i>i</i> is the one-based column ordinal and <i>name</i> is the name of the column.
<code>getFloat({int i String name})</code>	Retrieves a <code>float</code> value from <i>REAL</i> type columns, where <i>i</i> is the one-based column ordinal and <i>name</i> is the name of the column.
<code>getInt({int i String name})</code>	Retrieves an <code>int</code> value from <i>INTEGER</i> type columns, where <i>i</i> is the one-based column ordinal and <i>name</i> is the name of the column.
<code>getLong({int i String name})</code>	Retrieves a <code>long</code> value from <i>INTEGER</i> type columns, where <i>i</i> is the one-based column ordinal and <i>name</i> is the name of the column.
<code>getRow()</code>	Returns the current row number.
<code>getShort({int i String name})</code>	Retrieves a <code>short</code> value from <i>INTEGER</i> type columns, where <i>i</i> is the one-based column ordinal and <i>name</i> is the name of the column.
<code>getString({int i String name})</code>	Retrieves a <code>String</code> value from <i>CHARACTER</i> type columns, where <i>i</i> is the one-based column ordinal and <i>name</i> is the name of the column.
<code>getTime({int i String name})</code>	Retrieves a <code>Time</code> value from <i>TEMPORAL</i> type columns, where <i>i</i> is the one-based column ordinal and <i>name</i> is the name of the column.
<code>getTimestamp({int i String name})</code>	Retrieves a <code>Timestamp</code> value from <i>TEMPORAL</i> type columns, where <i>i</i> is the one-based column ordinal and <i>name</i> is the name of the column.

Beyond the `get` methods, there are three other methods on the `ResultSet` type that are frequently used when processing data from a query. Those three methods are listed in Table 5-7.

Table 5-7. Frequently used JDBC ResultSet methods

Method name	Description
<code>close()</code>	Closes the <code>ResultSet</code> object, freeing up resources held by the instance.
<code>next()</code>	Advances the <code>ResultSet</code> object to the next available row and returns true. If no rows remain false will be returned.
<code>wasNull()</code>	Returns true if the last column returned with a <code>get</code> method contained a database NULL value, returns false otherwise.