

**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.

---

## 5.2. Opening a Database Connection

Before interacting with a database, an application must first establish a connection to the database server. The APIs covered in this book abstract the low-level connectivity details into a few simple object-oriented classes, making it easier to focus on the database application instead of protocols and network topology.

### 5.2.1. Opening an ADO.NET Database Connection

Opening a connection with ADO.NET requires instantiating a connection object with a properly formatted connection string and then invoking the `Open` method on the connection object. The connection object can be an `OdbcConnection`, `SqlConnection`, or an `OleDbConnection`. The `OdbcConnection` is designed for any ODBC datasource, and the `OleDbConnection` type will work with any OLE DB Provider. For the highest performance data access, use connection objects specifically tuned to the specific database platform, such as `SqlConnection` for Microsoft SQL Server. Following is the syntax for creating a `Connection` object in ADO.NET:

```
{Odbc|OleDb|Sql}Connection connection =  
    new {Odbc|OleDb|Sql}Connection(connection_string);  
connection.Open( );
```

The format of the connection string is the same for all of the connection types. The format is a string of key/value pairs delimited by semicolons. For example:

```
key1=value1; key2=value2; key3=value3; ...
```

While the format is the same for every connection type, the keys and values are quite different. Tables [Table 5-1](#) through [Table 5-3](#) list the attributes for the three connection types listed above. Many database platforms support additional attributes that can also be set through the connection string. For a list of these attributes, please consult the appropriate database vendor documentation.

Following are examples of two connection strings for an `OdbcConnection`:

```
DSN=MyOracleDSN; UID=scott; PWD=tiger;  
DRIVER={SQL Server};SERVER=(local);UID=sa;PWD=;DATABASE=pubs;
```

The first string connects to a Data Source Name, or DSN, with the name **MyOracleDSN** using the username **scott** and the password **tiger**.

The second string connects using the **SQL Server** driver to a database named **pubs** on the local server. The username is **sa**. The password is blank, indicating to the driver that the password is not required for the **sa** user.

Following is an example of a connection string for an `OleDbConnection`, which connects to an Oracle9i data source using the MSDAORA OLE DB provider, the use name **scott**, and the password **tiger**.

```
Provider=MSDAORA;Data Source=Oracle9;User ID=scott;Password=tiger;
```

Finally, here is an example of a connection string for a `SqlConnection`, which connects to a SQL Server data source on the local server:

```
Server=(local);UID=sa;PWD=;DATABASE=pubs;Connection Timeout=60;
```

In Tables [Table 5-1](#) through [Table 5-3](#), you'll find synonyms for some keywords. For example, you can use "DSN" and "Data Source Name" interchangeably in your code.

**Table 5-1. Connection string attributes for OdbcConnection**

Keyword	Notes
Data Source Name DSN	The DSN, FILEDSN, or DRIVER attribute must be provided to connect. The DSN is the Data Source Name known by the client's driver manager. The advantage of using a DSN is that the DSN can be changed to point to a different database platform while the database applications are still running.
User ID UID	This value is set to the user identifier authorized to open a new connection.
Password PWD	The password for the user. If the user doesn't have a password for the data source, then this property should still be supplied with an empty value.
DSN_File Name FILEDSN	Similar to the DSN, the FILEDSN is a file, typically with a DSN extension that contains the attributes for the connection object to establish connectivity. Even when using a FILEDSN, the connection may still need the password provided, since passwords aren't stored within DSN files.
Driver Name DRIVER	A Driver may be explicitly used instead of a DSN or FILEDSN. The drawback of using a driver directly is that the application won't be isolated from changes made to the driver and will require slight modifications to target a different database platform.
SAVEFILE	When the SAVEFILE attribute is set to a legal filename, the connection attributes will be saved to the file once successful connectivity is established. This option is only available when using DRIVER and FILEDSN connectivity methods.

**Table 5-2. Connection string attribute for OleDbConnection**

Keyword	Notes
Provider	The name of the provider: this is the only required attribute. The specified provider may require additional attributes.

**Table 5-3. Connection string attributes for SqlConnection**

Keyword	Notes
Application Name	The name of the application that will be displayed from server management utilities.
AttachDBFilename Extended properties Initial File Name	The pathname to an attachable database.
Connect Timeout Connection Timeout	Number of seconds to wait before the connection attempt is aborted. The default is 15 seconds.
Connection Lifetime	Time in seconds the connection can remain in the connection pool, when pooling is enabled.
Connection Reset	Specifies whether or not the connection is reset when it is reused from the connection pool. Default is true.
Current Language	The language the connection session should use.

Keyword	Notes
Data Source Server Address Addr Network Address	The SQL Server instance name or network address.
Initial Catalog Database	The name of the database.
Integrated Security Trusted_Connection	Set to true or <code>sspi</code> for secure connections. The default is false.
Max Pool Size	The maximum number of connections allowed in the connection pool at a time. The default is 100.
Min Pool Size	The minimum number of connections to keep in the connection pool at a time. The default is 0.
Network Library Net	The network library to use in establishing the connection. Valid settings are <code>dbnmpntw</code> for Named Pipes, <code>dbmsrpcn</code> for Multiprotocol, <code>dbmsadsn</code> for AppleTalk, <code>dbmsgnet</code> for VIA, <code>dbmsipcn</code> for Shared Memory, <code>dbmsspxn</code> for IPX/SPX, and <code>dbmssocn</code> for TCP/IP. The default is <code>dbmssocn</code> for TCP/IP.
Packet Size	The network packet size in bytes; the default is 8,192.
Password Pwd	The user's password.
Persist Security Info	Determines if security-sensitive connection properties, such as the password, are stored within the connection object after a connection has been attempted or completed. The default is false.
Pooling	Determines if connection pooling should be used for the connection. The default is true.
User ID	The user's login name.
Workstation ID	The name of the computer connecting to the database.

### 5.2.2. Opening a JDBC Database Connection

Following is the syntax for registering a driver with the JDBC Driver Manager and then opening a database connection:

```
Class.forName(driver_name);
Connection connection = DriverManager.getConnection(connect_string,
                                                    username, password);
```

The first step in establishing JDBC connectivity is to instruct the Java Virtual Machine (JVM) class loader to load the appropriate JDBC driver.

The most common method of loading the driver into the class loader is to use the static `forName` method of the `Class` class. This method can provide applications with greater flexibility in changing database platforms by having the Java Virtual Machine (JVM) load the database driver at runtime:

```
Class.forName( "driver_name" );
```

After the database driver has been loaded, the application can establish connectivity by invoking the static `getConnection` method on the JDBC `DriverManager` class. The `getConnection` method takes three arguments: a connection string, username, and password:

```
Connection connection = DriverManager.getConnection(connect_string,
                                                    username,
                                                    password);
```

The connection string follows the following JDBC URL naming scheme:

```
jdbc:subprotocol:subname
```

Following are examples of how to connect to the different vendors covered in this book using JDBC.

#### 5.2.2.1. DB2

```
Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
Connection connection = DriverManager.getConnection(
    "jdbc:db2:DATABASE", "user", "passwd" );
```

#### 5.2.2.2. MySQL

```
Class.forName("org.gjt.mm.mysql.Driver");
Connection connection = DriverManager.getConnection(
    "jdbc:mysql://127.0.0.1:3306/DATABASE",
    "user", "passwd" );
```

#### 5.2.2.3. PostgreSQL

```
Class.forName("org.postgresql.Driver");
Connection connection = DriverManager.getConnection(
    "jdbc:postgresql://127.0.0.1:5432/DATABASE",
    "user", "passwd" );
```

#### 5.2.2.4. Oracle

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection connection = DriverManager.getConnection(
    "jdbc:oracle:thin:@myserver",
    "scott","tiger" );
```

#### 5.2.2.5. SQL Server

```
Class.forName(
    "com.microsoft.jdbc.sqlserver.SQLServerDriver"
);
Connection connection = DriverManager.getConnection(
    "jdbc:microsoft:sqlserver://SERVER:1433;" +
    "DatabaseName=pubs;" ,
    "user", "passwd" );
```