

Progress DataDirect for ODBC for Salesforce User's Guide

Release 8.0.0

Copyright

Visit the following page online to see Progress Software Corporation's current Product Documentation Copyright Notice/Trademark Legend: <https://www.progress.com/legal/documentation-copyright>.

Updated: 2023/02/22

Table of Contents

Welcome to the Progress DataDirect for ODBC for Salesforce Driver...11

What's new in this release?.....	12
Driver requirements.....	15
ODBC compliance.....	15
Version string information.....	16
getFileVersionString function.....	17
Support for multiple environments.....	17
Support for Windows environments.....	18
Support for UNIX and Linux environments.....	20
Mapping objects to tables.....	25
Data types.....	26
Contacting Technical Support.....	28

Getting started.....29

Configuring and connecting on Windows.....	29
Setting the library path environment variable (Windows).....	30
Configuring a data source.....	30
Testing the connection.....	31
Configuring and connecting on UNIX and Linux.....	31
Environment configuration.....	31
Test loading the driver.....	32
Setting the library path environment variable (UNIX and Linux).....	32
Configuring a data source.....	33
Testing the connection.....	33

Tutorials.....35

Accessing data in Tableau (Windows only).....	35
Accessing data in Microsoft Excel (Windows only).....	38
Accessing data in Microsoft Excel from the Query Wizard (Windows only).....	40

Using the driver.....43

Configuring and connecting to data sources.....	44
Configuring the product on UNIX/Linux.....	44
Data source configuration through a GUI.....	53
Using a connection string.....	76
Using a logon dialog box.....	77
Connecting through a proxy server.....	77

Performance considerations.....	78
Using the SQL engine server.....	80
Configuring the SQL engine server on Windows.....	80
Configuring the SQL engine server on UNIX/Linux.....	83
Configuring Java logging for the SQL engine server.....	85
Client-side caches.....	85
Creating a cache.....	86
Modifying a cache definition.....	86
Disabling and enabling a cache.....	87
Refreshing cache data.....	87
Dropping a cache.....	88
Cache metadata.....	88
Catalog tables.....	88
SYSTEM_CACHES catalog table.....	88
SYSTEM_CACHE_REFERENCES catalog table.....	90
SYSTEM_REMOTE_SESSIONS catalog table.....	91
SYSTEM_SESSIONINFO catalog table.....	92
SYSTEM_SESSIONS catalog table.....	93
Reports.....	94
Using security.....	95
Authentication.....	95
Data encryption across the network.....	100
TLS/SSL encryption.....	100
Summary of security-related options.....	102
Using DataDirect Connection Pooling.....	102
Creating a connection pool.....	103
Adding connections to a pool.....	103
Removing connections from a pool.....	103
Handling dead connections in a pool.....	104
Connection pool statistics.....	104
Configuring pooling-related options.....	105
Using identifiers.....	106
Timeouts.....	106
Views and remote/local tables.....	107
SQL support.....	107
Isolation and lock levels supported.....	107
Unicode support.....	107
Parameter metadata support.....	108
Insert and Update statements.....	108
Select statements.....	108
DataDirect Bulk Load.....	109
Bulk export and load methods.....	109
Exporting data from a database.....	110
Bulk loading to a database.....	112
The bulk load configuration file.....	113

Sample applications.....115

Character set conversions.....115

External overflow files.....116

Summary of DataDirect Bulk Load related options.....116

Using Bulk API with SQL statements118

 Summary of options for using the Bulk API with SQL statements118

Connection option descriptions.....123

Access Token.....129

Application Using Threads.....130

Apply ToLabel.....130

Authentication Method.....131

Bulk Fetch Threshold.....132

Bulk Load Asynchronous.....133

Bulk Load Batch Size.....133

Bulk Load Concurrency Mode.....134

Bulk Load Job Size.....135

Bulk Load Poll Interval.....135

Bulk Load Threshold.....136

Bulk Load Timeout.....137

Bulk Load Version.....137

Client ID.....138

Client Secret.....139

Config Options.....139

 AuditColumns (Config Option).....140

 CustomSuffix (Config Option).....141

 KeywordConflictSuffix (Config Option).....142

 MapSystemColumnNames (Config Option).....143

 NumberFieldMapping (Config Option).....144

 UppercaseIdentifiers (Config Option).....145

Connection Pooling.....145

Connection Reset.....146

Create Database.....147

Create Map.....148

Data Source Name.....148

Database.....149

Description.....150

Enable Bulk Fetch.....150

Enable Bulk Load.....151

Enable Primary Key Chunking.....152

Fetch Size.....153

Field Delimiter.....154

Host Name.....155

IANAAppCodePage.....155

Initialization String.....	156
JVM Arguments.....	157
JVM Classpath.....	158
LoadBalance Timeout.....	159
Log Config File.....	160
Login Timeout.....	161
Logon Domain.....	161
Max Pool Size.....	162
Min Pool Size.....	163
Password.....	163
Primary Key Chunk Size.....	164
Proxy Host.....	165
Proxy Password.....	166
Proxy Port.....	166
Proxy User.....	167
Read Only.....	167
Record Delimiter.....	168
Refresh Dirty Cache.....	169
Refresh Schema.....	169
Refresh Token.....	170
Report Codepage Conversion Errors.....	171
Schema Map.....	172
Security Token.....	173
Server Port Number.....	174
SQL Engine Mode.....	175
Statement Call Limit.....	176
Statement Call Limit Behavior.....	176
Transaction Mode.....	177
User Name.....	178
WSFetch Size.....	178
WSPoolSize.....	179
WSRetry Count.....	180
WSTimeout.....	181

Supported SQL statements and extensions.....183

Alter Cache (EXT).....	184
Relational caches.....	186
Alter Index.....	186
Alter Sequence.....	186
Alter Session (EXT).....	187
Alter Table.....	188
Altering a remote table.....	188
Altering a local table.....	191
Create Cache (EXT).....	194

Relational caches.....	196
Referencing clause.....	196
Refresh Interval clause.....	197
Initial Check clause.....	197
Persist clause.....	198
Enabled clause.....	198
Call Limit clause.....	199
Filter clause.....	200
Create Index.....	201
Create Sequence.....	201
Next Value For clause.....	202
Create Table.....	202
Creating a remote table.....	203
Creating a local table.....	207
Create View.....	214
Delete.....	215
Drop Cache (EXT).....	216
Drop Index.....	217
Drop Sequence.....	217
Drop Table.....	218
Drop View.....	218
Explain Plan.....	219
Insert.....	219
Specifying an external ID column.....	220
Refresh Cache (EXT).....	221
Refresh Map (EXT).....	222
Select.....	222
Select clause.....	224
Update.....	233
SQL expressions.....	234
Column names.....	235
Literals.....	235
Operators.....	237
Functions.....	241
Conditions.....	241
Subqueries.....	242
IN predicate.....	242
EXISTS predicate.....	243
UNIQUE predicate.....	243
Correlated subqueries.....	243

Welcome to the Progress DataDirect for ODBC for Salesforce Driver

The Progress® DataDirect® for ODBC Salesforce™ driver supports the standard SQL query language to fetch, insert, update, and delete data from Salesforce.com and Force.com. To support SQL access to Salesforce, the driver creates a map of the Salesforce data model and translates SQL statements provided by the application to Salesforce queries (SOQL) and Web service calls.

The driver optimizes performance when executing joins by leveraging data relationships among Salesforce objects to minimize the amount of data that needs to be fetched over the network. The Salesforce driver recognizes the relationships among standard objects and custom objects and can access and update both. Relationships among objects can be reported using the SQLForeignKeys and SQLPrimaryKeys functions.

The driver also provides a client-side data cache. You can improve performance for some queries by defining rules that specify which data to cache on the client as well as when the cached data becomes invalid and needs to be refreshed.

The documentation for the driver also includes the *Progress DataDirect for ODBC Drivers Reference*. The reference provides general reference information for all DataDirect drivers for ODBC, including content on troubleshooting, supported SQL escapes, and DataDirect tools. For the complete documentation set, visit to the Progress DataDirect Connectors Documentation Hub:

<https://docs.progress.com/bundle/datadirect-connectors/page/DataDirect-Connectors-by-data-source.html>.

For details, see the following topics:

- [What's new in this release?](#)
- [Driver requirements](#)
- [ODBC compliance](#)
- [Version string information](#)

- [Support for multiple environments](#)
- [Mapping objects to tables](#)
- [Data types](#)
- [Contacting Technical Support](#)

What's new in this release?

Support and certification

Visit the following web pages for the latest support and certification information.

- Release Notes: <https://www.progress.com/odbc/release-history/>
- Supported Configurations: <https://www.progress.com/supported-configurations/datadirect>
- DataDirect Support Matrices: <https://www.progress.com/matrices/datadirect>

Changes Since 8.0.0

• Driver Enhancements

- The driver has been enhanced to support the Salesforce Bulk API V2 for bulk load operations. This functionality can be enabled and configured with the Bulk Load Version and Bulk Load Job Size connection options. See [Bulk tab](#) on page 68 for details.
- The driver has been enhanced to support the Apply ToLabel connection option. This option determines whether the driver applies the toLabel() function to the picklist fields when executing queries. See [Apply ToLabel](#) on page 130 for details.
- The driver has been enhanced to include timestamp in the internal packet logs by default. If you want to disable the timestamp logging in packet logs, set `PacketLoggingOptions=1`. The internal packet logging is not enabled by default. To enable it, set `EnablePacketLogging=1`. The driver has been enhanced to support Apache CXF 3.3.4
- `yes`, `no`, `on` and `off` have been added as valid values for the connection options that accept boolean values.
- The driver has been enhanced to support OAuth 2.0 authentication. See [Configuring OAuth 2.0 authentication](#) on page 95 for details.

• Changed Behavior

- The driver has been updated to require a Java Virtual Machine (JVM) that is Java SE 8 or higher, including Oracle JDK, OpenJDK, and IBM SDK (Java) distributions. See [Driver requirements](#) on page 15 for more details on JVM.

Java SE 7 has reached the end of its product life cycle and will no longer receive generally available security updates. As a result, the driver will no longer support JVMs that are version Java SE 7 or earlier. Support for distributed versions of Java SE 7 and earlier will also end.
- The configuration options have been deprecated. However, the driver will continue to support them until the next major release of the driver.
- The precision for the Integer data type has been changed from 10 to 9.
- The NUM_PREC_RADIX value for the Double data type has been changed from 10 to 2.

- The following Windows platforms have reached the end of their product lifecycle and are no longer supported by the driver:
 - Windows 8.0 (versions 8.1 and higher are still supported)
 - Windows Vista (all versions)
 - Windows XP (all versions)
 - Windows Server 2003 (all versions)

Changes For 8.0.0

• Driver Enhancements

- The driver has been enhanced to support the Salesforce Bulk API, including PK chunking, for bulk fetch operations. This functionality can be enabled and configured with the Enable Bulk Fetch, Bulk Fetch Threshold, Enable Primary Key Chunking, and PK Chunk Size connection options. See [Summary of options for using the Bulk API with SQL statements](#) on page 118 for details.
- The driver has been enhanced to support multiple simultaneous sessions. This allows the driver to scale performance by distributing the workload across multiple sessions. The number of active sessions should not exceed the number permitted by your Salesforce account and can be limited by the setting of the new WSPoolSize connection option. See [WSPoolSize](#) on page 179 for details.
- The following Refresh Map SQL extension has been added to the driver. REFRESH MAP discovers native objects that have been added to the native data store since connection or since the last refresh and maps the objects into your relational view of native data. It also incorporates any configuration changes made to your relational view by reloading the schema map and associated files. See [Refresh Map \(EXT\)](#) on page 222 for details.
- The SQL Engine Mode connection option now supports Auto mode. When this setting is enabled, the driver automatically determines whether the SQL engine runs in server or direct mode based on availability. In addition, the default value for the SQL Engine Mode connection option on Windows has been updated to 0 (Auto). See [SQL Engine Mode](#) on page 175 for details.
- The driver is now compiled using Visual Studio 2015 for improved security.
- The driver includes a new Tableau data source file (Windows only) that provides improved functionality when accessing your Salesforce data with Tableau. See [Accessing data in Tableau \(Windows only\)](#) on page 35 for details.
- The driver and Driver Manager have been enhanced to support UTF-8 encoding in the `odbc.ini` and `odbcinst.ini` files.

Refer to the "Character encoding in the odbc.ini and odbcinst.ini files" in *Progress DataDirect for ODBC Drivers Reference* for details.

• Changed Behavior

- Support for HP-UX PA-RISC 32-bit, Oracle Solaris SPARC 32-bit, and Oracle Solaris x86 32-bit platforms has been deprecated.
- Support for Windows XP and Windows Server 2003 has been deprecated.
- The Salesforce driver has been updated to require a JVM that is version Java SE 7 or higher. This change has been implemented to remain compliant with Salesforce security standards. See [Driver requirements](#) on page 15 for more details on JVM.
- In addition to the information listed here, refer to the [compatibility FAQ](#) for guidance on upgrading from the Progress DataDirect for ODBC for Salesforce 7.1 driver to the 8.0 driver.

- The driver's SQL engine was upgraded for this release. Consequently, there are differences in how the driver handles some SQL queries. Refer to this [SQL engine upgrade document](#) for details. See also [Supported SQL statements and extensions](#) on page 183.
- The 8.0 driver pushes queries to Salesforce whenever possible. Queries that cannot be pushed to Salesforce with the 8.0 driver may be slower than comparable queries made with earlier versions of the driver because data may be paged to disk while completing an operation. If you experience slow performance, please contact [Technical Support](#). Our team will quickly address any performance issues you encounter.
- Bulk load operations are no longer restricted to 10,000 rows for evaluation installations of the driver.
- The native CURRENCY and PERCENTAGE data types now map to the SQL_DECIMAL data type. In earlier releases, these data types mapped to the DOUBLE data type. See [Data types](#) on page 26 details.
- The Database connection option has been replaced by the new Schema Map option. Similar to Database, Schema Map is used to specify the name and location of the configuration file where the relational map of native data is written; however, there are changes to the file name format and default behavior. The Database attribute will continue to be supported for this release, but it will be deprecated in subsequent versions of the product. See [Schema Map](#) on page 172 for details.
- The Create Database connection option has been replaced by the new Create Map option. The CreateDB attribute will continue to be supported for this release, but it will be deprecated in subsequent versions of the product. See [Create Map](#) on page 148 for details.
- The Logon Domain connection option has been deprecated. As a result, the User Name option has been updated to accept the user name and domain. See [User Name](#) on page 178 for details.
- The Refresh Dirty Cache option has been deprecated. Now, for every fetch operation, the driver refreshes the cached object to pick up changes made to tables and rows.
- The Server DB Directory connection option has been deprecated. To specify the location of database files, use the Schema Map connection option. See [Schema Map](#) on page 172 for details.
- The default value for the Server Port Number connection option has been updated:
 - For the 32-bit driver, the default is 19938.
 - For the 64-bit driver, the default is 19937.
- The default value for the JVM Arguments connection option has been updated:
 - For the 32-bit driver when the SQL Engine Mode connection option is set to 2 (Direct):
-Xmx256m
 - For all other configurations:
-Xmx1024m
- The default value of the Enable Bulk Load connection option has been updated to 1 (enabled). By default, the bulk load protocol is used for inserts, updates, and deletes when the size of the operation exceeds the threshold determined by the Bulk Load Threshold option. See [Enable Bulk Load](#) on page 151 for details.
- The default value for the Statement Call Limit connection option has been updated to 100. By default, the driver can make a maximum of 100 Web service calls when executing any single SQL statement or metadata query. See [Statement Call Limit](#) on page 176 for details.
- The default value for the AuditColumns config option has been updated to All (AuditColumns=All). By default, the driver includes the all of the audit columns and the master record id column in its table definitions. See [AuditColumns \(Config Option\)](#) on page 140 for details.

- The default value for the CustomSuffix config option has been updated to `Include` (`CustomSuffix=Include`). By default, the driver includes the "`__c`" suffix table and column names when mapping the remote data model to the relational data model. See [CustomSuffix \(Config Option\)](#) on page 141 for details.
- The default value for the MapSystemColumnNames config option has been updated to `0` (`MapSystemColumnNames=0`). By default, the driver does not change the names of the Salesforce system columns when mapping them to the relational model. See [MapSystemColumnNames \(Config Option\)](#) on page 143 for details.
- The short attribute name for the Config Options connection option has changed from `CO` to `CFGO`. See [Config Options](#) on page 139 for details.
- The Security tab has been removed from the setup dialog. As a result:
 - The Security Token connection option has been moved to the General tab.
 - The User Name connection option has been moved to the General tab.

Driver requirements

The driver has been updated to require a Java Virtual Machine (JVM) that is Java SE 8 or higher, including Oracle JDK, OpenJDK, and IBM SDK (Java) distributions.

ODBC compliance

The driver is compliant with the Open Database Connectivity (ODBC) specification.

Refer to "ODBC API and scalar functions" in the *Progress DataDirect for ODBC Drivers Reference* for additional information.

The Salesforce driver extends the standard results returned by the `SQLColumns` ODBC function to include the `IS_EXTERNAL_ID` column, as shown in the following table.

Table 1: Extended Functionality for the SQLColumns Function

Column	Data Type	Description
<code>IS_EXTERNAL_ID</code>	<code>VARCHAR (3), NOT NULL</code>	<p>Provides an indication of whether the column can be used as an External ID. External ID columns can be used as the lookup column for insert and upsert operations and foreign-key relationship values. Valid values are:</p> <ul style="list-style-type: none"> • <code>YES</code>: The column can be used as an external ID. • <code>NO</code>: The column cannot be used as an external ID. <p>The standard catalog table <code>SYSTEM_COLUMNS</code> is also extended to include the <code>IS_EXTERNAL_ID</code> column.</p>

The Salesforce driver supports only the following Level 2 functions:

- `SQLColumnPrivileges`

- SQLDescribeParam
- SQLForeignKeys
- SQLPrimaryKeys
- SQLProcedures
- SQLTablePrivileges

Version string information

The driver for Salesforce has a version string of the format:

```
XX.YY.ZZZZ(BAAAA, UBBBB, JDDDDDD)
```

The Driver Manager on UNIX and Linux has a version string of the format:

```
XX.YY.ZZZZ(UBBBB)
```

The component for the Unicode conversion tables (ICU) has a version string of the format:

```
XX.YY.ZZZZ
```

where:

XX is the major version of the product.

YY is the minor version of the product.

ZZZZ is the build number of the driver or ICU component.

AAAA is the build number of the driver's bas component.

BBBB is the build number of the driver's utl component.

DDDDDD is the version of the Java components used by the driver.

For example:

```
08.00.0017 (B0254, U0180, J000109)
  |__|  |__|  |__|  |__|
  Driver Bas  Utl  Java
```



On Windows, you can check the version string through the properties of the driver DLL. Right-click the driver DLL and select **Properties**. The Properties dialog box appears. On the Details tab, the **File Version** will be listed with the other file properties.

You can always check the version string of a driver on Windows by looking at the About tab of the driver's Setup dialog.

UNIX[®] On UNIX and Linux, you can check the version string by using the test loading tool shipped with the product. This tool, `ivtestlib` for 32-bit drivers and `ddtestlib` for 64-bit drivers, is located in `install_directory/bin`.

The syntax for the tool is:

```
ivtestlib shared_object
```


or

```
ddtestlib shared_object
```

For example, for the 32-bit driver on Linux:

```
ivtestlib ivsfrc28.so
```

returns:

```
08.00.0001 (B0002, U0001, J000003)
```

For example, for the Driver Manager on Linux:

```
ivtestlib libodbc.so
```

returns:

```
08.00.0001 (U0001)
```

For example, for the 64-bit Driver Manager on Linux:

```
ddtestlib libodbc.so
```

returns:

```
08.00.0001 (U0001)
```

For example, for the 32-bit ICU component on Linux:

```
ivtestlib libivicu28.so
08.00.0001
```

Note: On AIX, Linux, and Solaris, the full path to the driver does not have to be specified for the test loading tool. The HP-UX version of the tool, however, requires the full path.

getFileVersionString function

Version string information can also be obtained programmatically through the function `getFileVersionString`. This function can be used when the application is not directly calling ODBC functions.

This function is defined as follows and is located in the driver's shared object:

```
const unsigned char* getFileVersionString();
```

This function is prototyped in the `qesqlext.h` file shipped with the product.

Support for multiple environments

Your Progress DataDirect driver is ODBC-compliant for Windows, UNIX, and Linux operating systems. This section explains the environment-specific differences when using the database drivers in your operating environment.

The sections "Support for Windows Environments" and "Support for UNIX and Linux Environments" contain information specific to your operating environment.

The following sections refer to threading models.

Refer to "Threading" in the *Progress DataDirect for ODBC Drivers Reference* for more information.

Note: Support for operating environments and database versions are continually being added. For the latest information about supported platforms and databases, refer to the Progress DataDirect certification matrices page at <https://www.progress.com/matrices/datadirect>.

See also

[Support for Windows environments](#) on page 18

[Support for UNIX and Linux environments](#) on page 20

Support for Windows environments

The following are requirements for the 32- and 64-bit drivers on Windows operating systems.

32-bit driver

- All required network software that is supplied by your database system vendors must be 32-bit compliant.
- If your application was built with 32-bit system libraries, you must use 32-bit driver. If your application was built with 64-bit system libraries, you must use 64-bit driver (see "64-bit driver"). The database to which you are connecting can be either 32-bit or 64-bit enabled.
- The following processors are supported:
 - x86: Intel
 - x64: Intel and AMD
- The following operating systems are supported for your Progress DataDirect for ODBC driver. All editions are supported unless otherwise noted.
 - Windows Server 2016
 - Windows 10
 - Windows 8.1
 - Windows Server 2012
 - Windows 7
 - Windows Server 2008
- A 32-bit Java Virtual Machine (JVM), Java SE 8 or higher, is required. Also, you must set the PATH environment variable to the directory containing your 32-bit JVM's `jvm.dll` file, and that directory's parent directory. See "Driver requirements" for more details on JVM.
- An application that is compatible with components that were built using Microsoft Visual Studio 2015 compiler and the standard Win32 threading model.
- You must have ODBC header files to compile your application. For example, Microsoft Visual Studio includes these files.

See also

[64-bit driver](#) on page 19

[Driver requirements](#) on page 15

64-bit driver

- All required network software that is supplied by your database system vendors must be 64-bit compliant.
- The following processors are supported:
 - Intel
 - AMD
- The following operating systems are supported for your 64-bit driver. All editions are supported unless otherwise noted.
 - Windows Server 2016
 - Windows 10
 - Windows 8.1
 - Windows Server 2012
 - Windows 7
 - Windows Server 2008
- An application that is compatible with components that were built using Microsoft C/C++ Optimizing Compiler Version 14.00.40310.41 and the standard Windows 64 threading model.
- A 64-bit JVM, Java SE 8 or higher, is required. Also, you must set the PATH environment variable to the directory containing your 64-bit JVM's `jvm.dll` file, and that directory's parent directory. See "Driver Requirements" for more details on JVM.
- You must have ODBC header files to compile your application. For example, Microsoft Visual Studio includes these files.

See also

[Driver requirements](#) on page 15

Setup of the driver

The driver must be configured before it can be used. See "Getting started" for information about using the Windows ODBC Administrator. See "Configuring and connecting to data sources" for details about driver configuration.

See also

[Getting started](#) on page 29

[Configuring and connecting to data sources](#) on page 44

Driver file names for Windows

The prefix for all 32-bit driver file names is `iv`. The prefix for all 64-bit driver file names is `dd`. The file extension is `.dll`, which indicates dynamic link libraries. For example, the 32-bit Salesforce driver file name is `ivsfr cn .dll`, where nn is the revision number of the driver.

For the 8.0 version of the 32-bit driver, the file name is:

```
ivsfrc28.dll
```

For the 8.0 version of the 64-bit driver, the file name is:

```
ddsfrc28.dll
```

Refer to the readme file shipped with the product for a complete list of installed files.

Support for UNIX and Linux environments

UNIX[®]

The following are requirements for the 32- and 64-bit drivers on UNIX/Linux operating systems.

32-bit driver

- All required network software that is supplied by your database system vendors must be 32-bit compliant.
- If your application was built with 32-bit system libraries, you must use 32-bit drivers. If your application was built with 64-bit system libraries, you must use 64-bit drivers (see "64-bit driver requirements"). The database to which you are connecting can be either 32-bit or 64-bit enabled.
- For the Salesforce driver: A 32-bit Java Virtual Machine (JVM), Java SE 8 or higher, is required. Also, you must set the library path environment variable of your operating system to the directory containing your JVM's `libjvm.so` [a] file and that directory's parent directory.

The library path environment variable is:

- `LD_LIBRARY_PATH` on Oracle Solaris, Linux, and HP-UX Itanium
- `LIBPATH` on AIX

See "Driver requirements" for more details on JVM.

AIX

- IBM POWER processor
- AIX 5L operating system, version 5.3 fixpack 5 and higher, 6.1, and 7.1
- An application compatible with components that were built using Visual Age C++ 6.0.0.0 and the AIX native threading model
- IBM SDK, JAVA Technology Edition, Version 6 Service Refresh 9 or higher
- Before you can use the driver, you must set the `LIBPATH` environment variable to include the paths containing the `libjvm.so` library and the `libnio.so` library, which are installed in a subdirectory of your Java Development Kit (JDK). For example, you would add the following paths for Java 8 installed in the `/usr` directory:

```
/usr/java8/jre/lib/amd64/server:/usr/java8/jre/lib/amd64
```

In this example, `/usr/java8/jre/lib/amd64/server` is the location of `libjvm.so`, while `/usr/java8/jre/lib/amd64` is the location of `libnio.so`.

Note: The driver is compiled using the `-brtl` loader option. Your application must support runtime linking functionality.

HP-UX

- Intel Itanium II (IPF) processor is supported.
- HP-UX IPF 11i Versions 2 and 3 (B.11.23 and B.11.3x) are supported.
- An application compatible with components that were built using HP aC++ 5.36 and the HP-UX 11 native (kernel) threading model (posix draft 10 threads).

Note:

- Do not link with the `-lc` linker option.
 - Set the `LD_PRELOAD` environment variable to the `libjvm.so` from your JVM installation.
-

Linux

- The following processors are supported:
 - x86: Intel
 - x64: Intel and AMD
- The following operating systems are supported:
 - CentOS Linux 4.x, 5.x, 6.x, and 7.x
 - Debian Linux 7.11, 8.5
 - Oracle Linux 4.x, 5.x, 6.x, and 7.x
 - Red Hat Enterprise Linux 4.x, 5.x, 6.x, and 7.x
 - SUSE Linux Enterprise Server 10.x, and 11.x
 - Ubuntu Linux 14.04, 16.04
- An application compatible with components that were built using g++ GNU project C++ Compiler version 3.4.6 and the Linux native pthread threading model (Linuxthreads).

Oracle Solaris

- x64: Intel and AMD processor is supported.
- Oracle Solaris 10 and 11.x operating systems are supported.
- An application compatible with components that were built using Oracle C++ 5.8 and the Solaris native (kernel) threading model

See also

[64-bit driver](#) on page 22

[Driver requirements](#) on page 15

64-bit driver

All required network software that is supplied by your database system vendors must be 64-bit compliant.

- A 64-bit Java Virtual Machine (JVM), Java SE 8 or higher, is required. Also, you must set the library path environment variable of your operating system to the directory containing your JVM's `libjvm.so` [a] file and that directory's parent directory.

The library path environment variable is:

- `LD_LIBRARY_PATH` on Linux, HP-UX Itanium, and Oracle Solaris
- `LIBPATH` on AIX

See "Driver requirements" for more details on JVM.

AIX

- IBM POWER Processor
- AIX 5L operating system, version version 5.3 fixpack 5 and higher, 6.1, and 7.1
- An application compatible with components that were built using Visual Age C++ version 6.0.0.0 and the AIX native threading model
- IBM SDK, JAVA Technology Edition, Version 6 Service Refresh 9 or higher
- Before you can use the driver, you must set the `LIBPATH` environment variable to include the paths containing the `libjvm.so` library and the `libnio.so` library, which are installed in a subdirectory of your Java Development Kit (JDK). For example, you would add the following paths for Java 8 installed in the `/usr` directory:

```
:/usr/java8_64/jre/lib/ppc64/classic:/usr/java8_64/jre/lib/ppc64
```

In this example, `/usr/java8_64/jre/lib/ppc64/classic` is the location of `libjvm.so`, while `/usr/java8_64/jre/lib/ppc64` is the location of `libnio.so`.

Note: The driver is compiled using the `-brtl` loader option. Your application must support runtime linking functionality.

HP-UX

- HP-UX IPF 11i operating system, Versions 2 and 3 (B.11.23 and B.11.31)
- HP aC++ v. 5.36 and the HP-UX 11 native (kernel) threading model (posix draft 10 threads)

Note: Do not link with the `-lc` linker option.

Note: Set the `LD_PRELOAD` environment variable to the `libjvm.so` of your JVM installation.

Linux

- The following processors are supported:
 - x64: Intel and AMD
 - CentOS Linux 4.x, 5.x, 6.x, and 7.x
 - Debian Linux 7.11 and 8.5

- Oracle Linux 4.x, 5.x, 6.x, and 7.x
- Red Hat Enterprise Linux AS, ES, and WS version 4.x, 5.x, 6.x, and 7.x
- SUSE Linux Enterprise Server 10.x, and 11.x
- Ubuntu Linux 14.04 and 16.04
- For x64:
 - SUSE Linux Enterprise Server 10.x, 11, and 12
- An application compatible with components that were built using g++ GNU project C++ Compiler version 3.4 and the Linux native pthread threading model (Linuxthreads)

Oracle Solaris

- The following processors are supported:
 - Oracle SPARC
 - x64: Intel and AMD
- The following operating systems are supported:
 - For Oracle SPARC: Oracle Solaris 8, 9, 10, and 11.x
 - For x64: Oracle Solaris 10 and Oracle Solaris 11.x Express
- For Oracle SPARC: An application compatible with components that were built using Sun Studio 11, C++ compiler version 5.8 and the Solaris native (kernel) threading model
- For x64: An application compatible with components that were built using Sun C++ Compiler version 5.8 and the Solaris native (kernel) threading model

See also

[Driver requirements](#) on page 15

AIX

If you are building 64-bit binaries, you must pass the define ODBC64. The example Application provides a demonstration of this. See the installed file `example.txt` for details.

You must also include the correct compiler switches if you are building 64-bit binaries. For instance, to build example, you would use:

```
xlC_r -DODBC64 -q64 -qlonglong -qlongdouble -qvftable -o example
-I../include example.c -L../lib -lc_r -lC_r -lodbc
```

HP-UX 11 aCC

The ODBC drivers require certain runtime library patches. The patch numbers are listed in the readme file for your product. HP-UX patches are publicly available from the HP Web site <http://www.hp.com>.

HP updates the patch database regularly; therefore, the patch numbers in the readme file may be superseded by newer versions. If you search for the specified patch on an HP site and receive a message that the patch has been superseded, download and install the replacement patch.

If you are building 64-bit binaries, you must pass the define ODBC64. The example Application provides a demonstration of this. See the installed file `example.txt` for details. You must also include the +DD64 compiler switch if you are building 64-bit binaries. For instance, to build `example`, you would use:

```
aCC -Wl,+s +DD64 -DODBC64 -o example -I../include example.c -L../lib -lodbc
```

Linux

If you are building 64-bit binaries, you must pass the define ODBC64. The example Application provides a demonstration of this. Refer to the installed file `example.txt` for details.

You must also include the correct compiler switches if you are building 64-bit binaries. For instance, to build `example`, you would use:

```
g++ -o example -DODBC64 -I../include example.c -L../lib -lodbc -lodbcinst -lc
```

Oracle Solaris

If you are building 64-bit binaries, you must pass the define ODBC64. The example Application provides a demonstration of this. See the installed file `example.txt` for details.

You must also include the `-xarch=v9` compiler switch if you are building 64-bit binaries. For instance, to build `example`, you would use:

```
CC -mt -DODBC64 -xarch=v9 -o example -I../include example.c -L../lib -lodbc -lCrun
```

Setup of the environment and the drivers

On UNIX and Linux, several environment variables and the system information file must be configured before the drivers can be used. See the following topics for additional information:

- "Configuring and Connecting on UNIX and Linux" contains a brief description of these variables.
- "Configuring and Connection to Data Sources" provides details about driver configuration.
- "Configuring the Product on UNIX/Linux" provides complete information about using the drivers on UNIX and Linux.

See also

[Configuring and connecting on UNIX and Linux](#) on page 31

[Configuring and connecting to data sources](#) on page 44

[Configuring the product on UNIX/Linux](#) on page 44

Driver names for UNIX/Linux

The drivers are ODBC API-compliant dynamic link libraries, referred to in UNIX and Linux as shared objects. The prefix for all 32-bit driver file names is `iv`. The prefix for all 64-bit driver file names is `dd`. The driver file names are lowercase and the extension is `.so`, the standard form for a shared object. For example, the 32-bit driver file name is `ivsfrcnn.so`, where `nn` is the revision number of the driver.

For the 8.0 version of the 32-bit driver, the file name is:

```
ivsfrc28.so
```


For the 8.0 version of the 64-bit driver, the file name is:

```
ddsfrc28.so
```

Refer to the readme file shipped with the product for a complete list of installed files.

Mapping objects to tables

The driver automatically maps Salesforce objects and fields to tables and columns the first time it connects to a Salesforce instance. The driver maps both standard and custom objects and includes any relationships defined between objects. You can use the `SQLForeignKeys` and `SQLPrimaryKeys` functions to report relationships among objects.

Schema Map

The driver uses a local schema map to instantiate the mapping of the remote Salesforce data model as tables and the metadata associated with those tables. By default, the driver creates a schema map for each user. The schema map is created in each user's application data folder and uses the user ID specified for the connection as the name of the schema map file. If the user ID contains punctuation or other non-alphanumeric characters, the driver strips those characters from the user ID to form the name of the schema map file. The driver includes a `SchemaMap` connection option that allows you to override the default setting for the name and location of the schema map file.

Identifiers

When mapping the remote data model, the driver converts unquoted identifiers to uppercase by default. You can use the `UppercaseIdentifiers` configuration option to map identifiers to use the case of the remote object.

Audit Columns

Salesforce adds audit fields to all the objects defined in a Salesforce instance. By default, the driver includes corresponding audit columns in table definitions when mapping the remote data model. The `AuditColumns` configuration option can be used to exclude or limit audit columns.

Custom Objects and Fields

Salesforce appends custom objects and fields with a standard `"__c"` suffix. By default, the driver includes the standard `"__c"` suffix when mapping the remote data model. You can use the `CustomSuffix` configuration option to strip the `"__c"` suffix.

System Fields

Salesforce includes system fields in a Salesforce instance. By default, the driver uses the name of the system field when mapping the system fields as columns. You can use the `MapSystemColumnNames` configuration option to make it evident that the columns in a table correspond to system fields.

See also

[Schema Map](#) on page 172

[Using identifiers](#) on page 106

[Config Options](#) on page 139

[UppercaseIdentifiers \(Config Option\)](#) on page 145

[AuditColumns \(Config Option\)](#) on page 140

[CustomSuffix \(Config Option\)](#) on page 141

[MapSystemColumnNames \(Config Option\)](#) on page 143

Data types

The following table lists the data types supported by the Salesforce driver for local tables, how the Salesforce data types exposed by the Salesforce Web Service API map to them, and how the Salesforce Web Service API data types map to the ODBC data types.

Table 2: Salesforce Data Types for Local Tables

Salesforce Data Type	Web Service API Data Type	ODBC Data Type
ANYTYPE	anytype	SQL_WVARCHAR
AUTONUMBER	string	SQL_WVARCHAR
BINARY	binary	SQL_LONGVARBINARY
CHECKBOX	boolean	SQL_BIT
COMBOBOX	combobox	SQL_WVARCHAR
CURRENCY	currency	SQL_DECIMAL
DATE	date	SQL_TYPE_DATE
DATETIME	datetime	SQL_TYPE_TIMESTAMP
EMAIL	email	SQL_WVARCHAR
ENCRYPTEDTEXT	encryptedtext	SQL_WVARCHAR
HTML	html	SQL_WLONGVARCHAR
ID	id	SQL_WVARCHAR
INT	double	SQL_INTEGER
LONGTEXTAREA	longtextarea	SQL_WLONGVARCHAR
MULTISELECTPICKLIST	multipicklist	SQL_WVARCHAR
NUMBER	double	SQL_DOUBLE ¹
PERCENT	percent	SQL_DECIMAL
PHONE	phone	SQL_WVARCHAR
PICKLIST	picklist	SQL_WVARCHAR

¹ When the NumberFieldMapping config option is set to emulateInteger, which is the default value, the driver maps NUMBER fields with a precision of 9 or less and a scale of 0 to the INTEGER SQL type and maps all other NUMBER fields to the DOUBLE SQL type.

Salesforce Data Type	Web Service API Data Type	ODBC Data Type
REFERENCE	reference	SQL_WVARCHAR
TEXT	string	SQL_WVARCHAR
TEXTAREA	textarea	SQL_WVARCHAR
TIME	time	SQL_TYPE_TIME
URL	url	SQL_WVARCHAR

The following table lists the data types supported by the Salesforce driver for remote tables, how the Salesforce data types exposed by the Salesforce Web Service API map to them, and how the Salesforce Web Service API data types map to the ODBC data types.

Table 3: Salesforce Data Types for Remote Tables

Salesforce Data Type	Web Service API Data Type	ODBC Data Type
ANYTYPE	anytype	SQL_WVARCHAR
AUTONUMBER	string	SQL_WVARCHAR
BINARY	binary	SQL_LONGVARBINARY
CHECKBOX	boolean	SQL_BIT
COMBOBOX	combobox	SQL_WVARCHAR
CURRENCY	currency	SQL_DOUBLE
DATACATEGORYGROUPREFERENCE	DataCategoryGroupReference	SQL_WVARCHAR
DATE	date	SQL_TYPE_DATE
DATETIME	datetime	SQL_TYPE_TIMESTAMP
EMAIL	email	SQL_WVARCHAR
HTML	html	SQL_WLONGVARCHAR
ID	id	SQL_WVARCHAR
INT	double	SQL_INTEGER
LONGTEXTAREA	longtextarea	SQL_WLONGVARCHAR
MULTISELECTPICKLIST	multipicklist	SQL_WVARCHAR
NUMBER	double	SQL_DOUBLE ¹
PERCENT	percent	SQL_DOUBLE

Salesforce Data Type	Web Service API Data Type	ODBC Data Type
PHONE	phone	SQL_WVARCHAR
PICKLIST	picklist	SQL_WVARCHAR
REFERENCE	reference	SQL_WVARCHAR
TEXT	string	SQL_WVARCHAR
TEXTAREA	textarea	SQL_WVARCHAR
TIME	time	SQL_TYPE_TIME
URL	url	SQL_WVARCHAR

Contacting Technical Support

Progress DataDirect offers a variety of options to meet your support needs. Please visit our Web site for more details and for contact information:

<https://www.progress.com/support>

The Progress DataDirect Web site provides the latest support information through our global service network. The SupportLink program provides access to support contact details, tools, patches, and valuable information, including a list of FAQs for each product. In addition, you can search our Knowledgebase for technical bulletins and other information.

When you contact us for assistance, please provide the following information:

- Your number or the serial number that corresponds to the product for which you are seeking support, or a case number if you have been provided one for your issue. If you do not have a SupportLink contract, the SupportLink representative assisting you will connect you with our Sales team.
- Your name, phone number, email address, and organization. For a first-time call, you may be asked for full information, including location.
- The Progress DataDirect product and the version that you are using.
- The type and version of the operating system where you have installed your product.
- Any database, database version, third-party software, or other environment information required to understand the problem.
- A brief description of the problem, including, but not limited to, any error messages you have received, what steps you followed prior to the initial occurrence of the problem, any trace logs capturing the issue, and so on. Depending on the complexity of the problem, you may be asked to submit an example or reproducible application so that the issue can be re-created.
- A description of what you have attempted to resolve the issue. If you have researched your issue on Web search engines, our Knowledgebase, or have tested additional configurations, applications, or other vendor products, you will want to carefully note everything you have already attempted.
- A simple assessment of how the severity of the issue is impacting your organization.

Getting started

This chapter provides basic information about configuring your driver immediately after installation, testing your connection, and accessing your data with some commonly used third-party applications. To take full advantage of the features of the driver, see "Using the driver".

Information that the driver needs to connect to a database is stored in a *data source*. The ODBC specification describes three types of data sources: user data sources, system data sources (not a valid type on UNIX/Linux), and file data sources. On Windows, user and system data sources are stored in the registry of the local computer. The difference is that only a specific user can access user data sources, whereas any user of the machine can access system data sources. On all platforms, file data sources, which are simply text files, can be stored locally or on a network computer, and are accessible to other machines.

When you define and configure a data source, you store default connection values for the driver that are used each time you connect to a particular database. You can change these defaults by modifying the data source.

For details, see the following topics:

- [Configuring and connecting on Windows](#)
- [Configuring and connecting on UNIX and Linux](#)

Configuring and connecting on Windows



The following basic information enables you to configure a data source and test connect with a driver immediately after installation. On Windows, you can configure and modify data sources through the ODBC Administrator using a driver Setup dialog box. Default connection values are specified through the options on the tabs of the Setup dialog box and are stored either as a user or system data source in the Windows Registry, or as a file data source in a specified location.

Setting the library path environment variable (Windows)

Before you can use your driver, you must set the PATH environment variable to include the path of the `jvm.dll` file of your Java™ Virtual Machine (JVM).

Note: The installer program sets the PATH environment variable to include the path of the `jvm.dll` file by default.

Configuring a data source

To configure a data source:

1. From the Progress DataDirect program group, start the ODBC Administrator and click either the **User DSN**, **System DSN**, or **File DSN** tab to display a list of data sources.

- **User DSN:** If you installed a default DataDirect ODBC user data source as part of the installation, select the appropriate data source name and click **Configure** to display the driver Setup dialog box.

If you are configuring a new user data source, click **Add** to display a list of installed drivers. Select the appropriate driver and click **Finish** to display the driver Setup dialog box.

- **System DSN:** To configure a new system data source, click **Add** to display a list of installed drivers. Select the appropriate driver and click **Finish** to display the driver Setup dialog box.
- **File DSN:** To configure a new file data source, click **Add** to display a list of installed drivers. Select the driver and click **Advanced** to specify attributes; otherwise, click **Next** to proceed. Specify a name for the data source and click **Next**. Verify the data source information; then, click **Finish** to display the driver Setup dialog box.

The General tab of the Setup dialog box appears by default.

Note: The General tab displays only fields that are required for creating a data source. The fields on all other tabs are optional, unless noted otherwise in this book.

2. On the General tab, provide the following information; then, click **Apply**.

Host Name: Type the URL or IP address of the Salesforce instance to which you want to connect. The default is `login.salesforce.com`.

User Name: Type your logon ID, including domain, for Salesforce. For example, `jsmith@defcorp.com`.

Security Token: If your Salesforce instance requires a security token, type your Salesforce generated security token. This value is case-sensitive.

Note: You must provide the following information in the logon dialog box.

Password: Type your case-sensitive password for the Salesforce instance. This value is case-sensitive.

Testing the connection

To test the connection:

1. After you have configured the data source, you can click **Test Connect** on the Setup dialog box to attempt to connect to the data source using the connection options specified in the dialog box. The driver returns a message indicating success or failure. A logon dialog box appears as described in "Using a logon dialog box."
2. Supply the requested information in the logon dialog box and click **OK**. Note that the information you enter in the logon dialog box during a test connect is not saved.
 - If the driver can connect, it releases the connection and displays a `Connection Established` message. Click **OK**.
 - If the driver cannot connect because of an incorrect environment or connection value, it displays an appropriate error message. Click **OK**.
3. On the driver Setup dialog box, click **OK**. The values you have specified are saved and are the defaults used when you connect to the data source. You can change these defaults by using the previously described procedure to modify your data source. You can override these defaults by connecting to the data source using a connection string with alternate values. See "Using a connection string" for information about using connection strings.

See also

[Using a connection string](#) on page 76

[Using a logon dialog box](#) on page 77

Configuring and connecting on UNIX and Linux

The following basic information enables you to configure a data source and test connect with a driver immediately after installation. See "Configuring and connecting to data sources" for detailed information about configuring the UNIX and Linux environments and data sources.

Note: In the following examples, `xx` in a driver filename represents the driver level number.

See also

[Configuring and connecting to data sources](#) on page 44

Environment configuration

To configure the environment:

1. Check your permissions: You must log in as a user with full r/w/x permissions recursively on the entire product installation directory.
2. From your login shell, determine which shell you are running by executing:

```
echo $SHELL
```

3. Run one of the following product setup scripts from the installation directory to set variables: `odbc.sh` or `odbc.csh`. For Korn, Bourne, and equivalent shells, execute `odbc.sh`. For a C shell, execute `odbc.csh`. After running the setup script, execute:

```
env
```

to verify that the `installation_directory/lib` directory has been added to your shared library path.

4. Set the ODBCINI environment variable. The variable must point to the path from the root directory to the system information file where your data source resides. The system information file can have any name, but the product is installed with a default file called `odbc.ini` in the product installation directory. For example, if you use an installation directory of `/opt/odbc` and the default system information file, from the Korn or Bourne shell, you would enter:

```
ODBCINI=/opt/odbc/odbc.ini; export ODBCINI
```

From the C shell, you would enter:

```
setenv ODBCINI /opt/odbc/odbc.ini
```

Test loading the driver

The `ivtestlib` (32-bit drivers) and `ddtestlib` (64-bit drivers) test loading tools are provided to test load drivers and help diagnose configuration problems in the UNIX and Linux environments, such as environment variables not correctly set or missing database client components. This tool is installed in the `/bin` subdirectory in the product installation directory. It attempts to load a specified ODBC driver and prints out all available error information if the load fails.

For example, if the drivers are installed in `/opt/odbc/lib`, the following command attempts to load the 32-bit driver on Linux, where `xx` represents the version number of the driver:

```
ivtestlib /opt/odbc/lib/ivsfrcxx.so
```

Note: On Solaris, AIX, and Linux, the full path to the driver does not have to be specified for the tool. The HP-UX version, however, requires the full path.

If the load is successful, the tool returns a success message along with the version string of the driver. If the driver cannot be loaded, the tool returns an error message explaining why.

Setting the library path environment variable (UNIX and Linux)

Before you can use the Salesforce driver, you must set the library path environment variable for your UNIX/Linux operating system to include the directory containing your JVM's `libjvm.so` [a] file, and that directory's parent directory.

NOTE FOR HP-UX: You also must set the `LD_PRELOAD` environment variable to the fully qualified path of the `libjvm.so`.

32-bit Driver: Library Path Environment Variable

Set the library path environment variable to include the directory containing your 32-bit JVM's `libjvm.so` [a] file, and that directory's parent directory.

- `LD_LIBRARY_PATH` on Solaris, Linux, and HP-UX (Itanium)
- `LIBPATH` on AIX

64-bit Driver: Library Path Environment Variable

Set the library path environment variable to include the directory containing your 64-bit JVM's `libjvm.so` [a] file, and that directory's parent directory.

- `LD_LIBRARY_PATH` on Solaris, HP-UX (Itanium), and Linux
- `LIBPATH` on AIX

Configuring a data source

The default `odbc.ini` file installed in the installation directory is a template in which you create data source definitions. You enter your site-specific database connection information using a text editor. Each data source definition must include the keyword `Driver=`, which is the full path to the driver.

The following examples show the minimum connection string options that must be set to complete a test connection, where `xx` represents `iv` for 32-bit or `dd` for 64-bit drivers, and `yy` represents the extension. The values for the options are samples and are not necessarily the ones you would use.

```
[ODBC Data Sources]
Salesforce=DataDirect 8.0 Salesforce

[Salesforce]
Driver=ODBCHOME/lib/xxsfrc28.yy
HostName=test.salesforce.com
UserName=JohnDoe
Password=secret
SecurityToken=XaBARTsLZReM4Px47qPLOS
```

Connection option descriptions:

- `HostName`: The URL or IP address of the Salesforce instance to which you want to connect. The default is `login.salesforce.com`.
- `UserName`: Your logon ID for Salesforce
- `Password`: Your case-sensitive password for the Salesforce instance.
- `SecurityToken`: If required by your instance, your case-sensitive security token for the Salesforce instance.

Testing the connection

The driver installation includes an ODBC application called `example` that can be used to connect to a data source and execute SQL. The application is located in the `installation_directory/samples/example` directory.

To run the program after setting up a data source in the `odbc.ini`, enter `example` and follow the prompts to enter your data source name, user name, and password. If successful, a `SQL>` prompt appears and you can type in SQL statements such as `SELECT * FROM table`. If `example` is unable to connect, the appropriate error message is returned.

Tutorials

The following sections guide you through using the driver to access your data with some common third-party applications:

- [Accessing data in Tableau \(Windows only\)](#) on page 35
- [Accessing data in Microsoft Excel \(Windows only\)](#) on page 38
- [Accessing data in Microsoft Excel from the Query Wizard \(Windows only\)](#) on page 40


For details, see the following topics:

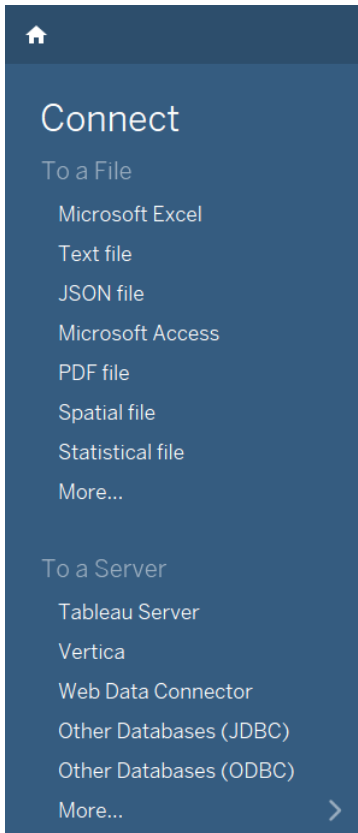
- [Accessing data in Tableau \(Windows only\)](#)
- [Accessing data in Microsoft Excel \(Windows only\)](#)
- [Accessing data in Microsoft Excel from the Query Wizard \(Windows only\)](#)

Accessing data in Tableau (Windows only)

After you have configured your data source, you can use the driver to access your Salesforce data with Tableau. Tableau is a business intelligence software program that allows you to easily create reports and visualized representations of your data. By using the driver with Tableau, you can improve performance when retrieving data while leveraging the driver's relational mapping tools.

To use the driver to access data with Tableau:

1. Navigate to the `\tools\Tableau` subdirectory of the Progress DataDirect installation directory; then, locate the Tableau data source file, `DataDirect Salesforce.tdc`.
2. Copy the `DataDirect Salesforce.tdc` into the following directory:
`C:\Users\user_name\Documents\My Tableau Repository\Datasources`
3. Open Tableau. If the **Connect** menu does not open by default, select **Data > New Data Source** or the Add New Data Source button  to open the menu.



4. From the **Connect** menu, select **Other Databases (ODBC)**.
5. The **Server Connection** dialog appears.

Other Databases (ODBC)

Connect Using

Generic ODBC requires additional configuration for publishing to succeed. Select DSN (data source name) for cross-platform portability. A DSN with the same name must be configured on Tableau Server.

DSN: My DSN

Driver:

Connect

Connection Attributes

Server: Port:

Database:

Username:

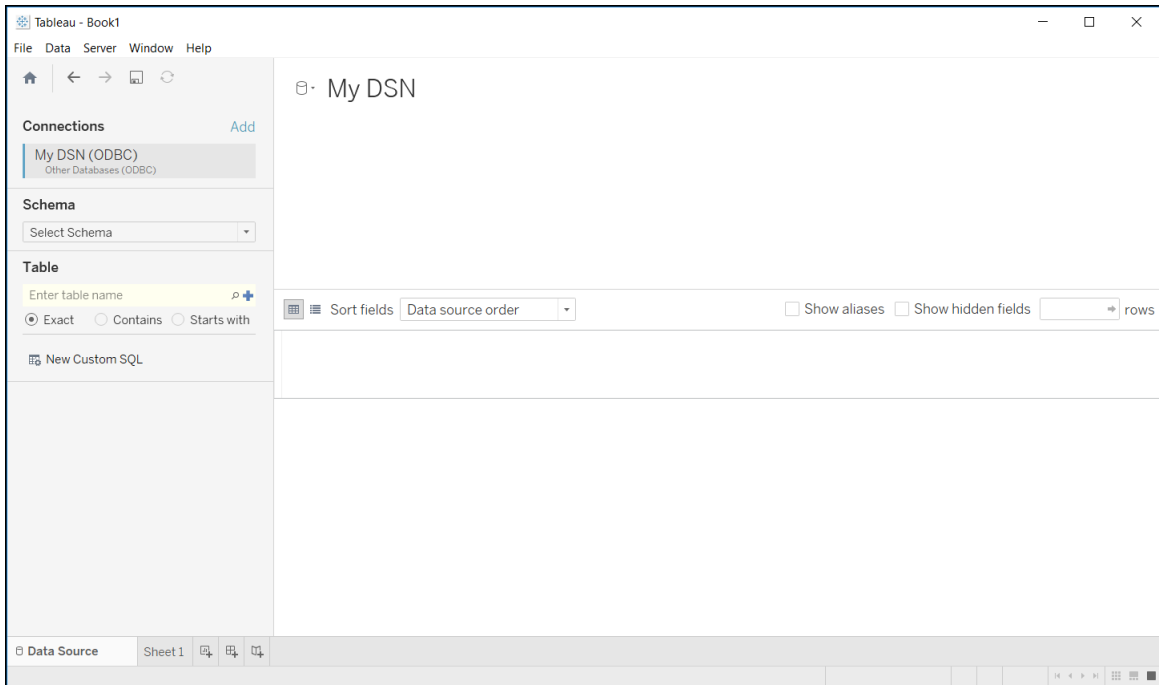
Password:

String Extras:

Sign In

In the DSN field, select the data source you want to use from the drop down menu. For example, **My DSN**. Then, click **Connect**. The Logon to Salesforce dialog appears pre-populated with the connection information you provided in your data source.

6. If required, type your user name and password; then, click **OK**. The Logon dialog closes. Then, click **OK** on the Server Connection dialog.
7. The **Data Source** window appears.



By default, Tableau connects live, or directly, to your data. We recommend that you use the default settings to avoid extracting all of your data. However, if you prefer, you can import your data by selecting the **Extract** option at the top of the dialog.

8. In the Schema field, select the database you want to use. The tables stored in this database are now available for selection in the Table field.

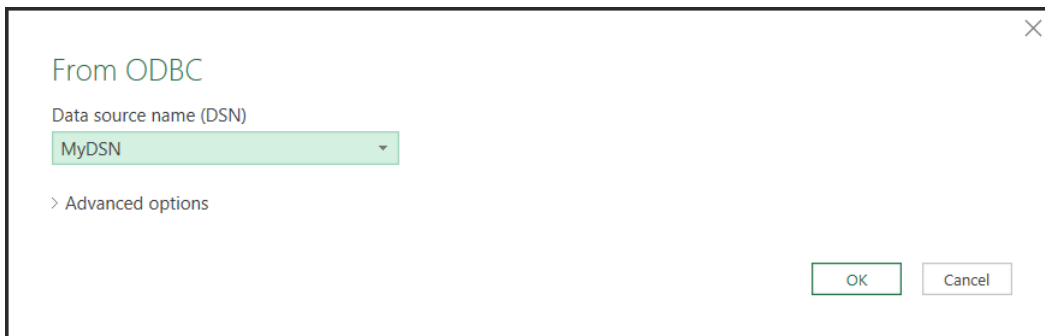
You have successfully accessed your data and are now ready to create reports with Tableau. For more information, refer to the Tableau product documentation at: <http://www.tableau.com/support/help>.

Accessing data in Microsoft Excel (Windows only)

After you have configured your data source, you can use the driver to access your data with Microsoft Excel from the Data Connection Wizard. Using the driver with Excel provides improved performance when retrieving data, while leveraging the driver's relational-mapping tools.

To use the driver to access data with Excel from the Data Connection Wizard:

1. Open your workbook in Excel.
2. From the **Data** menu, select **Get Data>From Other Sources>From ODBC**.
3. The **From ODBC** dialog appears.

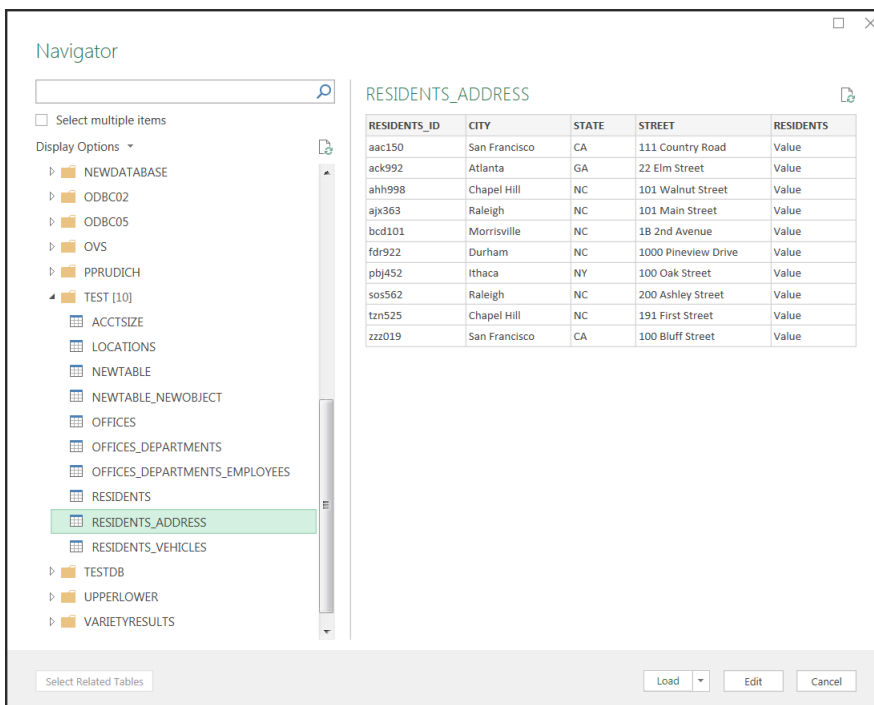


Select your data source from the Data Source Name (DSN) drop down; then, click **OK**.

4. You are prompted for logon credentials for your data source:

- If your data source does not require logon credentials or if you prefer to specify your credentials using a connection string, select **Default or Custom** from the menu on the left. Optionally, specify your credential-related properties using a connection string in the provided field. Click **Connect** to proceed.
- If your data source uses Windows credentials, select **Windows** from the menu; then, provide your credentials. Optionally, specify a connection string with credential-related properties in the provided field. Click **Connect** to proceed.
- If your data source uses credentials stored on the database, select **Database**; then, provide your user name and password. Optionally, specify a connection string in the provided field. Click **Connect** to proceed.

5. The **Navigator** window appears.

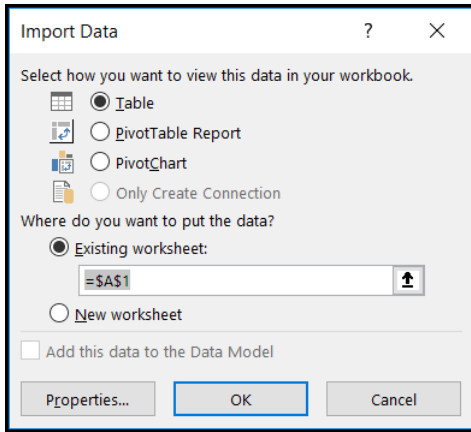


From the list, select the tables you want to access. A preview of your data will appear in the pane on the right. Optionally, click **Edit** to modify the results using the Query Editor. Refer to the Microsoft Excel product documentation for detailed information on using the Query Editor.

6. Load your data:

- Click **Load** to import your data into your work sheet. Skip to the end.
- Click **Load>Load To** to specify a location to import your data. Proceed to the next step.

7. The **Import Data** window appears.



Select the desired view and insertion point for the data. Click **OK**.

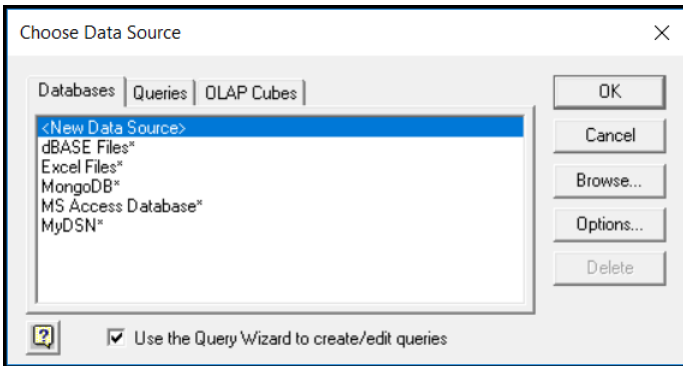
You have successfully accessed your data in Excel. For more information, refer to the Microsoft Excel product documentation at: <https://support.office.com/>.

Accessing data in Microsoft Excel from the Query Wizard (Windows only)

After you have configured your data source, you can use the driver to access your data with Microsoft Excel from the Query Wizard. Using the driver with Excel provides improved performance when retrieving data, while leveraging the driver's relational-mapping tools.

To use the driver to access data with Excel from the Query Wizard:

1. Open your workbook in Excel.
2. From the **Data** menu, select **Get Data>From Other Sources>From Microsoft Query**.
3. The **Choose Data Source** dialog appears.

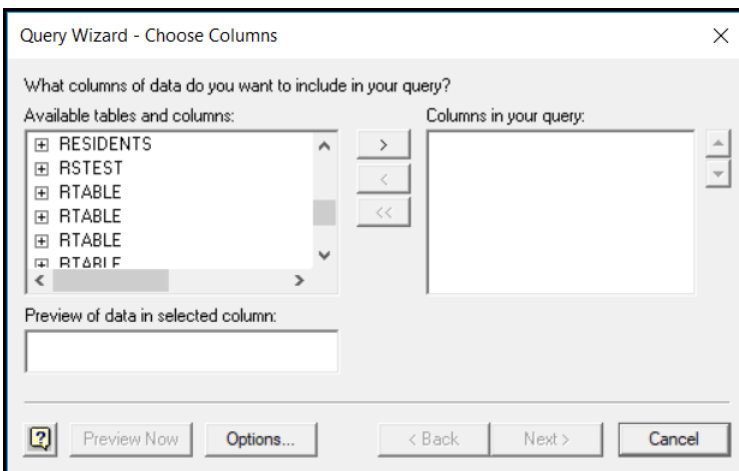


From the Databases list, select your data source. For example, **MyDSN**. Click **OK**.

- The logon dialog appears pre-populated with the connection information you provided in your data source. If required, type your password. Click **OK** to proceed.

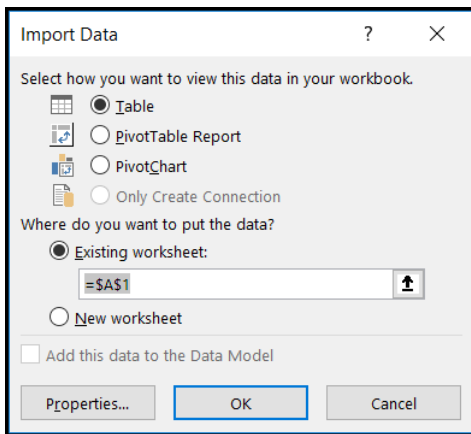
Note: The logon dialog may reappear if Excel needs to access additional information from the data source. If this occurs, re-enter your password; then, click **OK** to proceed to the next step.

- The **Query Wizard - Choose Columns** window appears.



Choose the columns you want to import into your workbook. To add a column, select the column name in Available tables and columns pane; then, click the **>** button. After you add the columns you want to include, click **Next** to continue.

- Optionally, filter your data using the drop-down menus; then, click **Next**.
- Optionally, sort your data using the drop-down menus; then, click **Next**.
- Select "Return Data to Microsoft Excel"; then, click **Finish**.
- The **Import Data** window appears.



Select the desired view and insertion point for your data. Click **OK**.

You have successfully accessed your data in Excel using the Query Wizard. For more information, refer to the Microsoft Excel product documentation at: <https://support.office.com/>.

Using the driver

This section guides you through the configuring and connecting to data sources. In addition, it explains how to use the functionality supported by your driver.

For details, see the following topics:

- [Configuring and connecting to data sources](#)
- [Connecting through a proxy server](#)
- [Performance considerations](#)
- [Using the SQL engine server](#)
- [Client-side caches](#)
- [Catalog tables](#)
- [Reports](#)
- [Using security](#)
- [Using DataDirect Connection Pooling](#)
- [Using identifiers](#)
- [Timeouts](#)
- [Views and remote/local tables](#)
- [SQL support](#)
- [Isolation and lock levels supported](#)

- [Unicode support](#)
- [Parameter metadata support](#)
- [DataDirect Bulk Load](#)
- [Using Bulk API with SQL statements](#)

Configuring and connecting to data sources

After you install the driver, you configure data sources to connect to the database. See "Getting started" for an explanation of different types of data sources. The data source contains connection options that allow you to tune the driver for specific performance. If you want to use a data source but need to change some of its values, you can either modify the data source or override its values at connection time through a connection string.

If you choose to use a connection string, you must use specific connection string attributes. See "Using a connection string" for an alphabetical list of driver connection string attributes and their initial default values.

See also

[Getting started](#) on page 29

[Using a connection string](#) on page 76

Configuring the product on UNIX/Linux

UNIX[®]

This chapter contains specific information about using your driver in the UNIX and Linux environments.

See "Environment variables" for additional platform information.

See also

[Environment variables](#) on page 44

Environment variables

The first step in setting up and configuring the driver for use is to set several environment variables. The following procedures require that you have the appropriate permissions to modify your environment and to read, write, and execute various files. You must log in as a user with full r/w/x permissions recursively on the entire Progress DataDirect *for* ODBC installation directory.

Library search path

The library search path variable can be set by executing the appropriate shell script located in the ODBC home directory. From your login shell, determine which shell you are running by executing:

```
echo $SHELL
```

C shell login (and related shell) users must execute the following command before attempting to use ODBC-enabled applications:

```
source ./odbc.csh
```

Bourne shell login (and related shell) users must initialize their environment as follows:

```
. ./odbc.sh
```

Executing these scripts sets the appropriate library search path environment variable:

- `LD_LIBRARY_PATH` on HP-UX IPF, Linux, and Oracle Solaris
- `LIBPATH` on AIX

The library search path environment variable must be set so that the ODBC core components and drivers can be located at the time of execution. After running the setup script, execute:

```
env
```

to verify that the `installation_directory/lib` directory has been added to your shared library path.

ODBCINI

Setup installs in the product installation directory a default system information file, named `odbc.ini`, that contains data sources. See "Data source configuration on UNIX/Linux" for an explanation of the `odbc.ini` file. The system administrator can choose to rename the file and/or move it to another location. In either case, the environment variable `ODBCINI` must be set to point to the fully qualified path name of the `odbc.ini` file.

For example, to point to the location of the file for an installation on `/opt/odbc` in the C shell, you would set this variable as follows:

```
setenv ODBCINI /opt/odbc/odbc.ini
```

In the Bourne or Korn shell, you would set it as:

```
ODBCINI=/opt/odbc/odbc.ini;export ODBCINI
```

As an alternative, you can choose to make the `odbc.ini` file a hidden file and not set the `ODBCINI` variable. In this case, you would need to rename the file to `.odbc.ini` (to make it a hidden file) and move it to the user's `$HOME` directory.

The driver searches for the location of the `odbc.ini` file as follows:

1. The driver checks the `ODBCINI` variable
2. The driver checks `$HOME` for `.odbc.ini`

If the driver does not locate the system information file, it returns an error.

See also

[Data source configuration on UNIX/Linux](#) on page 47

ODBCINST

Setup installs in the product installation directory a default file, named `odbcinst.ini`, for use with DSN-less connections. See "DSN-Less Connections" for an explanation of the `odbcinst.ini` file. The system administrator can choose to rename the file or move it to another location. In either case, the environment variable `ODBCINST` must be set to point to the fully qualified path name of the `odbcinst.ini` file.

For example, to point to the location of the file for an installation on `/opt/odbc` in the C shell, you would set this variable as follows:

```
setenv ODBCINST /opt/odbc/odbcinst.ini
```

In the Bourne or Korn shell, you would set it as:

```
ODBCINST=/opt/odbc/odbcinst.ini;export ODBCINST
```

As an alternative, you can choose to make the `odbcinst.ini` file a hidden file and not set the `ODBCINST` variable. In this case, you would need to rename the file to `.odbcinst.ini` (to make it a hidden file) and move it to the user's `$HOME` directory.

The driver searches for the location of the `odbcinst.ini` file as follows:

1. The driver checks the `ODBCINST` variable
2. The driver checks `$HOME` for `.odbcinst.ini`

If the driver does not locate the `odbcinst.ini` file, it returns an error.

See also

[DSN-less connections](#) on page 51

DD_INSTALLDIR

This variable provides the driver with the location of the product installation directory so that it can access support files. `DD_INSTALLDIR` must be set to point to the fully qualified path name of the installation directory.

For example, to point to the location of the directory for an installation on `/opt/odbc` in the C shell, you would set this variable as follows:

```
setenv DD_INSTALLDIR /opt/odbc
```

In the Bourne or Korn shell, you would set it as:

```
DD_INSTALLDIR=/opt/odbc;export DD_INSTALLDIR
```

The driver searches for the location of the installation directory as follows:

1. The driver checks the `DD_INSTALLDIR` variable
2. The driver checks the `odbc.ini` or the `odbcinst.ini` files for the `InstallDir` keyword (see "Configuration Through the System Information (`odbc.ini`) File" for a description of the `InstallDir` keyword)

If the driver does not locate the installation directory, it returns an error.

The next step is to test load the driver.

See also

[Configuration through the system information \(`odbc.ini`\) file](#) on page 47

The test loading tool

The second step in preparing to use a driver is to test load it.

The `ivtestlib` (32-bit driver) and `ddtestlib` (64-bit driver) test loading tools are provided to test load drivers and help diagnose configuration problems in the UNIX and Linux environments, such as environment variables not correctly set or missing database client components. This tool is installed in the `/bin` subdirectory in the product installation directory. It attempts to load a specified ODBC driver and prints out all available error information if the load fails.

For example, if the driver is installed in `/opt/odbc/lib`, the following command attempts to load the 32-bit driver on Linux, where `xx` represents the version number of the driver:

```
ivtestlib /opt/odbc/lib/ivsfrxx.so
```

Note: On Solaris, AIX, and Linux, the full path to the driver does not have to be specified for the tool. The HP-UX version, however, requires the full path.

If the load is successful, the tool returns a success message along with the version string of the driver. If the driver cannot be loaded, the tool returns an error message explaining why.

See "Version string information" for details about version strings.

The next step is to configure a data source through the system information file.

See also

[Version string information](#) on page 16

Data source configuration on UNIX/Linux

In the UNIX and Linux environments, a system information file is used to store data source information. Setup installs a default version of this file, called `odbc.ini`, in the product installation directory. This is a plain text file that contains data source definitions.

Configuration through the system information (odbc.ini) file

To configure a data source manually, you edit the `odbc.ini` file with a text editor. The content of this file is divided into three sections.

Note: The driver and driver manager support ASCII and UTF-8 encoding in the `odbc.ini` file. For additional details, see "Character encoding in the `odbc.ini` and `odbcinst.ini` files".

At the beginning of the file is a section named `[ODBC Data Sources]` containing `data_source_name=installed-driver` pairs, for example:

```
Salesforce=DataDirect 8.0 Salesforce
```

The driver uses this section to match a data source to the appropriate installed driver.

The `[ODBC Data Sources]` section also includes data source definitions. The default `odbc.ini` contains a data source definition for the driver. Each data source definition begins with a data source name in square brackets, for example, `[Salesforce 2]`. The data source definitions contain connection string `attribute=value` pairs with default values. You can modify these values as appropriate for your system. "Connection option descriptions" describes these attributes. See "Sample default `odbc.ini` file" for sample data sources.

The second section of the file is named `[ODBC File DSN]` and includes one keyword:

```
[ODBC File DSN]
DefaultDSNDir=
```

This keyword defines the path of the default location for file data sources (see "File data sources").

Note: This section is not included in the default `odbc.ini` file that is installed by the product installer. You must add this section manually.

The third section of the file is named [ODBC] and includes several keywords, for example:

```
[ODBC]
IANAAppCodePage=4
InstallDir=/opt/odbc
Trace=0
TraceFile=odctrace.out
TraceDll=/opt/odbc/lib/ivtrc28.so
ODBCTraceMaxFileSize=102400
ODBCTraceMaxNumFiles=10
```

The IANAAppCodePage keyword defines the default value that the UNIX/Linux driver uses if individual data sources have not specified a different value. See "IANAAppCodePage" in "Connection option descriptions". The default value is 4.

For supported code page values, refer to "Code page values" in the *Progress DataDirect for ODBC Drivers Reference*.

The InstallDir keyword must be included in this section. The value of this keyword is the path to the installation directory under which the /lib and /locale directories are contained. The installation process automatically writes your installation directory to the default odbc.ini file.

For example, if you choose an installation location of /opt/odbc, then the following line is written to the [ODBC] section of the default odbc.ini:

```
InstallDir=/opt/odbc
```

Note: If you are using only DSN-less connections through an odbcinst.ini file and do not have an odbc.ini file, then you must provide [ODBC] section information in the [ODBC] section of the odbcinst.ini file. The driver and Driver Manager always check first in the [ODBC] section of an odbc.ini file. If no odbc.ini file exists or if the odbc.ini file does not contain an [ODBC] section, they check for an [ODBC] section in the odbcinst.ini file. See "DSN-less connections" for details.

ODBC tracing allows you to trace calls to the ODBC driver and create a log of the traces for troubleshooting purposes. The following keywords all control tracing: Trace, TraceFile, TraceDLL, ODBCTraceMaxFileSize, and ODBCTraceMaxNumFiles.

For a complete discussion of tracing, refer to "ODBC trace" in the *Progress DataDirect for ODBC Drivers Reference*.

See also

[Connection option descriptions](#) on page 123

[Sample default odbc.ini file](#) on page 49

[File data sources](#) on page 52

[IANAAppCodePage](#) on page 155

[DSN-less connections](#) on page 51

Sample default odbc.ini file

The following is a sample `odbc.ini` file that the installer program installs in the installation directory. All occurrences of `ODBCHOME` are replaced with your installation directory path during installation of the file. Values that you must supply are enclosed by angle brackets (< >). If you are using the installed `odbc.ini` file, you must supply the values and remove the angle brackets before that data source section will operate properly. Commented lines are denoted by the `#` symbol. This sample shows a 32-bit driver with the driver file name beginning with `iv`. A 64-bit driver file would be identical except that driver name would begin with `dd` and the list of data sources would include only the 64-bit drivers.

```
[ODBC Data Sources]
Salesforce=DataDirect 8.0 Salesforce

[Salesforce]
Driver=ODBCHOME/lib/ivsfr28.so
Description=DataDirect 8.0 Salesforce
ApplicationUsingThreads=1
ApplyToLabel=0
BulkFetchThreshold=3000
BulkLoadAsync=0
BulkLoadBatchSize=1024
BulkLoadConcurrencyMode=1
BulkLoadJobSize=150000
BulkLoadPollInterval=10
BulkLoadProtocol=0
BulkLoadThreshold=4000
BulkLoadTimeout=0
BulkLoadVersion=V1
ConnectionReset=0
ConfigOptions=
CreateMap=2
EnableBulkFetch=1
EnableBulkLoad=1
EnablePKChunking=1
ExtendedOptions=
FetchSize=100
HostName=login.salesforce.com
InitializationString=
IANAAppCodePage=
JVMArgs=-Xmx256m
JVMClasspath=
LoadBalanceTimeout=0
LogConfigFile=
LoginTimeout=15
LogonID=
MaxPoolSize=100
MinPoolSize=0
PKChunkSize=100000
Pooling=0
ProxyHost=
ProxyPassword=
ProxyPort=
ProxyUser=
ReadOnly=0
RefreshSchema=0
ReportCodepageConversionErrors=0
SchemaMap=
SecurityToken=
ServerPortNumber=19928
SQLEngineMode=2
StmtCallLimit=100
StmtCallLimitBehavior=2
TransactionMode=0
WSFetchSize=0
WSPoolSize=1
WSRetryCount=0
WSTimeout=120
```

```
[ODBC]
IANAAppCodePage=4
InstallDir=ODBCHOME
Trace=0
TraceFile=odbctrace.out
TraceDll=ODBCHOME/lib/ivtrc28.so
ODBCTraceMaxFileSize=102400
ODBCTraceMaxNumFiles=10
[ODBC File DSN]
DefaultDSNDir=
UseCursorLib=0
```

To modify or create data sources in the `odbc.ini` file, use the following procedures.

- **To modify a data source:**

- a) Using a text editor, open the `odbc.ini` file.
- b) Modify the default attributes in the data source definitions as necessary based on your system specifics, for example, enter the host name and port number of your system in the appropriate location.

Consult the "Salesforce Attributes" table in "Connection option descriptions" for other specific attribute values.
- c) After making all modifications, save the `odbc.ini` file and close the text editor.

Important: The "Connection option descriptions" section lists both the long and short names of the attribute. When entering attribute names into `odbc.ini`, you must use the long name of the attribute. The short name is not valid in the `odbc.ini` file.

- **To create a new data source:**

- a) Using a text editor, open the `odbc.ini` file.
- b) Copy an appropriate existing default data source definition and paste it to another location in the file.
- c) Change the data source name in the copied data source definition to a new name. The data source name is between square brackets at the beginning of the definition, for example, `[Salesforce]`.
- d) Modify the attributes in the new definition as necessary based on your system specifics, for example, enter the host name and port number of your system in the appropriate location.

Consult the "Salesforce Attributes" table in "Connection option descriptions" for other specific attribute values.
- e) In the `[ODBC]` section at the beginning of the file, add a new `data_source_name=installed-driver` pair containing the new data source name and the appropriate installed driver name.
- f) After making all modifications, save the `odbc.ini` file and close the text editor.

Important: The "Salesforce Attributes" table in "Connection option descriptions" lists both the long and short name of the attribute. When entering attribute names into `odbc.ini`, you must use the long name of the attribute. The short name is not valid in the `odbc.ini` file.

See also

[Connection option descriptions](#) on page 123

The example application

Progress DataDirect ships an application, named *example*, that is installed in the `/samples/example` subdirectory of the product installation directory. Once you have configured your environment and data source, use the example application to test passing SQL statements. To run the application, enter `example` and follow the prompts to enter your data source name, user name, and password.

If successful, a `SQL>` prompt appears and you can type in SQL statements, such as `SELECT * FROM table_name`. If *example* is unable to connect to the database, an appropriate error message appears.

Refer to the `example.txt` file in the `example` subdirectory for an explanation of how to build and use this application.

Refer to "The example application" in *Progress DataDirect for ODBC Drivers Reference* for more information.

DSN-less connections

Connections to a data source can be made via a connection string without referring to a data source name (DSN-less connections). This is done by specifying the "DRIVER=" keyword instead of the "DSN=" keyword in a connection string, as outlined in the ODBC specification. A file named `odbcinst.ini` must exist when the driver encounters `DRIVER=` in a connection string.

Setup installs a default version of this file in the product installation directory (see "ODBCINST" for details about relocating and renaming this file). This is a plain text file that contains default DSN-less connection information. You should not normally need to edit this file. The content of this file is divided into several sections.

Note: The driver and driver manager support ASCII and UTF-8 encoding in the `odbcinst.ini` file.

Refer to the "Character encoding in the `odbc.ini` and `odbcinst.ini` files" in *Progress DataDirect for ODBC Drivers Reference* for details.

At the beginning of the file is a section named `[ODBC Drivers]` that lists installed drivers, for example,

```
DataDirect 8.0 Salesforce=Installed
```

This section also includes additional information for each driver.

The final section of the file is named `[ODBC]`. The `[ODBC]` section in the `odbcinst.ini` file fulfills the same purpose in DSN-less connections as the `[ODBC]` section in the `odbc.ini` file does for data source connections. See "Configuration through the system information (`odbc.ini`) file" for a description of the other keywords this section.

Note: The `odbcinst.ini` file and the `odbc.ini` file include an `[ODBC]` section. If the information in these two sections is not the same, the values in the `odbc.ini` `[ODBC]` section override those of the `odbcinst.ini` `[ODBC]` section.

See also

[ODBCINST](#) on page 45

[Configuration through the system information \(`odbc.ini`\) file](#) on page 47

Sample odbcinst.ini file

The following is a sample `odbcinst.ini`. All occurrences of `ODBCHOME` are replaced with your installation directory path during installation of the file. Commented lines are denoted by the `#` symbol. This sample shows a 32-bit driver with the driver file name beginning with `iv`; a 64-bit driver file would be identical except that driver names would begin with `dd`.

```
[ODBC Drivers]
DataDirect 8.0 Salesforce=Installed

[DataDirect 8.0 Salesforce]
Driver=ODBCHOME/lib/ivsfr28.so
JarFile=ODBCHOME/java/lib/sforce.jar
APILevel=0
ConnectFunctions=YYY
CPTimeout=60
DriverODBCVer=3.52
FileUsage=0
HelpRootDirectory=ODBCHOME/Help/SalesforceHelp
SQLLevel=0
UsageCount=1

[ODBC]
#This section must contain values for DSN-less connections
#if no odbc.ini file exists. If an odbc.ini file exists,
#the values from that [ODBC] section are used.

IANAAppCodePage=4
InstallDir=ODBCHOME
Trace=0
TraceFile=odbctrace.out
TraceDll=ODBCHOME/lib/ivtr28.so
ODBCTraceMaxFileSize=102400
ODBCTraceMaxNumFiles=10
```

File data sources

The Driver Manager on UNIX and Linux supports file data sources. The advantage of a file data source is that it can be stored on a server and accessed by other machines, either Windows, UNIX, or Linux. See "Getting started" for a general description of ODBC data sources on both Windows and UNIX.

A file data source is simply a text file that contains connection information. It can be created with a text editor. The file normally has an extension of `.dsn`.

For example, a file data source for the driver would be similar to the following:

```
[ODBC]
Driver=DataDirect 8.0 Salesforce
Port=19937
HostName=login.salesforce.com
LogonID=jsmith@defcorp.com
SchemaMap=~/.progress/datadirect/salesforce_schema/jsmith@defcorp.config
SecurityToken=XaBARTsLZReM4Px47qPLOS
```

It must contain all basic connection information plus any optional attributes. Because it uses the `DRIVER=` keyword, an `odbcinst.ini` file containing the driver location must exist (see "DSN-less connections").

The file data source is accessed by specifying the `FILEDSN=` instead of the `DSN=` keyword in a connection string, as outlined in the ODBC specification. The complete path to the file data source can be specified in the syntax that is normal for the machine on which the file is located. For example, on Windows:

```
FILEDSN=C:\Program Files\Common Files\ODBC\DataSources\Salesforce2.dsn
```

or, on UNIX and Linux:

```
FILEDSN=/home/users/john/filedsn/Salesforce2.dsn
```

If no path is specified for the file data source, the Driver Manager uses the DefaultDSNDir property, which is defined in the [ODBC File DSN] setting in the `odbc.ini` file to locate file data sources (see "Data source configuration on UNIX/Linux" for details). If the [ODBC File DSN] setting is not defined, the Driver Manager uses the InstallDir setting in the [ODBC] section of the `odbc.ini` file. The Driver Manager does not support the `SQLReadFileDSN` and `SQLWriteFileDSN` functions.

As with any connection string, you can specify attributes to override the default values in the data source:

```
FILEDSN=/home/users/john/filedsn/Salesforce2.dsn;UID=james;PWD=test01
```

See also

[Getting started](#) on page 29

[DSN-less connections](#) on page 51

[Getting started](#) on page 29

[Data source configuration on UNIX/Linux](#) on page 47

UTF-16 applications on UNIX and Linux

Because the DataDirect Driver Manager allows applications to use either UTF-8 or UTF-16 Unicode encoding, applications written in UTF-16 for Windows platforms can also be used on UNIX and Linux platforms.

The Driver Manager assumes a default of UTF-8 applications; therefore, two things must occur for it to determine that the application is UTF-16:

- The definition of `SQLWCHAR` in the ODBC header files must be switched from "char *" to "short *". To do this, the application uses `#define SQLWCHARSHORT`.
- The application must set the encoding for the environment or connection using one of the following attributes. If your application passes UTF-8 encoded strings to some connections and UTF-16 encoded strings to other connections in the same environment, encoding should be set for the connection only; otherwise, either method can be used.
 - To configure the encoding for the environment, set the ODBC environment attribute `SQL_ATTR_APP_UNICODE_TYPE` to a value of `SQL_DD_CP_UTF16`, for example:

```
rc = SQLSetEnvAttr(*henv,
SQL_ATTR_APP_UNICODE_TYPE, (SQLPOINTER)SQL_DD_CP_UTF16, SQL_IS_INTEGER);
```

- To configure the encoding for the connection only, set the ODBC connection attribute `SQL_ATTR_APP_UNICODE_TYPE` to a value of `SQL_DD_CP_UTF16`. For example:

```
rc = SQLSetConnectAttr(hdbc, SQL_ATTR_APP_UNICODE_TYPE, SQL_DD_CP_UTF16,
SQL_IS_INTEGER);
```

Data source configuration through a GUI

On Windows, data sources are stored in the Windows Registry. You can configure and modify data sources through the ODBC Administrator using a driver Setup dialog box, as described in this section.

UNIX® On UNIX and Linux, the GUI is not supported.

When the driver is first installed, the values of its connection options are set by default. These values appear on the driver Setup dialog box tabs when you create a new data source. You can change these default values by modifying the data source. In the following procedure, the description of each tab is followed by a table that lists the connection options for that tab and their initial default values. This table links you to a complete description of the options and their connection string attribute equivalents. The connection string attributes are used to override the default values of the data source if you want to change these values at connection time.

To configure a Salesforce data source:

1. Start the ODBC Administrator by selecting its icon from the Progress DataDirect for ODBC program group.
2. Select a tab:

- **User DSN:** If you are configuring an existing user data source, select the data source name and click **Configure** to display the driver Setup dialog box.

If you are configuring a new user data source, click **Add** to display a list of installed drivers. Select the driver and click **Finish** to display the driver Setup dialog box.

- **System DSN:** If you are configuring an existing system data source, select the data source name and click **Configure** to display the driver Setup dialog box.

If you are configuring a new system data source, click **Add** to display a list of installed drivers. Select the driver and click **Finish** to display the driver Setup dialog box.

- **File DSN:** If you are configuring an existing file data source, select the data source file and click **Configure** to display the driver Setup dialog box.

If you are configuring a new file data source, click **Add** to display a list of installed drivers; then, select a driver. Click **Advanced** if you want to specify attributes; otherwise, click **Next** to proceed. Specify a name for the data source and click **Next**. Verify the data source information; then, click **Finish** to display the driver Setup dialog box.

3. The General tab of the Setup dialog box appears by default.

Figure 1: General tab

The screenshot shows the 'ODBC Salesforce Driver Setup' dialog box. The 'General' tab is active, displaying the following fields and controls:

- Data Source Name:** Text box containing 'Salesforce'.
- Description:** Empty text box.
- Host Name:** Text box containing 'login.salesforce.com'.
- User Name:** Empty text box.
- Security Token:** Empty text box.
- Schema Map:** Empty text box with a 'Browse' button to its right.
- Buttons:** 'Help' button next to the Data Source Name field, and 'Test Connect', 'OK', 'Cancel', and 'Apply' buttons at the bottom.

On this tab, provide values for the options in the following table; then, click **Apply**. The table provides links to descriptions of the connection options. The General tab displays fields that are required for creating a data source. The fields on all other tabs are optional, unless noted otherwise.

Connection Options: General	Description
Data Source Name on page 148	Specifies the name of a data source in your Windows Registry or <code>odbc.ini</code> file. Default: None
Description on page 150	Specifies an optional long description of a data source. This description is not used as a runtime connection attribute, but does appear in the <code>ODBC.INI</code> section of the Registry and in the <code>odbc.ini</code> file. Default: None
Host Name on page 155	The base Salesforce URL or IP address of your Salesforce instance. If you are logging into a Salesforce instance other than the default, you must provide the root of the Salesforce URL or IP address. Default: <code>login.salesforce.com</code>
User Name on page 178	The default user ID and domain that is used to connect to your database. For example, <code>jsmith@defcorp.com</code> . Default: None
Security Token on page 173	Specifies the security token required to make a connection to a Salesforce instance that is configured for a security token. Default: Empty string
Schema Map on page 172	Specifies the name and location of the configuration file where the relational map of native data is written. The driver looks for this file when connecting to a Salesforce instance. If the file does not exist, the driver creates one. Default: <code>application_data_folder\Local\Progress\DataDirect\Salesforce Schema\user_name.config</code>

4. At any point during the configuration process, you can click **Test Connect** to attempt to connect to the data source using the connection options specified in the driver Setup dialog box. A logon dialog box appears (see "Using a logon dialog box" for details). Note that the information you enter in the logon dialog box during a test connect is not saved.
5. To further configure your driver, click on the following tabs. The corresponding sections provide details on the fields specific to each configuration tab:
 - [SQL Engine tab](#) allows you to configure the SQL engine's behavior.
 - [Advanced tab](#) allows you to configure advanced behavior.
 - [Web Service tab](#) allows you to configure the behavior of communications between the driver and the web service.
 - [Pooling tab](#) allows you to configure connection pooling behavior.

- [Bulk tab](#) allows you to specify settings for DataDirect Bulk Load.
6. Click **OK**. When you click **OK**, the values you have specified become the defaults when you connect to the data source. You can change these defaults by using this procedure to reconfigure your data source. You can override these defaults by connecting to the data source using a connection string with alternate values.

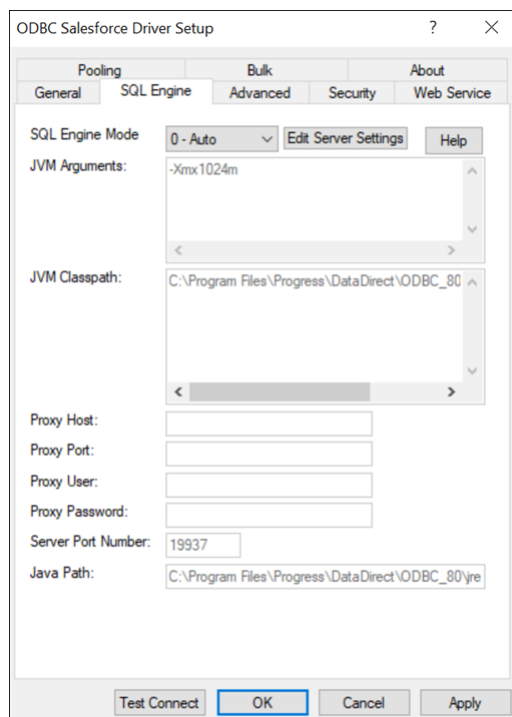
See also

[Using a logon dialog box](#) on page 77

SQL Engine tab

The SQL Engine tab allows you to specify additional data source settings. The fields are optional unless otherwise noted. On this tab, provide values for the options in the following tables; then, click **Apply**.

Figure 2: SQL Engine tab



The SQL engine can be run in one of two modes: direct mode or server mode. When set to direct mode, both the driver and its SQL engine run in the ODBC application's address space. Some applications may experience problems loading the JVM because the process exceeds the available heap space. To avoid this issue, you can configure the driver to operate in server mode. Server mode allows the driver to connect to an SQL engine JVM running as a separate service.

By default, the driver is set to **0 - Auto**. In this setting, the SQL engine attempts to run in server mode first, but will failover to direct mode if server mode is unavailable. If you prefer that the SQL engine runs exclusively in a particular mode, set the SQL Engine Mode option to **1 - Server** to run only in server mode or **2 - Direct** to run only in direct mode.

Table 4: SQL Engine Tab Connection Options

Connection Options: SQL Engine	Default
SQL Engine Mode on page 175	<p>If set to 0 - Auto, the SQL engine attempts to run in server mode first; however, if server mode is unavailable, it runs in direct mode.</p> <p>If set to 1 - Server, the SQL engine runs in server mode. The SQL engine operates in a separate process from the driver within its own JVM. If the SQL engine is unavailable, the connection will fail.</p> <p>If set to 2 - Direct, the SQL engine runs in direct mode. The driver and its SQL engine run in a single process within the same JVM.</p> <hr/> <p>Important: When the SQL engine is configured to run in server mode (0-Auto 1-Server), you must start the SQL engine service before using the driver (see "Starting the SQL engine server" for more information). Multiple drivers on different clients can use the same service.</p> <hr/> <p>Important: Changes you make to the server mode configuration affect all DSNs sharing the service.</p> <hr/> <p>Default: 0 - Auto</p>
JVM Arguments on page 157	<p>A string that contains the arguments that are passed to the JVM that the driver is starting. The location of the JVM must be specified on the driver library path. Values that include special characters or spaces must be enclosed in curly braces { } when used in a connection string.</p> <p>Default:</p> <p>For the 32-bit driver when the SQL Engine Mode is set to 2 - Direct: <code>-Xmx256m</code></p> <p>For all other configurations: <code>-Xmx1024m</code></p>
JVM Classpath on page 158	<p>Specifies the CLASSPATH for the Java Virtual Machine (JVM) used by the driver. The CLASSPATH is the search string the JVM uses to locate the Java jar files the driver needs.</p> <p>Separate multiple jar files by a semi-colon on Windows platforms and by a colon on all other platforms. CLASSPATH values with multiple jar files must be enclosed in curly braces { } when used in a connection string.</p> <hr/> <p>Note: If no value is specified, the driver automatically detects the CLASSPATHs for all ODBC drivers installed on your machine.</p> <hr/> <p>Default: Empty String</p>
Proxy Host on page 165	<p>Specifies the Hostname of the Proxy Server. The value specified can be a host name, a fully qualified domain name, or an IPv4 or IPv6 address.</p> <p>Default: None</p>

Connection Options: SQL Engine	Default
Proxy Port on page 166	Specifies the port number where the Proxy Server is listening for HTTP and/or HTTPS requests. Default: None
Proxy User on page 167	Specifies the user name needed to connect to the Proxy Server. Default: None
Proxy Password on page 166	Specifies the password needed to connect to the Proxy Server. Default: None

When set to **0 - Auto** or **1 - Server**, additional configuration settings that are specific to server mode are exposed in the setup dialog. The settings for server mode are read only in the Driver Setup Dialog. For a description of these settings, see the table below.

To define the settings for server mode, click **Edit Server Settings** from the SQL Engine tab. The SQL Engine Service Setup dialog box appears.

Caution: Modifying the Server Settings will affect all DSNs using this service.

Note: You must be an administrator to modify the server mode settings. Otherwise, the Edit Server Settings button does not appear on the SQL Engine tab.

You use the SQL Engine Service Setup dialog box to configure server mode and to start or stop the service. See "Configuring server mode" for detailed information.

Table 5: Server Mode Configuration Options

Configuration Options: SQL Engine Service	Description
Server Port Number on page 174	Specifies a valid port on which the SQL engine listens for requests from the driver. Default: For the 32-bit driver: 19938 For the 64-bit driver: 19937
Java Path	Specifies fully qualified path to the Java SE 8 or higher JVM executable that you want to use to run the SQL engine server. The path must not contain double quotation marks. Default: The fully qualified path to the Java SE 8 or higher JVM executable (java.exe)

If you finished configuring your driver, proceed to Step 6 in "Configuring the product using the GUI." Optionally, you can further configure your driver by clicking on the following tabs. The following sections provide details for the fields specific to each configuration tab:

- [General tab](#) allows you to configure options that are required for creating a data source.

- [Advanced tab](#) allows you to configure advanced behavior.
- [Web Service tab](#) allows you to configure the behavior of communications between the driver and the web service.
- [Pooling tab](#) allows you to configure connection pooling behavior.
- [Bulk tab](#) allows you to specify settings for DataDirect Bulk Load.

See also

[Starting the SQL engine server](#) on page 82

[Configuring server mode](#) on page 80

[Data source configuration through a GUI](#) on page 53

Advanced tab

The Advanced tab allows you to specify additional data source settings. The fields are optional unless otherwise noted. On this tab, provide values for the options in the following table; then, click **Apply**.

Figure 3: Advanced tab

The screenshot shows the 'ODBC Salesforce Driver Setup' dialog box with the 'Advanced' tab selected. The dialog has a title bar with a question mark and a close button. Below the title bar are tabs for 'Pooling', 'Bulk', and 'About'. Under 'Pooling' are 'General' and 'SQL Engine'. Under 'Bulk' is 'Advanced'. Under 'About' are 'Security' and 'Web Service'. The 'Advanced' tab contains the following settings:

- Create Map:** A dropdown menu set to '2 - NotExist'. A 'Help' button is to its right.
- Transaction Mode:** A dropdown menu set to '0 - No Transactions'. A 'Translate...' button is to its right.
- Application Using Threads:** A checked checkbox.
- Refresh Schema:** An unchecked checkbox.
- Read Only:** An unchecked checkbox.
- Apply ToLabel:** An unchecked checkbox.
- Login Timeout:** A text box containing '15'.
- Fetch Size:** A text box containing '100'.
- Report Codepage Conversion Errors:** A dropdown menu set to '0 - Ignore Errors'.
- Log Config File:** An empty text box.
- Initialization String:** A large text area with up and down arrow buttons on the right.
- Config Options:** A large text area with up and down arrow buttons on the right.
- Extended Options:** A large text area with up and down arrow buttons on the right.

At the bottom of the dialog are four buttons: 'Test Connect', 'OK' (highlighted with a blue border), 'Cancel', and 'Apply'.

Connection Options: Advanced	Description
Create Map on page 148	<p>Determines whether the driver creates a new schema map when establishing the connection.</p> <p>If set to 0 - No, the driver uses the current schema map specified by the Schema Map option. If one does not exist, the connection fails.</p> <p>If set to 1 - ForceNew, the driver deletes the current schema map specified by the Schema Map option and creates a new one at the same location.</p> <p>If set to 2 - NotExist, the driver uses the current schema map specified by the Schema Map option. If one does not exist, the driver creates one.</p> <p>Default: 2 -NotExist</p>
Transaction Mode on page 177	<p>Specifies how the driver handles manual transactions.</p> <p>If set to 1 - Ignore, the data source does not support transactions and the driver always operates in auto-commit mode. Calls to set the driver to manual commit mode and to commit transactions are ignored. Calls to rollback a transaction cause the driver to return an error indicating that no transaction is started. Metadata indicates that the driver supports transactions and the ReadUncommitted transaction isolation level.</p> <p>If set to 0 - No Transactions, the data source and the driver do not support transactions. Metadata indicates that the driver does not support transactions.</p> <p>Default: 0 - No Transactions</p>
Application Using Threads on page 130	<p>Determines whether the driver works with applications using multiple ODBC threads.</p> <p>If set to enabled, the driver works with single-threaded and multi-threaded applications.</p> <p>If set to disabled, the driver does not work with multi-threaded applications. If using the driver with single-threaded applications, this value avoids additional processing required for ODBC thread-safety standards.</p> <p>Default: Enabled</p>
Refresh Schema on page 169	<p>Determines whether the driver automatically refreshes the information in a remote schema (rebuilds the schema map for the schema) the first time a user connects to the specified embedded database.</p> <p>If set to 1 (Enabled), the driver automatically refreshes the schema map the first time a user connects to the specified database.</p> <p>If set to 0 (Disabled), the driver does not automatically refresh the schema map the first time a user connects to the specified database.</p> <p>Default: 0 (Disabled)</p>

Connection Options: Advanced	Description
Read Only on page 167	<p>Specifies whether the connection has read-only access to the data source. If enabled, the connection has read-only access.</p> <p>If disabled, the connection is opened for read/write access, and you can use all commands supported by the product.</p> <p>Default: Enabled</p>
Apply ToLabel on page 130	<p>Determines whether the driver applies the toLabel() function to the picklist fields when executing queries. The toLabel() function translates the result set of a query into the language of the user.</p> <p>If enabled, the driver applies the toLabel() function to the picklist fields when executing queries.</p> <p>If disabled, the driver does not apply the toLabel() function to the picklist fields when executing queries.</p> <p>Default: 0 (Disabled)</p>
Login Timeout on page 161	<p>The number of seconds the driver waits for a connection to be established before returning control to the application and generating a timeout error.</p> <p>If set to -1, the connection request does not time out. The driver silently ignores the SQL_ATTR_LOGIN_TIMEOUT attribute.</p> <p>If set to 0, the connection request does not time out, but the driver responds to the SQL_ATTR_LOGIN_TIMEOUT attribute.</p> <p>If set to x, the connection request times out after the specified number of seconds unless the application overrides this setting with the SQL_ATTR_LOGIN_TIMEOUT attribute.</p> <p>Default: 15</p>
Fetch Size on page 153	<p>Specifies the number of rows that the driver processes before returning data to the application. Smaller fetch sizes can improve the initial response time of the query. Larger fetch sizes improve overall fetch times at the cost of additional memory.</p> <p>Default: 100</p>
Report Codepage Conversion Errors on page 171	<p>The number of seconds the driver waits for a connection to be established before returning control to the application and generating a timeout error.</p> <p>If set to 0 - Ignore Errors, the driver substitutes 0x1A for each character that cannot be converted and does not return a warning or error.</p> <p>If set to 1 - Return Error, the driver returns an error instead of substituting 0x1A for unconverted characters.</p> <p>If set to 2 - Return Warning, the driver substitutes 0x1A for each character that cannot be converted and returns a warning.</p> <p>Default: 0 - Ignore Errors</p>

Connection Options: Advanced	Description
Log Config File on page 160	Specifies the filename of the configuration file used to initialize the driver logging mechanism. If the driver cannot locate the specified file when establishing the connection, the connection fails and the driver returns an error. Default: None
Initialization String on page 156	One or multiple SQL commands to be executed by the driver after it has established the connection to the database and has performed all initialization for the connection. If the execution of a SQL command fails, the connection attempt also fails and the driver returns an error indicating which SQL command or commands failed. Default: None
Config Options on page 139	Determines how the mapping of the native data model to the relational data model is configured, customized, and updated. Default: None

Extended Options: Type a semi-colon separated list of connection options and their values. Use this configuration option to set the value of undocumented connection options that are provided by Progress DataDirect Technical Support. You can include any valid connection option in the Extended Options string, for example:

```
CreateMap=0;UndocumentedOption1=value [;UndocumentedOption2=value;]
```

If the Extended Options string contains option values that are also set in the setup dialog or data source, the values of the options specified in the Extended Options string take precedence. However, connection options that are specified on a connection string override any option value specified in the Extended Options string.

If you finished configuring your driver, proceed to Step 6 in "Configuring the product using the GUI." Optionally, you can further configure your driver by clicking on the following tabs. The following sections provide details on the fields specific to each configuration tab:

- [General tab](#) allows you to configure options that are required for creating a data source.
- [SQL Engine tab](#) allows you to configure the SQL engine's behavior.
- [Web Service tab](#) allows you to configure the behavior of communications between the driver and web service.
- [Pooling tab](#) allows you to configure connection pooling behavior.
- [Bulk tab](#) allows you to specify settings for DataDirect Bulk Load.

See also

[Config Options](#) on page 139

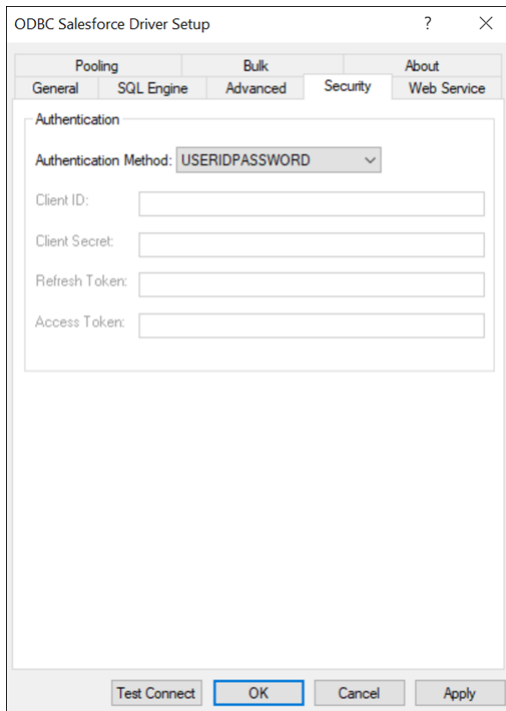
[Data source configuration through a GUI](#) on page 53

Security tab

The Security tab allows you to specify your security settings. The fields are optional unless otherwise noted. On this tab, provide values for the options in the following table; then, click **Apply**.

See "Using security" for a general description of authentication and encryption and their configuration requirements.

Figure 4: Security tab



Connection Options: Security	Description
Authentication Method on page 131	Determines which authentication method the driver uses when establishing a connection. If set to USERIDPASSWORD , the driver uses user ID/password authentication when establishing a connection. If set to OAuth2.0 , the driver uses OAuth 2.0 authentication when establishing a connection. Default: USERIDPASSWORD
Client ID on page 138	Specifies the consumer key for your application. The driver uses this value when authenticating to a Salesforce instance using OAuth 2.0 (<code>AuthenticationMethod=oauth2.0</code>).
Client Secret on page 139	Specifies the consumer secret for your application. This value can optionally be specified when authenticating to a Salesforce instance using OAuth 2.0 (<code>AuthenticationMethod=oauth2.0</code>).

Connection Options: Security	Description
Refresh Token on page 170	<p>Specifies the refresh token used to either request a new access token or renew an expired access token. When the refresh token is specified, the access token generated at connection is used to authenticate to a Salesforce instance when OAuth 2.0 is enabled (<code>AuthenticationMethod=oauth2.0</code>).</p> <hr/> <p>Note: If a value for the Access Token option is not specified, the driver uses the value of the Refresh Token option to make a connection. If both values are not specified, the driver cannot make a successful connection. If both are specified, the driver ignores the Access Token value and uses the Refresh Token value to generate a new Access Token value.</p> <hr/>
Access Token on page 129	<p>Specifies the access token required to authenticate to a Salesforce instance when OAuth 2.0 is enabled (<code>AuthenticationMethod=oauth2.0</code>).</p> <hr/> <p>Note: If a value for the Access Token option is not specified, the driver uses the value of the Refresh Token option to make a connection. If both values are not specified, the driver cannot make a successful connection. If both are specified, the driver ignores the Access Token value and uses the Refresh Token value to generate a new Access Token value.</p> <hr/>

If you finished configuring your driver, proceed to Step 6 in "Configuring the product using the GUI." Optionally, you can further configure your driver by clicking on the following tabs. The following sections provide details on the fields specific to each configuration tab:

- [General tab](#) allows you to configure options that are required for creating a data source.
- [SQL Engine tab](#) allows you to configure the SQL engine's behavior.
- [Advanced tab](#) allows you to configure the Advanced behavior.
- [Web Service tab](#) allows you to configure the behavior of communications between the driver and web service.
- [Pooling tab](#) allows you to configure connection pooling behavior.
- [Bulk tab](#) allows you to specify settings for DataDirect Bulk Load.

See also

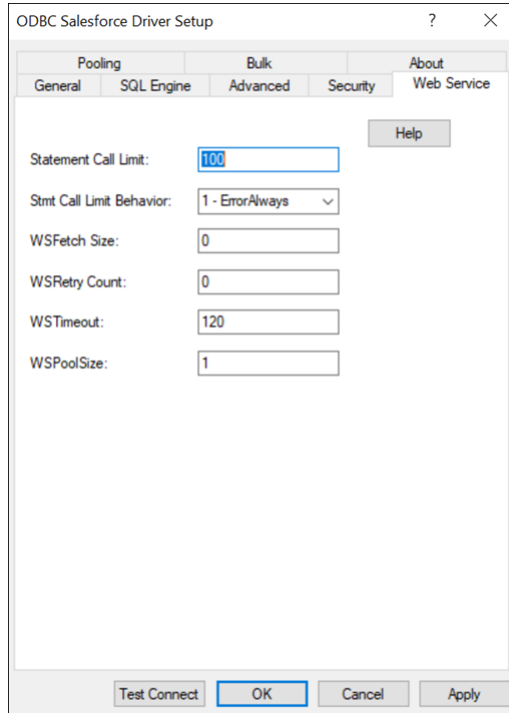
[Config Options](#) on page 139

[Data source configuration through a GUI](#) on page 53

Web Service tab

The Web Service tab allows you to specify additional datasource settings. The fields are optional unless otherwise noted. On this tab, provide values for the options in the following table; then, click **Apply**.

Figure 5: Web Service tab



Connection Options: Security	Description
Statement Call Limit on page 176	<p>Specifies the maximum number of Web service calls the driver can make when executing any single SQL statement or metadata query.</p> <p>If set to 0, there is no limit.</p> <p>If set to x, the driver uses this value to set the maximum number of Web service calls on a single connection that can be made when executing a SQL statement.</p> <p>Default: 100</p>
Statement Call Limit Behavior on page 176	<p>Specifies the behavior of the driver when the maximum Web service call limit specified by the Statement Call Limit option is exceeded.</p> <p>If set to 1 - ErrorAlways, the driver returns an error if the maximum Web service call limit is exceed.</p> <p>If set to 2 - ReturnResults, the driver returns any partial results it received prior to the call limit being exceeded. The driver generates a warning that not all of the results were fetched.</p> <p>Default: 1 - ErrorAlways</p>

Connection Options: Security	Description
WSFetch Size on page 178	<p>Specifies the number of rows of data the driver attempts to fetch for each ODBC call.</p> <p>If set to 0, the driver attempts to fetch up to a maximum of 2000 rows. This value typically provides the maximum throughput.</p> <p>If set to <i>x</i>, the driver attempts to fetch up to a maximum of the specified number of rows. Setting the value lower than 2000 can reduce the response time for returning the initial data. Consider using a smaller WSFetch Size for interactive applications only.</p> <p>Default: 0</p>
WSRetry Count on page 180	<p>The number of times the driver retries a timed-out Select request. Insert, Update, and Delete requests are never retried. The timeout period is specified by the WSTimeout (WST) connection option.</p> <p>If set to 0, the driver does not retry timed-out requests after the initial unsuccessful attempt.</p> <p>If set to <i>x</i>, the driver retries the timed-out request the specified number of times.</p> <p>Default: 0</p>
WSTimeout on page 181	<p>Specifies the time, in seconds, that the driver waits for a response to a Web service request.</p> <p>If set to 0, the driver waits indefinitely for a response; there is no timeout.</p> <p>If set to <i>x</i>, the driver uses the value as the default timeout for any statement created by the connection.</p> <p>Default: 120</p>
WSPoolSize on page 179	<p>Specifies the maximum number of sessions the driver uses when multiple connections to Salesforce are established. This allows the driver to have multiple web service requests active when multiple ODBC connections are open, thereby improving throughput and performance.</p> <hr/> <p>Note: The value specified should not exceed the number of sessions permitted by your Salesforce account.</p> <hr/> <p>Default: 1</p>

If you finished configuring your driver, proceed to Step 6 in "Configuring the product using the GUI". Optionally, you can further configure your driver by clicking on the following tabs. The following sections provide details on the fields specific to each configuration tab:

- [General tab](#) allows you to configure options that are required for creating a data source.
- [SQL Engine tab](#) allows you to configure the SQL engine's behavior.
- [Advanced tab](#) allows you to configure advanced behavior.
- [Pooling tab](#) allows you to configure connection pooling behavior.

- [Bulk tab](#) allows you to specify settings for DataDirect Bulk Load.

See also

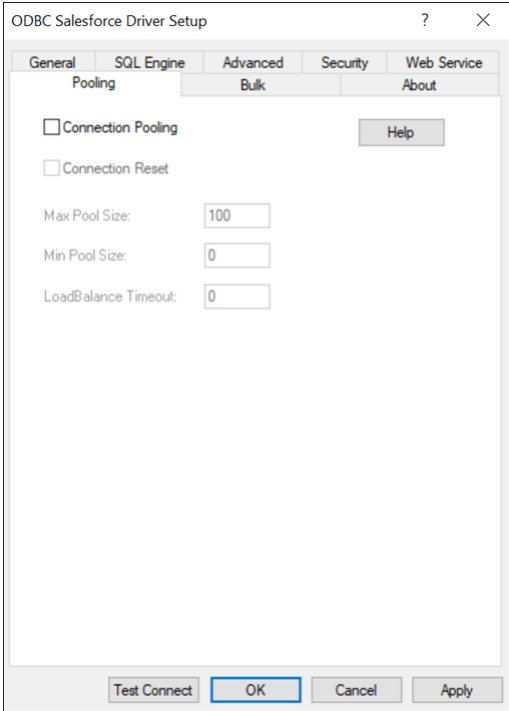
[Data source configuration through a GUI](#) on page 53

Pooling tab

The Pooling tab allows you to specify connection pooling data source settings. The fields are optional unless otherwise noted. On this tab, provide values for the options in the following table; then, click **Apply**.

See "Using DataDirect Connection Pooling" for a general description of connection pooling.

Figure 6: Pooling tab



Connection Options: Pooling	Description
Connection Pooling on page 145	Specifies whether to use the driver’s connection pooling. If enabled, the driver uses connection pooling. If disabled, the driver does not use connection pooling. Default: Disabled
Connection Reset on page 146	If enabled, the state of connections removed from the connection pool for reuse by an application is reset to the initial configuration of the connection. Resetting the state can negatively impact performance because additional commands must be sent over the network to the server to reset the state of the connection. If disabled, the state of connections is not reset. Default: Disabled

Connection Options: Pooling	Description
Max Pool Size on page 162	<p>The maximum number of connections allowed within a single connection pool. When the maximum number of connections is reached, no additional connections can be created in the connection pool.</p> <p>Default: 100</p>
Min Pool Size on page 163	<p>Specifies the minimum number of connections that are opened and placed in a connection pool, in addition to the active connection, when the pool is created. The connection pool retains this number of connections, even when some connections exceed their Load Balance Timeout value.</p> <p>If set to 0, no connections are opened in addition to the current existing connection.</p> <p>Default: 0</p>
LoadBalance Timeout on page 159	<p>Specifies the number of seconds to keep inactive connections open in a connection pool.</p> <p>If set to 0, inactive connections are kept open.</p> <p>If set to x, inactive connections are closed after the specified number of seconds passes.</p> <p>Default: 0</p>

If you finished configuring your driver, proceed to Step 6 in "Configuring the product using the GUI". Optionally, you can further configure your driver by clicking on the following tabs. The following sections provide details on the fields specific to each configuration tab:

- [General tab](#) allows you to configure options that are required for creating a data source.
- [SQL Engine tab](#) allows you to configure the SQL engine's behavior.
- [Advanced tab](#) allows you to configure advanced behavior.
- [Web Service tab](#) allows you to configure the behavior of communications between the driver and web service.
- [Bulk tab](#) allows you to specify settings for DataDirect Bulk Load.

See also

[Using DataDirect Connection Pooling](#) on page 102

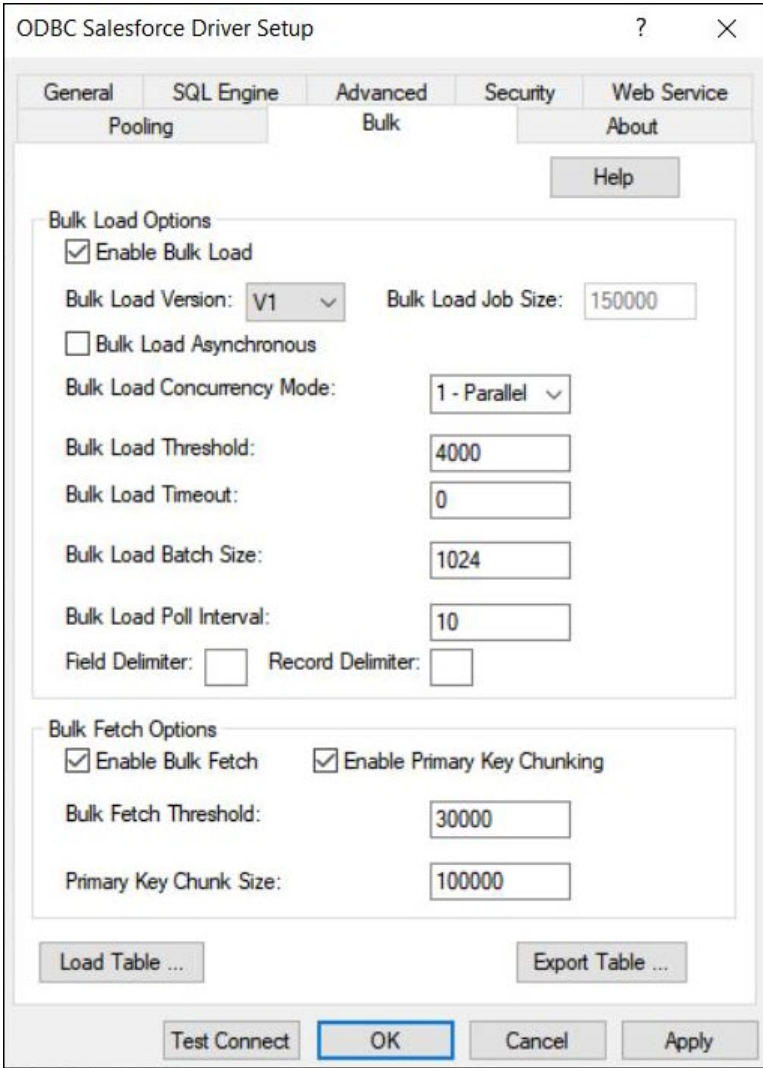
[Data source configuration through a GUI](#) on page 53

Bulk tab

The Bulk tab allows you to specify DataDirect Bulk Load data source settings. The fields are optional unless otherwise noted. On this tab, provide values for the options in the following table; then, click **Apply**.

See "Using DataDirect Bulk Load" for a general description of DataDirect Bulk Load.

Figure 7: Bulk tab



Connection Options: Bulk	Description
<p>Enable Bulk Load on page 151</p>	<p>Specifies whether the driver can use the Salesforce Bulk API for inserts, updates, and deletes based on the value of the Bulk Load Threshold connection option. If the number of affected rows exceeds the value of Bulk Load Threshold option, the driver uses the Salesforce Bulk API to execute the insert, update, or delete operation. Using the Salesforce Bulk API may significantly reduce the number of Web service calls used to execute a statement and, therefore, may improve performance.</p> <p>If enabled, the driver can use the Salesforce Bulk API for inserts, updates, and deletes based on the value of the Bulk Load Threshold connection option. If the number of affected rows exceeds the value of Bulk Load Threshold option, the driver uses the Salesforce Bulk API to execute the insert, update, or delete operation.</p> <p>If disabled, the driver does not use the Salesforce Bulk API, and the Bulk Load Threshold option is ignored.</p> <p>Default: Enabled</p>

Connection Options: Bulk	Description
Bulk Load Version on page 137	<p>Specifies which version of Salesforce Bulk Load API to be used for performing bulk load operations. This is applicable only if EnableBulkLoad is set to 1 (Enabled).</p> <p>If set to v1, the driver uses the Salesforce Bulk API V1 for all insert, update, and delete operations.</p> <p>If set to v2, the driver uses the Salesforce Bulk API V2 to execute the insert, update, and delete operations.</p> <p>Default: v1</p>
Bulk Load Job Size on page 135	<p>Determines the number of rows to load into a single job of a bulk operation when BulkLoadVersion is set to V2. Performance can be improved by increasing the number of rows the driver loads at a time because fewer network round trips are required. Increasing the number of rows also causes the driver to consume more memory on the client.</p> <p>Default: 150000</p>
Bulk Load Asynchronous on page 133	<p>Determines whether the driver treats bulk load operations as synchronous or asynchronous.</p> <p>If disabled, bulk load operations are synchronous. The driver does not return from the function that invoked an operation until the operation is complete or the BulkLoadTimeout period has expired. If the operation times out, the driver returns an error.</p> <p>If enabled, bulk load operations are asynchronous. The driver returns from the function that invoked an operation after the operation is submitted to the server. The driver does not verify the completion status of the bulk load operation.</p> <p>Default: Disabled</p>
Bulk Load Concurrency Mode on page 134	<p>Determines whether multiple batches associated with a bulk load operation are processed by Salesforce in parallel or one at a time.</p> <p>If set to 0 - Serial, multiple batches associated with a bulk load operation are processed one at a time.</p> <p>If set to 1 - Parallel, multiple batches associated with a bulk load operation are processed in parallel. The order in which the batches are processed can vary.</p> <p>Default: 1 - Parallel</p>
Bulk Load Threshold on page 136	<p>Determines when the driver uses bulk load for insert, update, delete, or batch operations.</p> <p>If set to 0, the driver always uses bulk load to execute insert, update, delete, or batch operations.</p> <p>If set to x, the driver only uses bulk load if the Enable Bulk Load option is enabled and the number of rows to be updated by an insert, update, delete, or batch operation exceeds the threshold. If the operation times out, the driver returns an error.</p> <p>Default: 4000</p>

Connection Options: Bulk	Description
Bulk Load Timeout on page 137	<p>The time, in seconds, that the driver waits for a Salesforce bulk job to complete. A value of zero means there is no timeout.</p> <p>Default: 0</p>
Bulk Load Batch Size on page 133	<p>The number of rows that the driver sends to the database at a time during bulk operations.</p> <p>Default: 1024</p>
Bulk Load Poll Interval on page 135	<p>Specifies the number of seconds the driver waits to request bulk operation status. This interval is used by the driver the first time it requests status and for all subsequent status requests.</p> <p>Default: 10</p>
Field Delimiter on page 154	<p>Specifies the character that the driver will use to delimit the field entries in a bulk load data file.</p> <p>Default: None</p>
Record Delimiter on page 168	<p>Specifies the character that the driver will use to delimit the record entries in a bulk load data file.</p> <p>Default: None</p>
Enable Bulk Fetch on page 150	<p>Specifies whether the driver can use the Salesforce Bulk API for selects based on the value of the Bulk Fetch Threshold connection option. If the number of rows expected in the result set exceeds the value of Bulk Fetch Threshold option, the driver uses the Salesforce Bulk API to execute the select operation. Using the Salesforce Bulk API may significantly reduce the number of Web service calls used to execute a statement and, therefore, may improve performance.</p> <p>If enabled, the driver can use the Salesforce Bulk API for selects based on the value of the Bulk Fetch Threshold connection option. If the number of rows expected in the result set exceeds the value of Bulk Fetch Threshold option, the driver uses the Salesforce Bulk API to execute the select operation.</p> <p>If disabled, the driver does not use the Salesforce Bulk API, and the Bulk Fetch Threshold option is ignored.</p> <p>Default: Enabled</p> <hr/> <p>Note:</p> <p>If there is a TOP or LIMIT clause in the select query, the driver considers the TOP or LIMIT clause value as the expected number of rows in the result set and compares it with the value of the Bulk Fetch Threshold option to choose either Bulk API or REST API for executing the select query.</p> <p>If the value of TOP or LIMIT clause is greater than that of the Bulk Fetch Threshold option, but the actual row count in the result set is less, the performance of the select query might be affected adversely.</p> <hr/>

Connection Options: Bulk	Description
Enable Primary Key Chunking on page 152	<p>Specifies whether the driver uses PK chunking for select operations. PK chunking breaks down bulk fetch operations into smaller, more manageable batches for improved performance.</p> <p>If enabled, the driver uses PK chunking for select operations when the expected number of rows in the result set is greater than the values of the Bulk Fetch Threshold and Primary Key Chunk Size options. For this behavior to take effect, the Enable Bulk Fetch option must also be enabled.</p> <p>If disabled, the driver does not use PK chunking when executing select operations, and the Primary Key Chunk Size option is ignored.</p> <p>Default: Enabled</p>
Bulk Fetch Threshold on page 132	<p>Specifies a number of rows that, if exceeded, signals the driver to use the Salesforce Bulk API for select operations. For this behavior to take effect, the Enable Bulk Fetch option must be enabled.</p> <p>Default: 30000 (rows)</p>
Primary Key Chunk Size on page 164	<p>Specifies the size, in rows, of a primary key (PK) chunk when PK chunking has been enabled via the Enable Primary Key Chunking option. The Salesforce Bulk API splits the query into chunks of this size.</p> <p>Default: 100000 (rows)</p>

If your application is already coded to use parameter array batch functionality, you can leverage DataDirect Bulk Load features through the Enable Bulk Load connection option. Enabling this option automatically converts the parameter array batch operation to use the database bulk load protocol.

If you are not using parameter array batch functionality, you can export data to a bulk load data file, verify the metadata of the bulk load configuration file against the structure of the target table, and bulk load data to a table. Use the following steps to accomplish these tasks.

1. To export data from a table to a bulk load data file, click **Export Table** from the Bulk tab. The Export Table dialog box appears.

Figure 8: ODBC Salesforce Export Table Driver Setup dialog box

Both a bulk data file and a bulk configuration file are produced by exporting a table. The configuration file has the same name as the data file, but with an XML extension. See "Using DataDirect Bulk Load" for details about these files.

The bulk export operation can create a log file and can also export to external files. See "External overflow files" for more information. The export operation can be configured such that if any errors or warnings occur:

- The operation always completes.
- The operation always terminates.
- The operation terminates after a certain threshold of warnings or errors is exceeded.

Table Name: A string that specifies the name of the source database table and, optionally, the columns containing the data to be exported. The driver uses the table name in the FROM clause of a SELECT * FROM tablename SQL statement. If you want to only export certain columns from your Salesforce table, then you can enter a SELECT statement in this field using the format:

```
(SELECT column1, column2, ... FROM tablename)
```

For example, to export data from the Salesforce ACCOUNT table excluding some of the audit columns, enter the following SQL in the Table Name field:

```
(SELECT SYS_NAME, TYPE, BILLINGSTREET, BILLINGCITY, BILLINGSTATE, BILLINGPOSTALCODE,
BILLINGCOUNTRY,
SHIPPINGSTREET, SHIPPINGCITY, SHIPPINGSTATE, SHIPPINGPOSTALCODE, SHIPPINGCOUNTRY, PHONE,
FAX, WEBSITE,
INDUSTRY, ANNUALREVENUE, NUMBEROFEMPLOYEES, DESCRIPTION FROM ACCOUNT)
```

Export Filename: A string that specifies the path (relative or absolute) and file of the bulk load data file to which the data is to be exported. It also specifies the file name of the bulk configuration file. The file name must be the fully qualified path to the bulk data file. These files must not already exist; if one of both of them already exists, an error is returned.

Log Filename: A string that specifies the path (relative or absolute) and file name of the bulk log file. The log file is created if it does not exist. The file name must be the fully qualified path to the log file. Events logged to this file are:

- Total number of rows fetched
- A message for each row that failed to export
- Total number of rows that failed to export
- Total number of rows successfully exported

Information about the load is written to this file, preceded by a header. Information about the next load is appended to the end of the file.

If you do not supply a value for Log Filename, no log file is created.

Error Tolerance: A value that specifies the number of errors to tolerate before an operation terminates. A value of 0 indicates that no errors are tolerated; the operation fails when the first error is encountered.

The default of -1 means that an infinite number of errors is tolerated.

Warning Tolerance: A value that specifies the number of warnings to tolerate before an operation terminates. A value of 0 indicates that no warnings are tolerated; the operation fails when the first warning is encountered.

The default of -1 means that an infinite number of warnings is tolerated.

Code Page: A value that specifies the code page value to which the driver must convert all data for storage in the bulk data file. See "Character set conversions" for more information.

The default value on Windows is the current code page of the machine.

Click **Export Table** to connect to the database and export data to the bulk data file or click **Cancel**.

- To bulk load data from the bulk data file to a database table, click **Load Table** from the Bulk tab. The Load File dialog box appears.

Figure 9: ODBC Salesforce Load File Driver Setup dialog box

The load operation can create a log file and can also create a discard file that contains rows rejected during the load. The discard file is in the same format as the bulk load data file. After fixing reported issues in the discard file, the bulk load can be reissued using the discard file as the bulk load data file.

The export operation can be configured such that if any errors or warnings occur:

- The operation always completes.
- The operation always terminates.
- The operation terminates after a certain threshold of warnings or errors is exceeded.

If a load fails, the Load Start and Load Count options can be used to control which rows are loaded when a load is restarted after a failure.

Table Name: A string that specifies the name of the target database table and, optionally, the columns into which the data is loaded.

The fields defined in the load data file must have the same ordering of the fields defined in the Salesforce destination table. Because Salesforce defines additional audit columns that are managed by the database, your load data file may not contain data to load into these fields.

In this case, you can specify the exact columns that you want for the data to be inserted into using a Table Name string of the format:

```
table(column1, column2, ...)
```

For example, if your load data file contains only five fields of billing data that you wanted to load into the Salesforce ACCOUNT table, then the Table Name field would contain:

```
ACCOUNT(BILLINGSTREET, BILLINGCITY, BILLINGSTATE, BILLINGPOSTALCODE, BILLINGCOUNTRY)
```

Load Data Filename: A string that specifies the path (relative or absolute) and file name of the bulk data file from which the data is loaded.

Configuration Filename: A string that specifies the path (relative or absolute) and file name of the bulk configuration file.

Log Filename: A string that specifies the path (relative or absolute) and file name of the bulk log file. The file name must be the fully qualified path to the log file. Specifying a value for Log Filename creates the file if it does not already exist. Events logged to this file are:

- Total number of rows read
- Message for each row that failed to load
- Total number of rows that failed to load
- Total number of rows successfully loaded

Information about the load is written to this file, preceded by a header. Information about the next load is appended to the end of the file.

If you do not specify a value for Log Filename, no log file is created.

Discard Filename: A string that specifies the path (relative or absolute) and file name of the bulk discard file. Any row that cannot be inserted into the database as result of bulk load is added to this file, with the last row rejected added to the end of the file.

Information about the load is written to this file, preceded by a header. Information about the next load is appended to the end of the file.

If you do not specify a value for Discard Filename, a discard file is not created.

Error Tolerance: A value that specifies the number of errors to tolerate before an operation terminates. A value of 0 indicates that no errors are tolerated; the operation fails when the first error is encountered.

The default of -1 means that an infinite number of errors is tolerated.

Load Start: A value that specifies the first row to be loaded from the data file. Rows are numbered starting with 1. For example, when Load Start is 10, the first 9 rows of the file are skipped and the first row loaded is row 10. This option can be used to restart a load after a failure.

The default value is 1.

Read Buffer Size (KB): A value that specifies the size, in KB, of the buffer that is used to read the bulk data file for a bulk load operation.

The default value is 2048.

Warning Tolerance: A value that specifies the number of warnings to tolerate before an operation terminates. A value of 0 indicates that no warnings are tolerated; the operation fails when the first warning is encountered.

The default of -1 means that an infinite number of warnings is tolerated.

Load Count: A value that specifies the number of rows to be loaded from the data file. The bulk load operation loads rows up to the value of Load Count from the file to the database. It is valid for Load Count to specify more rows than exist in the data file. The bulk load operation completes successfully when either the number of rows specified by the Load Count value has been loaded or the end of the data file is reached. This option can be used in conjunction with Load Start to restart a load after a failure.

The default value is the maximum value for SQLULEN. If set to 0, no rows are loaded.

Click **Load Table** to connect to the database and load the table or click **Cancel**.

If you finished configuring your driver, proceed to Step 6 on page 56 in [Data source configuration through a GUI](#) on page 53. Optionally, you can further configure your driver by clicking on the following tabs. The following sections provide details on the fields specific to each configuration tab:

- [General tab](#) allows you to configure options that are required for creating a data source.
- [SQL Engine tab](#) allows you to configure the SQL engine's behavior.
- [Advanced tab](#) allows you to configure advanced behavior.
- [Web Service tab](#) allows you to configure the behavior of communications between the driver and web service.
- [Pooling tab](#) allows you to configure connection pooling behavior.

See also

[DataDirect Bulk Load](#) on page 109

[External overflow files](#) on page 116

[Character set conversions](#) on page 115

[Data source configuration through a GUI](#) on page 53

Using a connection string

If you want to use a connection string for connecting to a database, or if your application requires it, you must specify either a DSN (data source name), a File DSN, or a DSN-less connection in the string. The difference is whether you use the `DSN=`, `FILEDSN=`, or the `DRIVER=` keyword in the connection string, as described in the ODBC specification. A DSN or FILEDSN connection string tells the driver where to find the default connection information. Optionally, you may specify *attribute=value* pairs in the connection string to override the default values stored in the data source.

The DSN connection string has the form:

```
DSN=data_source_name[;attribute=value[;attribute=value]. . .]
```

The FILEDSN connection string has the form:

```
FILEDSN=filename.dsn[;attribute=value[;attribute=value]. . .]
```

The DSN-less connection string specifies a driver instead of a data source. All connection information must be entered in the connection string because the information is not stored in a data source.

The DSN-less connection string has the form:

```
DRIVER=[{driver_name}] [;attribute=value[;attribute=value]. . .]
```

"Connection option descriptions" lists the long and short names for each attribute, as well as the initial default value when the driver is first installed. You can specify either long or short names in the connection string.

An example of a DSN connection string with overriding attribute values for Salesforce for Linux/UNIX/Windows is:

```
DSN=Salesforce;UID=JOHN;PWD=XYZZY
```

A FILEDSN connection string is similar except for the initial keyword:

```
FILEDSN=Salesforce;UID=JOHN;PWD=XYZZY
```

A DSN-less connection string must provide all necessary connection information:

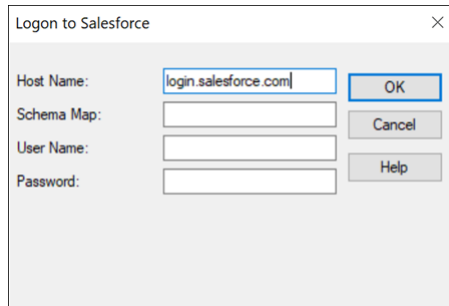
```
DRIVER=DataDirect 8.0 Salesforce;UID=JohnQPublic;PWD=XYZZY;HOST=login.salesforce.com;  
SM=C:\Users\Default\AppData\Local\Progress\DataDirect\Salesforce Schema\JohnQPublic.config;  
SecurityToken=XaBARTsLzReM4Px47qPLOS
```

See also

[Connection option descriptions](#) on page 123

Using a logon dialog box

Some ODBC applications display a logon dialog box when you are connecting to a data source. In these cases, the host name has already been specified.

Figure 10: Logon to Salesforce dialog box

In this dialog box, provide the following information:

1. In the Host Name field, type the URL or the IP address of the Salesforce instance to which you want to connect.
2. Optionally, in the Schema Map field, type the name and location of the configuration file where the relational map of native data is written.
3. In the User Name field, type the default user ID and domain that is used to connect to your Salesforce instance. For example, `jsmith@defcorp.com`
4. In the Password field, type the password that is used to connect to your Salesforce instance.
5. Click **OK** to complete the logon.

Connecting through a proxy server

In some environments, your application may need to connect through a proxy server, for example, if your application accesses an external resource such as a Web service. At a minimum, your application needs to provide the following connection information when you invoke the JVM if the application connects through a proxy server:

- Server name or IP address of the proxy server
- Port number on which the proxy server is listening for HTTP/HTTPS requests

In addition, if authentication is required, your application may need to provide a valid user ID and password for the proxy server. Consult with your system administrator for the required information.

For example, the following command invokes the JVM while specifying a proxy server named `pserver`, a port of 808, and provides a user ID and password for authentication:

```
java -Dhttp.proxyHost=pserver -Dhttp.proxyPort=808 -Dhttp.proxyUser=smith
-Dhttp.proxyPassword=secret -cp sforce.jar com.acme.myapp.Main
```

Alternatively, you can use the Proxy Host, Proxy Port, Proxy User, and Proxy Password connection attributes. See "Connection option descriptions" for details about these attributes.

See also

[Connection option descriptions](#) on page 123

Performance considerations

Application Using Threads (ApplicationUsingThreads): The driver coordinates concurrent database operations (operations from different threads) by acquiring locks. Although locking prevents errors in the driver, it also decreases performance. If your application does not make ODBC calls from different threads, the driver has no reason to coordinate operations. In this case, the ApplicationUsingThreads attribute should be disabled (set to 0).

Note: If you are using a multi-threaded application, you must enable the Application Using Threads option.

Bulk Load Job Size (BulkLoadJobSize): This option can be used to improve performance by increasing the number of rows the driver loads at a time because fewer network round trips are required. However, a higher number of rows also causes the driver to consume more memory on the client.

Connection Pooling (Pooling): If you enable the driver to use connection pooling, you can set additional options that affect performance:

- **Load Balance Timeout (LoadBalanceTimeout):** You can define how long to keep connections in the pool. The time that a connection was last used is compared to the current time and, if the timespan exceeds the value of the Load Balance Timeout option, the connection is destroyed. The Min Pool Size option can cause some connections to ignore this value.
- **Connection Reset (ConnectionReset):** Resetting a re-used connection to the initial configuration settings impacts performance negatively because the connection must issue additional commands to the server.
- **Max Pool Size (MaxPoolSize):** Setting the maximum number of connections that the pool can contain too low might cause delays while waiting for a connection to become available. Setting the number too high wastes resources.
- **Min Pool Size (MinPoolSize):** A connection pool is created when the first connection with a unique connection string connects to the database. The pool is populated with connections up to the minimum pool size, if one has been specified. The connection pool retains this number of connections, even when some connections exceed their Load Balance Timeout value.

Enable Bulk Fetch (EnableBulkFetch): The Enable Bulk Fetch option can be used to improve performance by enabling the driver to use the Salesforce Bulk API for selects. Using the Salesforce Bulk API may significantly reduce the number of Web service calls used to execute a statement and, therefore, may improve performance. When Enable Bulk Fetch is set to 1 (enabled), the driver uses the Salesforce Bulk API based on the value of the Bulk Fetch Threshold connection option. If the number of rows expected in the result set exceeds the value of Bulk Fetch Threshold option, the driver uses the Salesforce Bulk API to execute the select operation.

Note:

If there is a TOP or LIMIT clause in the select query, the driver considers the TOP or LIMIT clause value as the expected number of rows in the result set and compares it with the value of the Bulk Fetch Threshold option to choose either Bulk API or REST API for executing the select query.

If the value of TOP or LIMIT clause is greater than that of the Bulk Fetch Threshold option, but the actual row count in the result set is less, the performance of the select query might be affected adversely.

Enable Bulk Load (EnableBulkLoad): The Enable Bulk Load option can be used to improve performance by enabling the driver to use the Salesforce Bulk API for inserts, updates, and deletes. Using the Salesforce Bulk API may significantly reduce the number of Web service calls used to execute a statement and, therefore, may improve performance. When Enable Bulk Load is set to 1 (enabled), the driver uses the Salesforce Bulk API based on the value of the Bulk Load Threshold connection option. If the number of affected rows exceeds the value of Bulk Load Threshold option, the driver uses the Salesforce Bulk API to execute the insert, update, or delete operation.

Enable Primary Key Chunking (EnablePKChunking): The Enable Primary Key Chunking option can be used to improve performance by enabling the driver to use PK chunking for bulk fetch operations. When Enable Primary Key Chunking is set to 1(enabled), the driver uses PK chunking to execute the select operations if the expected number of rows in the result set is greater than the values of the Bulk Fetch Threshold and Primary Key Chunk Size options. For this behavior to take effect, the Enable Bulk Fetch option must also be set to 1 (enabled).

Note: PK chunking is supported for all custom objects and the following standard objects: Account, Campaign, CampaignMember, Case, Contact, Lead, LoginHistory, Opportunity, Task, and User. In addition, PK chunking is supported for sharing objects as long as the parent object is supported.

Fetch Size/Web Service Fetch Size (FetchSize/WSFetchSize): The connection options Fetch Size and Web Service Fetch Size can be used to adjust the trade-off between throughput and response time. In general, setting larger values for Web Service Fetch Size and Fetch Size will improve throughput, but can reduce response time.

For example, if an application attempts to fetch 100,000 rows from the remote data source and Web Service Fetch Size is set to 500, the driver must make 200 Web service calls to get the 100,000 rows. If, however, Web Service Fetch Size is set to 4000, the driver only needs to make 25 Web service calls to retrieve 100,000 rows. Web service calls are expensive, so generally, minimizing Web service calls increases throughput. In addition, many Cloud data sources impose limits on the number of Web service calls that can be made in a given period of time. Minimizing the number of Web service calls used to fetch data also can help prevent exceeding the data source call limits.

For many applications, throughput is the primary performance measure, but for interactive applications, such as Web applications, response time (how fast the first set of data is returned) is more important than throughput. For example, suppose that you have a Web application that displays data 50 rows to a page and that, on average, you view three or four pages. Response time can be improved by setting Fetch Size to 50 (the number of rows displayed on a page) and WSFetch Size to 200. With these settings, the driver fetches all of the rows from the remote data source that you would typically view in a single Web service call and only processes the rows needed to display the first page.

WSPoolSize (WSPoolSize)PK chunking is supported for all custom objects and the: WSPoolSize determines the maximum number of sessions the driver uses when there are multiple active connections to Salesforce. By increasing this number, you increase the number of sessions the driver uses to distribute calls to Salesforce, thereby improving throughput and performance. For example, if WSPoolSize is set to 1, and you have two open connections, the session must complete a call from one connection before it can begin processing a call from the other connection. However, if WSPoolSize is equal to 2, a second session is opened that allows calls from both connections to be processed simultaneously.

Note: The number specified for WSPoolSize should not exceed the amount of sessions permitted by your Salesforce account.

Using the SQL engine server

Some applications may experience problems loading the JVM required for the SQL engine because the process exceeds the available heap space. If your application experiences problems loading the JVM, you can configure the driver to operate in server mode.

In direct mode, the driver operates with the SQL engine and JVM running in a single process. While in server mode, the driver's SQL engine runs in a separate process with its own JVM instead of trying to load the SQL engine and JVM in the same process used by the driver.

For Windows, the driver is configured to attempt to run in server mode first by default. However, if server mode is unavailable, the SQL engine will failover to run in direct mode. For non-Windows platforms, the driver operates in direct mode by default.

Note: You must be an administrator to start or stop the service, or to configure any settings for the service.

See the following sections for details on configuring the SQL engine server on your platform.

Configuring the SQL engine server on Windows

The following sections describe how to configure, start, and stop the SQL engine server on Windows platforms.

On Windows, the driver is configured to run in Auto mode by default. This means that driver attempts to run in server mode first; however, if server mode is unavailable, the SQL engine will failover to run in direct mode.

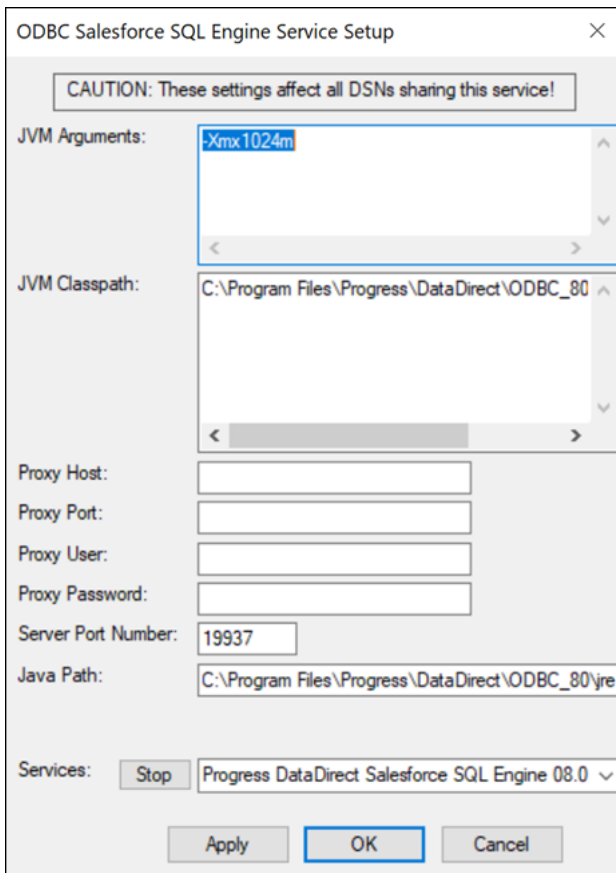
Configuring server mode

1. Set the SQL Engine Mode connection option to a value of **0 - Auto** or **1 - Server**. All fields on the SQL Engine tab become read only, and the **Edit Server Settings** button appears.

Note: Server mode is enabled when the SQL Engine Mode connection option is set to **0 - Auto** or **1 - Server**. When set **0 - Auto**, the SQL engine attempts to run in server mode first, but will failover to direct mode if server mode is unavailable. When set to **1 - Server**, the SQL engine mode runs exclusively in server mode.

2. Click **Edit Server Setting** to display the ODBC Salesforce SQL Engine Service Setup dialog box. Use this dialog box to define settings for Server Mode and to start and stop the SQL engine service.

The SQL Engine Service Setup dialog box appears.



JVM Arguments: A string that contains the arguments that are passed to the JVM that the driver is starting. The location of the JVM must be specified on your PATH. See "JVM Arguments."

JVM Class Path: Specifies the CLASSPATH for the JVM used by the driver. See "JVM Classpath."

Server Port Number: Specifies a valid port on which the SQL engine listens for requests from the driver. By default, the server listens on port 19937 for 64-bit installations and 19938 for 32-bit installations.. See "Server Port Number" for more information.

Java Path: Specifies fully qualified path to the Java SE 8 or higher JVM executable that you want to use to run the SQL engine server. The path must not contain double quotation marks.

Services: Shows the Salesforce ODBC SQL engine service that runs as a separate process instead of being loaded within the process of an ODBC application.

Start (Stop): Starts or stops the Salesforce service. A message window is displayed, confirming that the Salesforce service was started or stopped.

Apply: Applies the changes.

Note: After the initial configuration, in order for changes to these connection option values to take effect, you must restart the SQL engine server.

3. When you complete your changes, click **Apply**.
4. Click **OK** to save the changes and return to the SQL Engine tab or click **Cancel**.

See also

[JVM Arguments](#) on page 157

[JVM Classpath](#) on page 158

[Server Port Number](#) on page 174

Starting the SQL engine server

In server mode, you must start the SQL engine server before using the driver. Before starting the SQL engine server, choose a directory to store the local database files. Make sure that you have the correct permissions to write to this directory.

By default, the JVM Classpath is set to the `sforce.jar` file in the installation directory.

To start the SQL engine server:

1. Start the ODBC Administrator by selecting its icon from the Progress DataDirect for ODBC program group.
2. Select a tab:
 - **User DSN:** If you are configuring an existing user data source, select the data source name and click **Configure** to display the driver Setup dialog box.

If you are configuring a new user data source, click **Add** to display a list of installed drivers. Select the driver and click **Finish** to display the driver Setup dialog box.

- **System DSN:** If you are configuring an existing system data source, select the data source name and click **Configure** to display the driver Setup dialog box.

If you are configuring a new system data source, click **Add** to display a list of installed drivers. Select the driver and click **Finish** to display the driver Setup dialog box.

- **File DSN:** If you are configuring an existing file data source, select the data source file and click **Configure** to display the driver Setup dialog box.

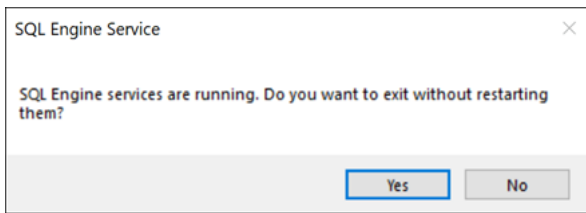
If you are configuring a new file data source, click **Add** to display a list of installed drivers; then, select a driver. Click **Advanced** if you want to specify attributes; otherwise, click **Next** to proceed. Specify a name for the data source and click **Next**. Verify the data source information; then, click **Finish** to display the driver Setup dialog box.

3. On the ODBC Salesforce Driver Setup dialog box, select the **SQL Engine** tab; then, select **0 - Auto** or **1 - Server** from the SQL Engine Mode drop-down list.

Note: Server mode is enabled when the SQL Engine Mode connection option is set to **0 - Auto** or **1 - Server**. When set **0 - Auto**, the SQL engine attempts to run in server mode first, but will failover to direct mode if server mode is unavailable. When set to **1 - Server**, the SQL engine mode runs exclusively in server mode.

4. Click **Edit Server Settings**.
5. When you complete your changes, click **Apply**.
6. Verify that Progress DataDirect Salesforce SQL Engine is selected in the Services drop-down list, and then, click **Start** to start the service. A message window appears to confirm that the service is running. Click **OK**.
7. Click **OK** to close the ODBC Salesforce SQL Engine Service Setup dialog box.

Note: If you made changes after starting the service, a message window is displayed:



If you want the service to run with the new settings, click **No**. Then, click **Stop** to stop the service, and then click **Start** to restart the service. Then, click **OK** to close the ODBC Salesforce SQL Engine Service Setup dialog box.

Stopping the SQL engine server

To stop the SQL engine server:

1. Open the ODBC Salesforce Driver Setup dialog box and select the SQL Engine tab.
2. Select **0 - Auto** or **1 - Server** from the SQL Engine Mode drop-down list. Then, click **Edit Server Settings**.

Note: Server mode is enabled when the SQL Engine Mode connection option is set to **0 - Auto** or **1 - Server**. When set **0 - Auto**, the SQL engine attempts to run in server mode first, but will failover to direct mode if server mode is unavailable. When set to **1 - Server**, the SQL engine mode runs exclusively in server mode.

3. Click **Stop** to stop the service. A message window appears to confirm that the service is stopped. Click **OK**.
4. Click **OK** to close the ODBC Salesforce SQL Engine Service Setup dialog box.

Configuring the SQL engine server on UNIX/Linux

The following sections describe how to configure, start, and stop the SQL engine server on UNIX and Linux platforms.

By default, the driver operates in direct mode by default on UNIX and Linux platforms.

Configuring and starting the SQL engine server on UNIX/Linux

In server mode, you must start the SQL engine server before using the driver. Before starting the SQL engine server, verify that you have the correct permissions to write to the directory specified by the SchemaMap option.

To configure the SQL engine server, specify values for the Java options in the following JVM argument to suit your environment. See the "SQL engine server Java options" table for a description of these options.

```
java -Xmx<heap_size>m -cp "<jvm_classpath>" com.ddtek.phoenix.sql.Server
-port <port_number> -Dhttp.proxyHost=<proxy_host> -Dhttp.proxyPort=<proxy_port>
-Dhttp.proxyUser=<proxy_user> -Dhttp.proxyPassword=<proxy_password>
```

For example:

```
java -Xmx1024m -cp "/opt/Progress/DataDirect/ODBC_80_64bit/java/lib/sforce.jar"
com.ddtek.phoenix.sql.Server -port 19938 -Dhttp.proxyHost=myhost@mydomain.com
-Dhttp.proxyPort=12345 -Dhttp.proxyUser=JohnQPublic -Dhttp.proxyPassword=secret
```

To start the SQL engine service, execute the JVM Argument after configuring the Java options. A confirmation message is returned once the server is online.

Note: After the initial configuration, in order for changes to these connection option values to take effect, you must restart the SQL engine server.

Table 6: SQL engine server Java options

Java Option	Description
Required Java Options	
-cp	Specifies the CLASSPATH for the Java Virtual Machine (JVM) used by the driver. The CLASSPATH is the search string the JVM uses to locate the Java jar files the driver needs. The Salesforce driver's JVM is located on the following path: <i>install_dir/java/lib/sforce.jar</i>
-port	Specifies a valid port on which the SQL engine listens for requests from the driver. We recommend specifying one of the following values: <ul style="list-style-type: none"> • 19938 (32-bit drivers) • 19937 (64-bit drivers)
Optional Java Options	
-Xmx	Specifies the maximum memory heap size, in megabytes, for the JVM. The default size is determined by your JVM. We recommend specifying a size no smaller than 1024. Note: Although this option is not required to start the SQL engine server, we highly recommend specifying a value.
-Dhttp.proxyHost	Specifies the Hostname of the Proxy Server. The value specified can be a host name, a fully qualified domain name, or an IPv4 or IPv6 address.
-Dhttp.proxyPort	Specifies the port number where the Proxy Server is listening for HTTP and/or HTTPS requests.
-Dhttp.proxyUser	Specifies the user name needed to connect to the Proxy Server.
-Dhttp.proxyPassword	Specifies the password needed to connect to the Proxy Server.

Stopping the SQL engine server

To stop the SQL engine server, choose one of the following:

- Using an application, execute SHUTDOWN SQL.
- From a command line, press `Ctrl + C`.

A message is returned to confirm that the service is stopped.

Configuring Java logging for the SQL engine server

Java logging can be configured by placing a logging configuration file named `ddlog.properties` in the directory specified by the Schema Map option. The simple way to create one of these is to make a copy of the `ddlog.properties` file, which is located in your driver installation directory, in the `install_dir/Sample/Example` subdirectory.

For more information on logging, refer to "Loggers and logging levels" in the *Progress DataDirect for ODBC Drivers Reference*.

See also

[Schema Map](#) on page 172

Client-side caches

The Salesforce driver can implement a client-side data cache for improved performance. Data is cached from the remote data source to the local machine on which the driver is located.

The driver caches data on a per-table basis, as opposed to caching the result of a particular query. Caching data on a table level allows the caches to be queried, filtered, and sorted in other queries. Once a cache is created, its use is transparent to the application. For example, if a cache is created on the Account table, then all subsequent queries that reference Account access the Account cache. Disabling or dropping the cache allows references to the Account table to access the remote data again. Because the use of the cache is transparent, no changes to the application are required to take advantage of the cache.

You must specifically create a cache before it can be populated; caches are not created automatically. After you have created a cache on a table, the cache will be populated as a result of the next operation on the table. For example, after creating a cache on Account, data is returned from the Salesforce data source and stored locally in the cache when you first execute the following statement:

```
SELECT ROWID, SYS_NAME FROM Account
```

Any subsequent queries against the Account table return data from the cache, which reduces response time. SQL queries can access both cached data and remote data (data stored in Salesforce that has not been assigned to a cache) in the same statement.

The caches maintained by the Salesforce driver are write-through caches. This means that, for any operation that modifies data in a table that is cached, the driver performs the operation on the remote data first and then updates the cache as much as possible.

To create, modify, refresh, or delete client-side data caches, use the following SQL statement extensions:

- Create Cache
- Alter Cache
- Refresh Cache
- Drop Cache

See the following sections for overviews of each extension. See "SQL statements and extensions" for descriptions of the syntax of these extensions.

See also

[Supported SQL statements and extensions](#) on page 183

Creating a cache

You create a cache using the Create Cache statement (see "Create Cache (EXT)"). A cache can be created on a single table or on a set of related tables. When creating a cache on a single table, you specify the name of the table to cache and can optionally specify a filter for the table. The filter determines whether the cache holds all of the data in the remote table or a subset of the data that matches the filter. You can also specify attributes for the Create Cache statement that determine:

- Whether the cache data is held on disk or in memory
- How often the cache data is refreshed
- Whether the cache is initially enabled
- Whether the driver checks to see if a refresh is needed at connect time

Creating a cache for a set of related tables is similar to creating a cache on a single table except that a primary table and one or more referencing tables are specified. This is useful if you want to cache a subset of data for a table and also cache data related to that subset of data. For example, you might have three tables, Account, Contact, and Opportunity, where both a contact and an opportunity belong to a particular account. Using a relational cache, you could specify that accounts that have had activity in the past year be cached, as well as caching the opportunities and contacts for only those cached accounts.

See also

[Create Cache \(EXT\)](#) on page 194

Modifying a cache definition

Once a cache has been created, you can modify the definition of the cache or set of related caches with the Alter Cache statement (see "Alter Cache (EXT)"). Only the attributes of the cache can be modified through the Alter Cache statement; the table or related set of tables cannot be changed and a single table cache cannot be changed to a relational cache.

Warning: Changing the attributes of a cache may cause the current data in the cache to be discarded and refetched from the remote data source.

See also

[Alter Cache \(EXT\)](#) on page 184

Disabling and enabling a cache

When a cache is defined on a table, all fetch operations performed on that table access the cache, essentially hiding the remote table from the application. At times, you may want an application to query the remote data instead of the cached data. For example, assume that a cache was created on Account with a filter set to cache accounts that have had activity in the past year. You may want to run a query to get information about an account that has not been active for two years. One alternative would be to drop the Account cache, run the query, and then recreate the cache on Account, but this can be problematic. First, you must recreate the cache and make sure it had the same attributes as before. Second, the data in the cache is discarded and needs to be refetched when the cache is recreated. Depending on the amount of cached data, this could take a significant amount of time. To address this type of issue, the Salesforce driver can temporarily disable a cache. When a cache is disabled, its definition and data are maintained. Any queries that reference a table with a disabled cache access the remote table. When you want to access cached data again, the cache can be enabled.

Refreshing cache data

To prevent the data in a cache from becoming out of date, the driver periodically refreshes the cache data with data from the remote data source. To minimize the amount of data that needs to be moved when a cache is refreshed, and the time required to refresh it, the driver checks to see which records in the remote table have been added, modified, or deleted since the last time the cache was refreshed. The driver retrieves only data for added or modified records and removes only deleted records from the cache. Your application can explicitly refresh the cache or the driver can refresh the cache automatically.

You can refresh a cache explicitly at any time by using the Refresh Cache statement (see Refresh Cache (EXT)). The Refresh Cache statement can also be used to perform a Clean (complete) refresh in addition to the standard optimized refresh. A Clean refresh discards all the data from the cache and repopulates it with data from the remote data source.

The driver also can refresh a cache automatically. When you create a cache, one of the attributes that you set is the refresh interval for the cache. During each cache query, the driver checks to see whether the time elapsed since the last refresh has exceeded the refresh interval for the cache. If it has, the driver refreshes the cache before satisfying the query.

Update operations to a table that is cached can trigger the driver to refresh the cache automatically. The caches maintained by the Salesforce driver are write-through caches. For any operation that modifies data in a table that is cached, the driver performs the operation on the remote data first and then updates the cache. The driver may not be able to update the cache with all modifications because some of the modified data may have been generated by the remote data source. For example, if a row is inserted but a value for all columns in the row is not required, any default values generated by the remote data source for columns not specified in the Insert statement would not be set in the cache. Because the driver cannot reflect all the changes made when a cached table is modified, it sets the cache state to dirty. When a cache state is dirty, the next query that attempts to fetch data from that cache causes the driver to refresh the cache before the fetch operation is performed. This allows the fetch to access the values populated by the remote data source.

The state of a cache can be viewed by selecting the STATUS column of the INFORMATION_SCHEMA.SYSTEM_CACHES table. See "SYSTEM_CACHES catalog table" for more information.

See also

[Refresh Cache \(EXT\)](#) on page 221

[SYSTEM_CACHES catalog table](#) on page 88

Dropping a cache

You can drop an existing cache using the Drop Cache statement (see "Drop Cache (EXT).") If a cache is a relational cache, the Drop Cache statement drops the cache for the primary table as well as the caches for the related tables.

Note: When a cache is dropped, all of the data in that cache is discarded.

See also

[Drop Cache \(EXT\)](#) on page 216

Cache metadata

The Salesforce driver maintains information about the caches that have been created. The driver provides two system tables to expose the cache information, the SYSTEM_CACHES table and the SYSTEM_CACHE_REFERENCES table.

The SYSTEM_CACHES and SYSTEM_CACHE_REFERENCES system tables exist in the INFORMATION_SCHEMA schema. See "Catalog tables" for a complete description of the contents of these system tables.

See also

[Catalog tables](#) on page 88

Catalog tables

The Salesforce driver provides a standard set of catalog tables that maintain the information returned by various ODBC catalog functions such as SQLTables, SQLColumns, SQLDescribeParam and SQLDescribeCol. If possible use the ODBC catalog functions to obtain this information instead of querying the catalog tables directly.

The driver also provides additional catalog tables that maintain metadata specific to the Salesforce driver. This section defines the catalog tables that provide Salesforce driver-specific information. The catalog tables are defined in the INFORMATION_SCHEMA schema.

SYSTEM_CACHES catalog table

The SYSTEM_CACHES catalog table stores the definitions of the caches created on remote tables. The data in the SYSTEM_CACHES table provides the name, type (single table or relational), status, and other information for each defined cache. The table name returned for a remote relational cache is the name of the primary table of the relational cache; however, its type is REMOTE RELATIONAL. You can query SYSTEM_CACHES to determine the caches currently defined by the driver. The values in the SYSTEM_CACHES table are read-only. The referenced tables of a relational cache can be determined by querying the SYSTEM_CACHE_REFERENCES catalog table (see "SYSTEM_CACHE_REFERENCES").

The following table describes the columns of the SYSTEM_CACHES table, which is sorted on the following columns: CACHE_TYPE, TABLE_SCHEMA, and TABLE_NAME.

Table 7: SYSTEM_CACHES Catalog Table

Column Name	Data Type	Description
TABLE_CAT	VARCHAR(128),NULLABLE	The catalog that contains the remote table on which the cache is defined. It is NULL for the Salesforce driver.
TABLE_SCHEM	VARCHAR(128),NULLABLE	The schema that contains the remote table on which the cache is defined.
TABLE_NAME	VARCHAR(128),NOT NULL	The name of the remote table on which the cache is defined.
CACHE_TYPE	VARCHAR(20),NOT NULL	The type cache, which can be either REMOTE TABLE or REMOTE RELATIONAL.
REFRESH_INTERVAL	INTEGER,NOT NULL	The refresh interval (in minutes).
INITIAL_CHECK	VARCHAR(20),NOT NULL	The value that defines when the initial refresh check is performed: ONFIRSTCONNECT or FIRSTUSE.
PERSIST	VARCHAR(20),NOT NULL	The value that defines whether the data in the cache is persisted past the lifetime of the connection: TEMPORARY, MEMORY, or DISK.
ENABLED	BOOLEAN,NOT NULL	The value that defines whether the cache is enabled for use with SQL statements: TRUE or FALSE.
CALL_LIMIT	INTEGER,NOT NULL	The maximum number of Web service calls that can be made when refreshing the cache. The value 0 indicates no call limit.
REFRESH_MODE	INTEGER,NOT NULL	For internal use only.
FILTER	VARCHAR(128),NULLABLE	The Where clause used to filter the rows that are cached.

Column Name	Data Type	Description
LAST_REFRESH	DATETIME, NULLABLE	The time, in Coordinated Universal Time (UTC), the cache was last refreshed.
STATUS	VARCHAR(30)	The Cache status. Valid values are: New: The cache has been created, but the data has not been populated. Initialized: The cache has been created and the data has been populated. Load aborted: The cache has been created, but the last attempt to populate the data failed. The cache is still valid. The next access attempts to populate the data again. Invalid: The cache is invalid. The second attempt to populate the data failed. Dirty: An insert or update operation has been performed on the cache and the cache has not been refreshed.

See also

[SYSTEM_CACHE_REFERENCES catalog table](#) on page 90

SYSTEM_CACHE_REFERENCES catalog table

The referenced tables in a relational cache can be determined by querying the SYSTEM_CACHE_REFERENCES system table. This table contains the names of the referenced tables as well as the name of the primary table with which they are associated.

The following table defines the columns of the SYSTEM_CACHES table, which is sorted on the following columns: TABLE_SCHEMA, TABLE_NAME, and REF_TABLE_NAME.

Table 8: SYSTEM_CACHE_REFERENCES

Column	Data Type	Description
PRIMARY_TABLE_CAT	VARCHAR (128), NULLABLE	The catalog that contains the primary table of the relational cache. It is NULL for the Salesforce driver.
PRIMARY_TABLE_SCHEM	VARCHAR (128), NULLABLE	The schema that contains the primary table of the relational cache.
PRIMARY_TABLE_NAME	VARCHAR (128), NOT NULL	The primary table of the relational cache.

REF_TABLE_NAME	VARCHAR (128), NOT NULL	The name of the referenced table.
RELATIONSHIP_NAME	VARCHAR(128), NOT NULL	The name of the foreign key relationship used to relate this table to the primary table or one of the other tables in the relational cache.

SYSTEM_REMOTE_SESSIONS catalog table

The system table named SYSTEM_REMOTE_SESSIONS stores information about each of the remote sessions that are active for a given database. The values in the SYSTEM_REMOTE_SESSION table are read-only.

The following table defines the columns of the SYSTEM_REMOTE_SESSIONS table, which is sorted on the following columns: SESSION_ID and SCHEMA.

Table 9: SYSTEM_REMOTE_SESSIONS Catalog Table

Column Name	Data Type	Description
SESSION_ID	INTEGER, NOT NULL	The connection (session) id with which the remote session is associated.
SCHEMA	VARCHAR(128), NOT NULL	The schema name that is mapped to the remote session.
TYPE	VARCHAR(30), NOT NULL	The remote session type. The current valid type is Salesforce.
INSTANCE	VARCHAR(128)	The remote session instance name or null if the remote data source does not have multiple instances. The Salesforce value for INSTANCE has the following form: <i>Organization_Name</i> [Sandbox] where <i>Organization_Name</i> is the organization name of the Salesforce instance to which the connection is established. If the connection is established to a sandbox of the organization, then the word Sandbox is added to the end of the name.
VERSION	VARCHAR(30), NOT NULL	The version of the remote data source to which the session is connected. For Salesforce, this is the version of the Web Service API the driver is using to connect to Salesforce.
CONFIG_OPTIONS	LONGVARCHAR, NOT NULL	The configuration options used to define the remote data model to relational data model mapping.

Column Name	Data Type	Description
SESSION_OPTIONS	LONGVARCHAR, NOT NULL	The options used to establish the remote connection. This typically is information needed to log into the remote data source. The password value is not displayed.
WS_CALL_COUNT	INTEGER, NOT NULL	The number of Web service calls made through this remote session. The value of the WS_CALL_COUNT column can be reset using the ALTER SESSION statement.
WS_AGGREGATE_CALL_COUNT	INTEGER, NOT NULL	The total of all of the Web service calls made to the same remote data source by all active connections using the same server name and user ID.
REST_AGGREGATE_CALL_COUNT	INTEGER, NOT NULL	The number of REST calls made by this connection. REST calls are used for bulk operations, invoking reports, and describing report parameters.

SYSTEM_SESSIONINFO catalog table

The system table named SYSTEM_SESSIONINFO describes details about your connection to Salesforce.

The following table defines the keys for the SYSTEM_SESSIONINFO table. The values change based on your data source settings.

Table 10: SYSTEM_SESSIONINFO Catalog Table

Key	Description
AUTOCOMMIT	Autocommit is always enabled.
DATABASE	The location and the filename prefix for the data mapping and configuration files.
DATABASE_READONLY	Indicates whether the database the session is connected to is read only.
DB_FILE_LOCATION	The fully qualified path to the directory or folder that contains the database and mapping files.
DB_FILE_PREFIX	The filename prefix of the database and mapping files the driver is using.
IDENTITY	Currently always zero.
MAXROWS	Currently always zero.
LOG_CONFIG_FILE	The fully qualified path to the directory or folder that contains the logging configuration file.
SCHEMA	The name of the remote Salesforce schema.
SESSION_ID	The ID for this session.
SESSION_READONLY	Indicates whether the session is read only.
USER	The user that is associated with this session.

SYSTEM_SESSIONS catalog table

The system table named SYSTEM_SESSIONS stores information about current system sessions. The values in the SYSTEM_SESSIONS table are read-only.

The following table defines the columns of the SYSTEM_SESSIONS table.

Table 11: SYSTEM_SESSIONS

Column	Data Type	Description
SESSION_ID	INTEGER, NOT NULL	A unique ID that identifies this session. The system function CURSESSIONID() returns the session ID associated with the connection.

Column	Data Type	Description
CONNECTED	DATETIME, NOT NULL	The date and time the session was established.
USER_NAME	VARCHAR (128), NOT NULL	The name of the schema map that the session is using.
IS_ADMIN	BOOLEAN	For internal use only.
AUTOCOMMIT	BOOLEAN, NOT NULL	For future use.
READONLY	BOOLEAN, NOT NULL	True if the connection is in read-only mode. The READONLY status is based on whether the connection has been explicitly set to read-only mode by the Read Only connection option.
MAXROWS	INTEGER, NOT NULL	For future use.
LAST_IDENTITY	BIGINT, NULLABLE	For future use.
TRANSACTION_SIZE	INTEGER, NOT NULL	For future use.
CURRENT_SCHEMA	VARCHAR (128), NOT NULL	The current schema for the session. The current schema may be changed using the ALTER SESSION SET CURRENT_SCHEMA statement.
STMT_CALL_LIMIT	INTEGER, NOT NULL	The maximum number of Web service calls that the driver uses in attempting to execute a query to a remote data source. The statement call limit for the session may be changed via the ALTER SESSION SET STMT_CALL_LIMIT statement.

Refer to "ODBC API and scalar functions" in the *Progress DataDirect for ODBC Drivers Reference* for additional information.

Reports

The Salesforce driver exposes reports defined on a Salesforce instance as stored procedures. An application can obtain a list of the reports defined on a Salesforce instance by calling the SQLProcedures catalog function. The names of the reports that can be invoked through the driver are listed in the PROCEDURE_NAME name column of the SQLProcedures results.

Salesforce organizes reports into folders. The Salesforce driver incorporates the folder name and report name into the procedure name reported by SQLProcedures. The driver creates the reported procedure name by prepending the folder name to the report name using an underscore to join them. Additionally, any spaces in the report or folder names are replaced with an underscore character. Like all identifier name metadata returned by the driver, the procedure name is uppercase. For example, if a report named Opportunity Pipeline is in the folder Opportunity Reports, it would be rendered as:

OPPORTUNITY_REPORTS_OPPORTUNITY_PIPELINE

An application invokes a report using the standard Call escape syntax, `{call report name}`, and ODBC mechanisms for calling a stored procedure that returns a resultset. The following example shows one way to invoke the Opportunity Pipeline report:

```
SQLRETURN      retVal;
HSTMT          hStmt = NULL;
SQLWCHAR*      sql;
sql = L"{call OPPORTUNITY_REPORTS_OPPORTUNITY_PIPELINE}";
retVal = SQLExecDirect(hStmt, sql, SQL_NTS);
if (SQL_SUCCESS == retVal) {
    //      process results
}
```

Note: The API used by the driver to obtain the list of reports and execute the reports is not an API that is documented by Salesforce. This API may change or may not be supported in the future.

Note: When passing parameters to stored procedures, reports are not supported.

Using security

The driver supports the *data encryption* security feature, which converts data into a form that cannot be easily understood by unauthorized users.

Authentication

Authentication ensures that only the authorized users are allowed to connect to a Salesforce instance.

The Salesforce driver supports the following methods of authentication:

- *User ID/password authentication* authenticates the user to a Salesforce instance using the user ID and password specified by the application.
- *OAuth 2.0 authentication* allows the user to authenticate to a Salesforce instance without having to specify user ID and password.

See also

[Security tab](#) on page 62

Configuring OAuth 2.0 authentication

The driver supports OAuth 2.0 to access Salesforce resources. Before you can configure the driver for OAuth, you must register your client application with Salesforce and obtain information such as the client ID, the client secret, and the refresh token. The following workflow describes the process for setting up OAuth 2.0 access.

1. [Register your application with Salesforce](#). See this topic for step-by-step instructions for registering your application.
2. [Obtain client ID, client secret, and scopes](#). If the application has already been registered, see this topic for steps to obtain information required to configure the driver.
3. [Obtain access and refresh tokens with Postman](#). To configure the driver, you will need to specify either an access token or a refresh token. This topic provides instructions to obtain these tokens using Postman.
4. [Configure the driver to use OAuth 2.0](#). You can configure the driver to access Salesforce resources by specifying the connection options described in this topic.

Note: For more information, refer to the [Authorize Apps with OAuth](#) section of the *Salesforce Help*.

Registering your application with Salesforce

Prerequisites:

- The callback URL (or redirect URI) for the client application.

Take the following steps to register your application as a Salesforce Connected App.

1. Sign in to Salesforce.
`https://login.salesforce.com`
2. Navigate to the **Setup** page.
3. Navigate to **Apps > App Manager**.
4. Click **New Connected App** to open the **New Connected App** form.
5. Complete and save the **New Connected App** form. The following list of parameters are required for OAuth.
 - **Basic Information**
 - **Connected App Name:** An application name must be specified.
 - **API Name:** The API Name is based on the application name and generated automatically, but you may modify it.
 - **Contact Email:** A contact email must be specified.
 - **API (Enable OAuth Settings)**
 - **Enable OAuth Settings:** Check this box to enable OAuth.
 - **Callback URL (OAuth Redirect URI):** Specify one or more callback URLs. This is the URL that Salesforce calls back to your application during the authorization code flow.

Notes:

- If you are using **Postman** to implement or test OAuth connectivity, the callback URL may be either of the following:

`https://oauth.pstmn.io/v1/callback`

`https://www.getpostman.com/oauth2/callback`

For details, see [Obtaining access and refresh tokens with Postman](#).

- If you are using **Hybrid Data Pipeline**, the callback URL is:

`https://MyHDPDomain:8443/hdpui/oauth2callback`

where *MyHDPDomain* is the name of the Hybrid Data Pipeline host or load balancer.

- **Selected OAuth Scopes:** Choose OAuth scopes to define permissions for the application. For example, to grant an application full access to Salesforce resources at any time, you would specify the following scopes:
 - `Full access (full)`: Allows access to all data accessible by the logged-in user, and encompasses all other scopes.
 - `Perform requests at any time (refresh_token, offline_access)`: Allows a refresh token to be returned when the requesting client is eligible to receive one.

Note: The values you select should be saved to a secure location. You will need to specify these values if you are using the authorization code flow to obtain access and refresh tokens, as described in [Obtaining access and refresh tokens with Postman](#).

6. After saving the **New Connected App** form, the page for the Connected App will be displayed.
7. Click **Manage Consumer Details** to obtain the client ID and client secret (Consumer Key and Consumer Secret).

Note: The values for the client ID and secret should be saved to a secure location. You will need to specify these values for the Client ID and Client Secret connection options.

Results:

You have successfully registered your application as a Salesforce Connected App, as well as obtained the OAuth client ID and client secret.

Obtaining the client ID, client secret, and scopes

Take the following steps to obtain the client ID, the client secret, and the scope.

1. Sign in to Salesforce.
`https://login.salesforce.com`
2. Navigate to the **Setup** page.
3. Navigate to **Apps > App Manager**.
4. Select the application for which you are obtaining the client ID and secret from the list of Connected Apps.
5. For the scopes, navigate to **API (Enable OAuth Settings) > Selected OAuth Scopes**.

In most cases, the client application will be granted full access to Salesforce resources at any time. In this scenario, the following scopes would be specified:

- `Full access (full)`: Allows access to all data accessible by the logged-in user, and encompasses all other scopes.
- `Perform requests at any time (refresh_token, offline_access)`: Allows a refresh token to be returned when the requesting client is eligible to receive one.

Note: You will need to specify scopes if you use are using the authorization code flow to obtain access and refresh tokens, as described in [Obtaining access and refresh tokens with Postman](#).

6. For the client ID and client secret (Consumer Key and Consumer Secret), click **Manage Consumer Details**.

Note: The values for the client ID and secret should be saved to a secure location. You will need to specify these values for the Client ID and Client Secret connection options.

Results:

You have obtained the OAuth client ID and client secret.

Obtaining access and refresh tokens with Postman

Prerequisites:

- The client ID and client secret (Consumer Key and Consumer Secret) for the Salesforce Connected App.
- The Postman callback URL (or redirect URI) for the client application.

The following steps show how to use Postman to obtain access and refresh tokens from Salesforce.

1. Open Postman.
2. Select the **Authorization** tab.
3. Select **OAuth 2.0**. Then, enter required values. For example:

Token Name: MySalesforceToken

Grant Type: Authorization Code

Callback URL: *Depending on the version of Postman you are using, this may be either of the following endpoints:*

- `https://oauth.pstmn.io/v1/callback`
- `https://www.getpostman.com/oauth2/callback`

Auth URL: `https://login.salesforce.com/services/oauth2/authorize`

Token URL: `https://login.salesforce.com/services/oauth2/token`

Client ID: `client-id`

Client Secret: `client-secret`

Scope: `scope` where `scope` is the list of scopes that define application permissions. In most cases, the client application will be granted full access to Salesforce resources at any time. In this scenario, the following scopes would be specified:

- `Full access (full)`: Allows access to all data accessible by the logged-in user, and encompasses all other scopes.
 - `Perform requests at any time (refresh_token, offline_access)`: Allows a refresh token to be returned when the requesting client is eligible to receive one.
4. Press **Get New Access Token**.
- Salesforce returns the access and refresh tokens to Postman.

Results:

You have obtained access and refresh tokens for OAuth access to Salesforce.

Configuring the driver to use OAuth 2.0**Prerequisites:**

- Client application registered as a Salesforce Connected App.
- The client ID and client secret (Consumer Key and Consumer Secret)
- An access token or refresh token

After you have registered your client application as a Salesforce Connected App and obtained the required OAuth information, you may configure the driver to access Salesforce resources using OAuth 2.0.

To configure the driver, set the following connection options:

- Set the Authentication Method option to `oauth2.0`.
- Set the Schema Map option to specify either the name or the absolute path and name of the configuration file where the map of the Salesforce data model is written. Note that a value for the Schema Map option must be specified every time you authenticate to a Salesforce instance using OAuth 2.0.
- Set the Client ID option to specify the consumer key for your application. See [Registering your application with Salesforce](#) or [Obtaining the client ID, client secret, and scopes](#).
- Set either of the following options:
 - Set the Access Token option to specify the access token you have obtained from Salesforce. Generally, an access token is available for a short period of time and may only be used for a single session.
 - Set the Refresh Token option to specify the refresh token you have obtained from Salesforce. With a valid refresh token, the driver can obtain a new access token. In this way, a refresh token enables application access to Salesforce for multiple sessions over an extended period of time, as dictated by your organization's policies.

If a value for the Access Token option is not specified, the driver uses the value of the Refresh Token option to make a connection. If both values are not specified, the driver cannot make a successful connection. If both are specified, the driver ignores the Access Token value and uses the Refresh Token value to generate a new Access Token value.

See [Obtaining access and refresh tokens with Postman](#) for details on how to obtain access or refresh tokens using Postman.

- Optionally, set the Client Secret option to specify the consumer secret for your application. See [Registering your application with Salesforce](#) or [Obtaining the client ID, client secret, and scopes](#).

The following examples show how to connect to a Salesforce instance using OAuth2.0 authentication.

Using a connection string:

```
DRIVER=DataDirect 8.0 Salesforce;AuthenticationMethod=oauth2.0;
SchemaMap=ABC;clientId=RaARBTsXZTeN4Qx67qPLOS;RefreshToken=YaTARBsRZLeM4Px47qSOLP;
AccessToken=ZbTARBsRZLeM4Px56qOLPS
```

Using the `odbc.ini` file:

```
Driver=ODBCHOME/lib/xxsfrc28.yy
AuthenticationMethod=oauth2.0
SchemaMap=ABC
ClientId=RaARBTsXZTeN4Qx67qPLOS
RefreshToken=YaTARBsRZLeM4Px47qSOLP
AccessToken=ZbTARBsRZLeM4Px56qOLPS
```

See also

[Authentication Method](#) on page 131

[Access Token](#) on page 129

[Refresh Token](#) on page 170

[Client ID](#) on page 138

[Client Secret](#) on page 139

[Schema Map](#) on page 172

Data encryption across the network

If your database connection is not configured to use data encryption, data is sent across the network in a format that is designed for fast transmission and can be decoded by interceptors, given some time and effort. For example, text data is often sent across the wire as clear text. Because this format does not provide complete protection from interceptors, you may want to use data encryption to provide a more secure transmission of data.

For example, you may want to use data encryption in the following scenarios:

- You have offices that share confidential information over an intranet.
- You send sensitive data, such as credit card numbers, over a database connection.
- You need to comply with government or industry privacy and security requirements.

Your Progress DataDirect *for* ODBC driver supports Transport Layer Security (TLS) and Secure Sockets Layer (SSL). TLS/SSL are industry-standard protocols for sending encrypted data over database connections. TLS/SSL secures the integrity of your data by encrypting information and providing client/server authentication.

Note: Data encryption may adversely affect performance because of the additional overhead (mainly CPU usage) required to encrypt and decrypt data.

TLS/SSL encryption

The driver supports TLS/SSL data encryption. TLS/SSL works by allowing the client and server to send each other encrypted data that only they can decrypt. SSL negotiates the terms of the encryption in a sequence of events known as the *handshake*. During the handshake, the driver negotiates the highest TLS/SSL protocol available. The result of this negotiation determines the encryption cipher suite to be used for the TLS/SSL session.

The encryption cipher suite defines the type of encryption that is used for any data exchanged through a TLS/SSL connection. Some cipher suites are very secure and, therefore, require more time and resources to encrypt and decrypt data, while others provide less security, but are also less resource intensive.

The handshake involves the following types of authentication:

- *TLS/SSL server authentication* requires the server to authenticate itself to the client.
- *TLS/SSL client authentication* is optional and requires the client to authenticate itself to the server after the server has authenticated itself to the client.

Note: The version of TLS/SSL that is used and which cryptographic algorithm is used depends on which JVM you are using. Refer to your JVM documentation for more information about its TLS/SSL support.

Certificates

SSL requires the use of a digitally-signed document, an x.509 standard certificate, for authentication and the secure exchange of data. The purpose of this certificate is to tie the public key contained in the certificate securely to the person/company that holds the corresponding private key. Your Progress DataDirect for ODBC drivers supports many popular formats. Supported formats include:

- DER Encoded Binary X.509
- Base64 Encoded X.509
- PKCS #12 / Personal Information Exchange

TLS/SSL server authentication

When the client makes a connection request, the server presents its public certificate for the client to accept or deny. The client checks the issuer of the certificate against a list of trusted Certificate Authorities (CAs) that resides in an encrypted file on the client known as a *truststore*. If the certificate matches a trusted CA in the truststore, an encrypted connection is established between the client and server. If the certificate does not match, the connection fails and the driver generates an error.

Most truststores are password-protected. The driver must be able to locate the truststore and unlock the truststore with the appropriate password. Two connection options are available to the driver to provide this information: Trust Store (Truststore) and Trust Store Password (TruststorePassword). The value of Trust Store is a pathname that specifies the location of the truststore file. The value of Trust Store Password is the password required to access the contents of the truststore.

Alternatively, you can configure the driver to trust any certificate sent by the server, even if the issuer is not a trusted CA. Allowing a driver to trust any certificate sent from the server is useful in test environments because it eliminates the need to specify truststore information on each client in the test environment. Setting the Validate Server Certificate (ValidateServerCertificate) connection option to false allows the driver to accept any certificate returned from the server regardless of whether the issuer of the certificate is a trusted CA.

Finally, the connection option, Host Name In Certificate (HostNameInCertificate), allows an additional method of server verification. When a value is specified for Host Name In Certificate, it must match the common name of the host in the Subject of the certificate. This prevents malicious intervention between the client and the server and ensures that the driver is connecting to the server that was requested.

TLS/SSL client authentication

If the server is configured for TLS/SSL client authentication, the server asks the client to verify its identity after the server identity has been proven. Similar to server authentication, the client sends a public certificate to the server to accept or deny. The client stores its public certificate in an encrypted file known as a *keystore*. Public certificates are paired with a private key in the keystore. To send the public certificate, the driver must access the private key.

Like the truststore, most keystores are password-protected. The driver must be able to locate the keystore and unlock the keystore with the appropriate password. Two connection options are available to the driver to provide this information: Keystore (KeyStore) and Keystore Password (KeyStorePassword). The value of KeyStore is a pathname that specifies the location of the keystore file. The value of Keystore Password is the password required to access the keystore.

The private keys stored in a keystore can be individually password-protected. In many cases, the same password is used for access to both the keystore and to the individual keys in the keystore. It is possible, however, that the individual keys are protected by passwords different from the keystore password. The driver needs to know the password for an individual key to be able to retrieve it from the keystore. An additional connection option, Key Password (KeyPassword), allows you to specify a password for an individual key.

Summary of security-related options

The following table summarizes how security-related connection options work with the drivers. See "Connection option descriptions" for details about configuring the options.

Table 12: Summary: Security Connection Options

Option	Description
Security Token (SecurityToken)	Specifies the security token required to make a connection to a Salesforce instance that is configured for a security token. If a security token is required and you do not supply one, the driver returns an error indicating that an invalid user or password was supplied. Contact your Salesforce administrator to find out if a security token is required. Default: None
User Name (LogonID)	The default user ID and domain used to connect to your database. For example, jsmith@defcorp.com. Default: None

See also

[Connection option descriptions](#) on page 123

Using DataDirect Connection Pooling

Connection pooling allows you to *reuse* connections rather than creating a new one every time the driver needs to establish a connection to the underlying database. Your driver enables connection pooling without requiring changes to your client application.

Note: Connection pooling works only with connections that are established using `SQLConnect` or `SQLDriverConnect` with the `SQL_DRIVER_NO_PROMPT` argument and only with applications that are thread-enabled.

DataDirect connection pooling that is implemented by the DataDirect driver is different than connection pooling implemented by the Windows Driver Manager. The Windows Driver Manager opens connections dynamically, up to the limits of memory and server resources. DataDirect connection pooling, however, allows you to control the number of connections in a pool through the Min Pool Size (minimum number of connections in a pool) and Max Pool Size (maximum number of connections in a pool) connection options. In addition, DataDirect connection pooling is cross-platform, allowing it to operate on UNIX and Linux. See "Connection option descriptions" for details about how the connection options manage DataDirect connection pooling.

Important: On a Windows system, do not use both Windows Driver Manager connection pooling and DataDirect connection pooling at the same time.

See also

[Connection option descriptions](#) on page 123

Creating a connection pool

Each connection pool is associated with a specific connection string. By default, the connection pool is created when the first connection with a unique connection string connects to the data source. The pool is populated with connections up to the minimum pool size before the first connection is returned. Additional connections can be added until the pool reaches the maximum pool size. If the Max Pool Size option is set to 10 and all connections are active, a request for an eleventh connection has to wait in queue for one of the 10 pool connections to become idle. The pool remains active until the process ends or the driver is unloaded.

If a new connection is opened and the connection string does not exactly match an existing pool, a new pool must be created. By using the same connection string, you can enhance the performance and scalability of your application.

Adding connections to a pool

A connection pool is created in the process of creating each unique connection string that an application uses. When a pool is created, it is populated with enough connections to satisfy the minimum pool size requirement, set by the Min Pool Size connection option. The maximum pool size is set by the Max Pool Size connection option. If an application needs more connections than the number set by Min Pool Size, the driver allocates additional connections to the pool until the number of connections reaches the value set by Max Pool Size.

Once the maximum pool size has been reached and no usable connection is available to satisfy a connection request, the request is queued in the driver. The driver waits for the length of time specified in the Login Timeout connection option for a usable connection to return to the application. If this time period expires and a connection has not become available, the driver returns an error to the application.

A connection is returned to the pool when the application calls SQLDisconnect. Your application is still responsible for freeing the handle, but this does not result in the database session ending.

Removing connections from a pool

A connection is removed from a connection pool when it exceeds its lifetime as determined by the Load Balance Timeout connection option. In addition, DataDirect has created connection attributes described in the following table to give your application the ability to reset connection pools. If connections are in use at the time of these calls, they are marked appropriately. When SQLDisconnect is called, the connections are discarded instead of being returned to the pool.

Table 13: Pool Reset Connection Attributes

Connection Attribute	Description
SQL_ATTR_CLEAR_POOLS Value: SQL_CLEAR_ALL_CONN_POOL	Calling SQLSetConnectAttr (SQL_ATTR_CLEAR_POOLS, SQL_CLEAR_ALL_CONN_POOL) clears all the connection pools associated with the driver that created the connection. This is a write-only connection attribute. The driver returns an error if SQLGetConnectAttr (SQL_ATTR_CLEAR_POOLS) is called.
SQL_ATTR_CLEAR_POOLS Value: SQL_CLEAR_CURRENT_CONN_POOL	Calling SQLSetConnectAttr (SQL_ATTR_CLEAR_POOLS, SQL_CLEAR_CURRENT_CONN_POOL) clears the connection pool that is associated with the current connection. This is a write-only connection attribute. The driver returns an error if SQLGetConnectAttr (SQL_ATTR_CLEAR_POOLS) is called.

Note: By default, if removing a connection causes the number of connections to drop below the number specified in the Min Pool Size option, a new connection is not created until an application needs one.

Handling dead connections in a pool

What happens when an idle connection loses its physical connection to the database? For example, suppose the database server is rebooted or the network experiences a temporary interruption. An application that attempts to connect could receive errors because the physical connection to the database has been lost.

Your driver handles this situation transparently to the user. The application does not receive any errors on the connection attempt because the driver simply returns a connection from a connection pool. The first time the connection handle is used to execute a SQL statement, the driver detects that the physical connection to the server has been lost and attempts to reconnect to the server *before* executing the SQL statement. If the driver can reconnect to the server, the result of the SQL execution is returned to the application; no errors are returned to the application.

The driver uses connection failover option values, if they are enabled, when attempting this seamless reconnection; however, it attempts to reconnect even if these options are not enabled.

Note: If the driver cannot reconnect to the server (for example, because the server is still down), an error is returned indicating that the reconnect attempt failed, along with specifics about the reason the connection failed.

The technique that Progress DataDirect uses for handling dead connections in connection pools allows for maximum performance of the connection pooling mechanism. Some drivers periodically test the server with a dummy SQL statement while the connections sit idle. Other drivers test the server when the application requests the use of the connection from the connection pool. Both of these approaches add round trips to the database server and ultimately slow down the application during normal operation.

Connection pool statistics

Progress DataDirect has created a connection attribute to monitor the status of the DataDirect for ODBC connection pools. This attribute, which is described in the following table, allows your application to fetch statistics for the pool to which a connection belongs.

Table 14: Pool Statistics Connection Attribute

Connection Attribute	Description
SQL_ATTR_POOL_INFO Value: SQL_GET_POOL_INFO	Calling SQLGetConnectAttr (SQL_ATTR_POOL_INF, SQL_GET_POOL_INFO) returns a PoolInfoStruct that contains the statistics for the connection pool to which this connection belongs. This PoolInfoStruct is defined in qesqltext.h. For example:SQLGetConnectAttr(hdbc, SQL_ATTR_POOL_INFO, PoolInfoStruct *, SQL_LEN_BINARY_ATTR(PoolInfoStruct), &len); This is a read-only connection attribute. The driver returns an error if SQLSetConnectAttr (SQL_ATTR_POOL_INFO) is called.

Configuring pooling-related options

The following table summarizes how connection pooling-related connection options work with the drivers. See "Connection option descriptions" for details about configuring the options.

Table 15: Summary: Connection Pooling Connection Options

Option	Characteristic
Connection Pooling (Pooling)	Specifies whether to use the driver's connection pooling. If set to 1 (Enabled), the driver uses connection pooling. If set to 0 (Disabled), the driver does not use connection pooling. Default: 0 (Disabled)
Connection Reset (ConnectionReset)	Determines whether the state of connections that are removed from the connection pool for reuse by the application is reset to the initial configuration of the connection. If set to 1 (Enabled), the state of connections removed from the connection pool for reuse by an application is reset to the initial configuration of the connection. If set to 0 (Disabled), the state of connections is not reset. Default: 0 (Disabled)
LoadBalance Timeout (LoadBalanceTimeout)	An integer value to specify the amount of time, in seconds, to keep connections open in a connection pool. Default: 0
Max Pool Size (MaxPoolSize)	An integer value to specify the maximum number of connections within a single pool. Default: 100
Min Pool Size (MinPoolSize)	An integer value to specify the minimum number of connections that are opened and placed in a connection pool when it is created. Default: 0

See also

[Connection option descriptions](#) on page 123

Using identifiers

Identifiers are used to refer to objects exposed by the driver, such as tables, columns, or caches. The driver supports both unquoted and quoted identifiers for naming objects. An unquoted identifier must start with an ASCII alpha character and can be followed by zero or more ASCII alpha or numeric characters. Unquoted identifiers are converted to uppercase before being used.

Quoted identifiers must be enclosed in double quotation marks (""). A quoted identifier can contain any Unicode character, including the space character, and is case-sensitive. The Salesforce driver recognizes the Unicode escape sequence \uxxxx as a Unicode character. You can specify a double quotation mark in a quoted identifier by escaping it with a double quotation mark.

The maximum length of both quoted and unquoted identifiers is 128 characters.

Note: When object names are passed as arguments to catalog functions, the case of the value must match the case of the name in the database. If an unquoted identifier name was used when the object was created, the value passed to the catalog function must be uppercase because unquoted identifiers are converted to uppercase before being used. If a quoted identifier name was used when the object was created, the value passed to the catalog function must match the case of the name as it was defined. Object names in results returned from catalog functions are returned in the case that they are stored in the database.

Timeouts

The following types of timeout situations can occur when connecting to Salesforce:

- **Session timeouts.** Most remote data sources impose a limit on the duration of active sessions, meaning a session can fail with a session timeout error if the session extends past the limit. This is particularly true when connection pooling is used. The driver automatically attempts to re-establish a new session if the driver receives a session timeout error from a data source. The driver uses the initial servername, port (if appropriate), remote user ID, and remote password (encrypted) to re-establish the session. If the attempt fails, the driver returns an error indicating that the session timed out and the attempt to re-establish the session failed.
- **Web service request timeouts.** You can configure the driver to never time out while waiting for a response to a Web service request or to wait for a specified interval before timing out by setting the connection option `WSTimeout`. For fetch requests only, if the request times out, you can configure driver to retry the request a specified number of times by setting the `WSRetry Count` connection option. If all subsequent attempts to retry a request fails, the driver returns an error indicating that the service request timed out and the subsequent requests failed. See "Connection option descriptions" for details on the `WS Timeout` and `WS Retry Count` connection options.

See also

[WSTimeout](#) on page 181

[WSRetry Count](#) on page 180

[Connection option descriptions](#) on page 123

Views and remote/local tables

You can create views with the Create View statement. A view is like a named query. The view can refer to any combination of remote and local tables as well as other views.

You can create a remote or local table using the Create Table statement. A remote table is a Salesforce object and is exposed in the SFORCE schema. A local table is maintained by the driver and is local to the machine on which the driver is running. A local table is exposed in the PUBLIC schema.

See "SQL statements and extensions" for details on the Create View and Create Table statements and other SQL statements supported by the driver.

See also

[Supported SQL statements and extensions](#) on page 183

SQL support

See "Supported SQL statements and extensions" for information about the SQL statements and extensions supported by the Salesforce driver.

See also

[Supported SQL statements and extensions](#) on page 183

Isolation and lock levels supported

Salesforce supports isolation level 0 (read uncommitted).

Refer to "Locking and isolation levels" in the *Progress DataDirect for ODBC Drivers Reference* for details.

Unicode support

The Salesforce driver is fully Unicode enabled. On UNIX and Linux platforms, the driver supports both UTF-8 and UTF-16. On Windows platforms, the Salesforce driver supports UCS-2/UTF-16 only.

The driver supports the Unicode ODBC W (Wide) function calls, such as SQLConnectW. This allows the Driver Manager to transmit these calls directly to the driver. Otherwise, the Driver Manager would incur the additional overhead of converting the W calls to ANSI function calls, and vice versa.

See "UTF-16 applications on UNIX and Linux" for related details.

Refer to "Internationalization, localization, and Unicode" in the *Progress DataDirect for ODBC Drivers Reference* for details.

See also

[UTF-16 applications on UNIX and Linux](#) on page 53

Parameter metadata support

The driver supports returning parameter metadata as described in this section.

Insert and Update statements

The driver supports returning parameter metadata for the following forms of Insert and Update statements:

- `INSERT INTO FOO VALUES(?, ?, ?)`
- `INSERT INTO FOO (COL1, COL2, COL3) VALUES(?, ?, ?)`
- `UPDATE FOO SET COL1=?, COL2=?, COL3=? WHERE COL1 operator ? [{AND | OR} COL2 operator ?]`

where:

operator

is any of the following SQL operators:

=, <, >, <=, >=, and <>

.

Select statements

The driver supports returning parameter metadata for Select statements that contain parameters in ANSI SQL 92 entry-level predicates, for example, such as COMPARISON, BETWEEN, IN, LIKE, and EXISTS predicate constructs. Refer to the ANSI SQL reference for detailed syntax.

Parameter metadata can be returned for a Select statement if one of the following conditions is true:

- The statement contains a predicate value expression that can be targeted against the source tables in the associated FROM clause. For example:

```
SELECT * FROM FOO WHERE BAR > ?
```

In this case, the value expression "BAR" can be targeted against the table "FOO" to determine the appropriate metadata for the parameter.

- The statement contains a predicate value expression part that is a nested query. The nested query's metadata must describe a single column. For example:

```
SELECT * FROM FOO WHERE (SELECT X FROM Y WHERE Z = 1) < ?
```

The following Select statements show further examples for which parameter metadata can be returned:

```
SELECT COL1, COL2 FROM FOO WHERE COL1 = ? AND COL2 > ?
SELECT ... WHERE COLNAME = (SELECT COL2 FROM T2 WHERE COL3 = ?)
SELECT ... WHERE COLNAME LIKE ?
SELECT ... WHERE COLNAME BETWEEN ? AND ?
SELECT ... WHERE COLNAME IN (?, ?, ?)
SELECT ... WHERE EXISTS(SELECT ... FROM T2 WHERE COL1 < ?)
```

ANSI SQL 92 entry-level predicates in a WHERE clause containing GROUP BY, HAVING, or ORDER BY statements are supported. For example:

```
SELECT * FROM T1 WHERE COL = ? ORDER BY 1
```

Joins are supported. For example:

```
SELECT * FROM T1,T2 WHERE T1.COL1 = ?
```

Fully qualified names and aliases are supported. For example:

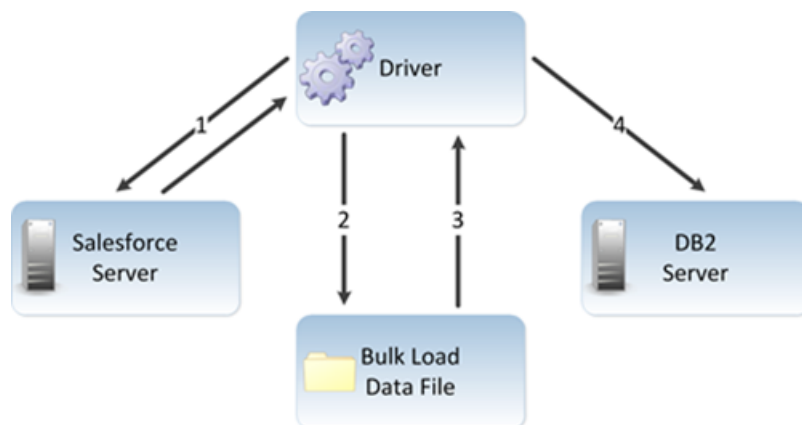
```
SELECT A, B, C, D FROM T1 AS A, T2 AS B WHERE A.A = ? AND B.B = ?
```

DataDirect Bulk Load

The driver supports DataDirect Bulk Load, a feature that allows your application to send large numbers of rows of data to a Salesforce instance. The driver sends data to a Salesforce instance using the Salesforce Bulk API instead of the Web Service API. Similar to batch operations, using the Bulk API significantly reduces the number of Web service calls the driver uses to transfer data and may improve performance.

Bulk load operations are accomplished by exporting the results of a query from a database into a comma-separated value (CSV) file, a bulk load data file. The driver then loads the data from bulk load data file into a different database. The file can be used by any DataDirect *for* ODBC drivers. In addition, the bulk load data file is supported by other DataDirect product lines that feature bulk loading, for example, a DataDirect Connect for ADO.NET data provider that supports bulk load.

Suppose that you had customer data on a Salesforce server and need to export it to a DB2 server. The driver would perform the following steps:



1. Application using Salesforce driver sends query to and receives results from Salesforce server.
2. Driver exports results to bulk load data file.
3. Driver retrieves results from bulk load data file.
4. Driver bulk loads results on DB2 server.

Bulk export and load methods

You can take advantage of DataDirect Bulk Load either through the Driver setup dialog or programmatically.

Applications that are already coded to use parameter array batch functionality can leverage DataDirect Bulk Load features through the Enable Bulk Load connection option on the Bulk tab of the Driver setup dialog. Enabling this option automatically converts the parameter array batch operation to use the Salesforce bulk load protocol without any code changes to your application.

If you are not using parameter array batch functionality, the bulk operation buttons **Export Table** and **Load Table** on the Bulk tab of the driver Setup dialog also allow you to use bulk load functionality without any code changes. See the individual driver chapters for a description of the Bulk tab.

If you want to integrate bulk load functionality seamlessly into your application, you can include code to use the bulk load functions exposed by the driver.

For your applications to use DataDirect Bulk Load functionality, they must obtain driver connection handles and function pointers, as follows:

1. Use `SQLGetInfo` with the parameter `SQL_DRIVER_HDBC` to obtain the driver's connection handle from the Driver Manager.
2. Use `SQLGetInfo` with the parameter `SQL_DRIVER_HLIB` to obtain the driver's shared library or DLL handle from the Driver Manager.
3. Obtain function pointers to the bulk load functions using the function name resolution method specific to your operating system. The `bulk.c` example program shipped with the drivers contains the function `resolveName` that illustrates how to obtain function pointers to the bulk load functions.

This is detailed in the code samples that follow.

Exporting data from a database

You can export data from a database in one of three ways:

- From a table by using the driver Setup dialog
- From a table by using DataDirect functions
- From a result set by using DataDirect statement attributes

From the DataDirect driver Setup dialog, select the **Bulk** tab and click **Export Table**. See "Bulk tab" for a description of this procedure.

Your application can export a table using the DataDirect functions `ExportTableToFile` (ANSI application) or `ExportTableToFileW` (Unicode application). The application must first obtain driver connection handles and function pointers, as shown in the following example:

```
HDBC      hdbc;
HENV      henv;
void      *driverHandle;
HMODULE   hmod;
PExportTableToFile exportTableToFile;

char      tableName[128];
char      fileName[512];
char      logFile[512];
int       errorTolerance;
int       warningTolerance;
int       codePage;

/* Get the driver's connection handle from the DM.
   This handle must be used when calling directly into the driver. */
rc = SQLGetInfo (hdbc, SQL_DRIVER_HDBC, &driverHandle, 0, NULL);
if (rc != SQL_SUCCESS) {
    ODBC_error (henv, hdbc, SQL_NULL_HSTMT);
}
```

```

    EnvClose (henv, hdbc);
    exit (255);
}

/* Get the DM's shared library or DLL handle to the driver. */

rc = SQLGetInfo (hdbc, SQL_DRIVER_HLIB, &hmod, 0, NULL);
if (rc != SQL_SUCCESS) {
    ODBC_error (henv, hdbc, SQL_NULL_HSTMT);
    EnvClose (henv, hdbc);
    exit (255);
}
exportTableToFile = (PExportTableToFile)
    resolveName (hmod, "ExportTableToFile");
if (! exportTableToFile) {
    printf ("Cannot find ExportTableToFile!\n");
    exit (255);
}

rc = (*exportTableToFile) (
    driverHandle,
    (const SQLCHAR *) tableName,
    (const SQLCHAR *) fileName,
    codePage,
    errorTolerance, warningTolerance,
    (const SQLCHAR *) logFile);
if (rc == SQL_SUCCESS) {
    printf ("Export succeeded.\n");
}
else {
    driverError (driverHandle, hmod);
}

```

Your application can export a result set using the DataDirect statement attributes `SQL_BULK_EXPORT` and `SQL_BULK_EXPORT_PARAMS`.

The export operation creates a bulk load data file with a `.csv` extension in which the exported data is stored. For example, assume that a Salesforce source table named `GBMAXTABLE` contains four columns. The resulting bulk load data file `GBMAXTABLE.csv` containing the results of a query would be similar to the following:

```

1,0x6263,"bc","bc"
2,0x636465,"cde","cde"
3,0x64656667,"defg","defg"
4,0x6566676869,"efghi","efghi"
5,0x666768696a6b,"fghijk","fghijk"
6,0x6768696a6b6c6d,"ghijklm","ghijklm"
7,0x68696a6b6c6d6e6f,"hijklmno","hijklmno"
8,0x696a6b6c6d6e6f7071,"ijklmnopq","ijklmnopq"
9,0x6a6b6c6d6e6f70717273,"jklmnopqrs","jklmnopqrs"
10,0x6b,"k","k"

```

A bulk load configuration file with and `.xml` extension is also created when either a table or a result set is exported to a bulk load data file. See "The bulk load configuration file" for an example of a bulk load configuration file.

In addition, a log file of events as well as external overflow files can be created during a bulk export operation. The log file is configured through either the driver Setup dialog Bulk tab, the `ExportTableToFile` function, or the `SQL_BULK_EXPORT` statement attribute. The external overflow files are configured through connection options; see "External overflow files" for details.

See also

[The bulk load configuration file](#) on page 113

[External overflow files](#) on page 116

Bulk loading to a database

You can load data from the bulk load data file into the target database through the DataDirect driver Setup dialog by selecting the Bulk tab and clicking **Load Table**. See "Bulk tab" for a description of this procedure.

Your application can also load data from the bulk load data file into the target database using the using the DataDirect functions LoadTableFromFile (ANSI application) or LoadTableFromFileW (Unicode application). The application must first obtain driver connection handles and function pointers, as shown in the following example:

```

HDBC      hdbc;
HENV      henv;
void      *driverHandle;
HMODULE   hmod;
PLoadTableFromFile loadTableFromFile;
char      tableName[128];
char      fileName[512];
char      configFile[512];
char      logFile[512];
char      discardFile[512];
int       errorTolerance;
int       warningTolerance;
int       loadStart;
int       loadCount;
int       readBufferSize;

/* Get the driver's connection handle from the DM.
   This handle must be used when calling directly into the driver.*/

rc = SQLGetInfo (hdbc, SQL_DRIVER_HDBC, &driverHandle, 0, NULL);
if (rc != SQL_SUCCESS) {
    ODBC_error (henv, hdbc, SQL_NULL_HSTMT);
    EnvClose (henv, hdbc);
    exit (255);
}
/* Get the DM's shared library or DLL handle to the driver. */

rc = SQLGetInfo (hdbc, SQL_DRIVER_HLIB, &hmod, 0, NULL);
if (rc != SQL_SUCCESS) {
    ODBC_error (henv, hdbc, SQL_NULL_HSTMT);
    EnvClose (henv, hdbc);
    exit (255);
}

loadTableFromFile = (PLoadTableFromFile)
    resolveName (hmod, "LoadTableFromFile");
if (! loadTableFromFile) {
    printf ("Cannot find LoadTableFromFile!\n");
    exit (255);
}

rc = (*loadTableFromFile) (
    driverHandle,
    (const SQLCHAR *) tableName,
    (const SQLCHAR *) fileName,
    errorTolerance, warningTolerance,
    (const SQLCHAR *) configFile,
    (const SQLCHAR *) logFile,
    (const SQLCHAR *) discardFile,
    loadStart, loadCount,
    readBufferSize);
if (rc == SQL_SUCCESS) {
    printf ("Load succeeded.\n");
}
else {

```



```

    driverError (driverHandle, hmod);
}

```

Use the BulkLoadBatchSize connection attribute to specify the number of rows the driver loads to the data source at a time when bulk loading data. Performance can be improved by increasing the number of rows the driver loads at a time because fewer network round trips are required. Be aware that increasing the number of rows that are loaded also causes the driver to consume more memory on the client.

A log file of events as well as a discard file that contains rows rejected during the load can be created during a bulk load operation. These files are configured through either the driver Setup dialog Bulk tab or the LoadTableFromFile function.

The discard file is in the same format as the bulk load data file. After fixing reported issues in the discard file, the bulk load can be reissued using the discard file as the bulk load data file.

In addition to bulk Insert, the driver also supports bulk Delete, Update, and Upsert. This functionality is enabled with the SetBulkOperation function which is implemented in the driver.

Refer to "DataDirect Bulk Load" in the *Progress DataDirect for ODBC Drivers Reference* for supported functions and statement attributes.

See also

[Bulk tab](#) on page 68

The bulk load configuration file

A bulk load configuration file is created when either a table or a result set is exported to a bulk load data file. This file has the same name as the bulk load data file, but with an .xml extension.

The bulk load configuration file defines in its metadata the names and data types of the columns in the bulk load data file. The file defines these names and data types based on the table or result set created by the query that exported the data.

It also defines other data properties, such as length for character and binary data types, the character encoding code page for character types, precision and scale for numeric types, and nullability for all types.

When a bulk load data file cannot read its configuration file, the following defaults are assumed:

- All data is read in as character data. Each value between commas is read as character data.
- The default character set is defined, on Windows, by the current Windows code page. On UNIX/Linux, it is the IANAAppCodePage value, which defaults to 4.

For example, the format of the bulk load data file GBMAXTABLE.CSV (discussed in "Exporting data from a database") is defined by the bulk load configuration file, GBMAXTABLE.XML, as follows:

```

<?xml version="1.0" encoding="utf-8"?>
<table codepage="UTF-16LE" xsi:noNamespaceSchemaLocation=
"http://media.datadirect.com/download/docs/ns/bulk/BulkData.xsd" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <column datatype="DECIMAL" precision="38" scale="0" nullable=
      "false">INTEGERCOL</column>
    <column datatype="VARBINARY" length="10" nullable=
      "true">VARBINCOL</column>
    <column datatype="VARCHAR" length="10" sourcecodepage="Windows-1252"
      externalfilecodepage="Windows-1252" nullable="true">VCHARCOL</column>
    <column datatype="VARCHAR" length="10" sourcecodepage="Windows-1252"
      externalfilecodepage="Windows-1252" nullable="true">UNIVCHARCOL</column>
  </row>
</table>

```

See also

[Exporting data from a database](#) on page 110

Bulk load configuration file schema

The bulk load configuration file is supported by an underlying XML Schema defined at:

<http://media.datadirect.com/download/docs/ns/bulk/BulkData.xsd>

The bulk load configuration file must conform to the bulk load configuration XML schema. Each bulk export operation generates a bulk load configuration file in UTF-8 format. If the bulk load data file cannot be created or does not comply with the XML Schema described in the bulk load configuration file, an error is generated.

Verification of the bulk load configuration file

You can verify the metadata in the configuration file against the data structure of the target database table. This insures that the data in the bulk load data file is compatible with the target database table structure.

The verification does not check the actual data in the bulk load data file, so it is possible that the load can fail even though the verification succeeds. For example, if you were to update the bulk load data file manually such that it has values that exceed the maximum column length of a character column in the target table, the load would fail.

Not all of the error messages or warnings that are generated by verification necessarily mean that the load will fail. Many of the messages simply notify you about possible incompatibilities between the source and target tables. For example, if the bulk load data file has a column that is defined as an integer and the column in the target table is defined as smallint, the load may still succeed if the values in the source column are small enough that they fit in a smallint column.

To verify the metadata in the bulk load configuration file through the DataDirect driver Setup dialog, select the Bulk tab and click **Verify**. See the individual driver chapters of the drivers that support bulk load for a description of this procedure.

Your application can also verify the metadata of the bulk load configuration file using the DataDirect functions `ValidateTableFromFile` (ANSI application) or `ValidateTableFromFileW` (Unicode application). The application must first obtain driver connection handles and function pointers, as shown in the following example:

```

HDBC      hdbc;
HENV      henv;
void      *driverHandle;
HMODULE   hmod;
PValidateTableFromFile validateTableFromFile;
char      tableName[128];
char      configFile[512];
char      messageList[10240];
SQLLEN    numMessages;
/* Get the driver's connection handle from the DM.
   This handle must be used when calling directly into the driver. */
rc = SQLGetInfo (hdbc, SQL_DRIVER_HDBC, &driverHandle, 0, NULL);
if (rc != SQL_SUCCESS) {
    ODBC_error (henv, hdbc, SQL_NULL_HSTMT);
    EnvClose (henv, hdbc);
    exit (255);
}/* Get the DM's shared library or DLL handle to the driver. */
rc = SQLGetInfo (hdbc, SQL_DRIVER_HLIB, &hmod, 0, NULL);
if (rc != SQL_SUCCESS) {
    ODBC_error (henv, hdbc, SQL_NULL_HSTMT);
    EnvClose (henv, hdbc);
    exit (255);
}validateTableFromFile = (PValidateTableFromFile)
    resolveName (hmod, "ValidateTableFromFile");

```

```

if (!validateTableFromFile) {
    printf ("Cannot find ValidateTableFromFile!\n");
    exit (255);
}messageList[0] = 0;
numMessages = 0;
rc = (*validateTableFromFile) (
    driverHandle,
    (const SQLCHAR *) tableName,
    (const SQLCHAR *) configFile,
    (SQLCHAR *) messageList,
    sizeof (messageList),
    &numMessages);
printf ("%d message%s\n", numMessages,
        (numMessages == 0) ? "s" :
        ((numMessages == 1) ? " : " : "s : "),
        (numMessages > 0) ? messageList : "");
if (rc == SQL_SUCCESS) {
    printf ("Validate succeeded.\n");
}
else {
    driverError (driverHandle, hmod);
}
}

```

Sample applications

Progress DataDirect provides a sample application that demonstrates the bulk export, verification, and bulk load operations. This application is located in the `\samples\bulk` subdirectory of the product installation directory along with a text file named `bulk.txt`. Please consult `bulk.txt` for instructions on using the sample bulk load application.

A bulk streaming application is also provided in the `\samples\bulkstrm` subdirectory along with a text file named `bulkstrm.txt`. Please consult `bulkstrm.txt` for instructions on using the bulk streaming application.

Character set conversions

It is most performance-efficient to transfer data between databases that use the same character sets. At times, however, you might need to bulk load data between databases that use different character sets. You can do this by choosing a character set for the bulk load data file that will accommodate all data. If the source table contains character data that uses different character sets, then one of the Unicode character sets, UTF-8, UTF-16BE, or UTF-16LE must be specified for the bulk load data file. A Unicode character set should also be specified in the case of a target table uses a different character set than the source table to minimize conversion errors. If the source and target tables use the same character set, that set should be specified for the bulk load data file.

A character set is defined by a code page. The code page for the bulk load data file is defined in the configuration file and is specified through either the Code Page option of the Export Table driver Setup dialog or through the `IANAAppCodePage` parameter of the `ExportTableToFile` function.

For supported code page values, refer to "Code page values" in the *Progress DataDirect for ODBC Drivers Reference*.

Any character conversion errors are handled based on the value of the Report Codepage Conversion Errors connection option. See the individual driver chapters for a description of this option.

The configuration file may optionally define a second code page value for each character column (`externalfilecodepage`). If character data is stored in an external overflow file (see "External overflow files"), this second code page value is used for the external file.

See also

[External overflow files](#) on page 116

External overflow files

In addition to the bulk load data file, DataDirect Bulk Load can store bulk data in external overflow files. These overflow files are located in the same directory as the bulk load data file. Different files are used for binary data and character data. Whether or not to use external overflow files is a performance consideration. For example, binary data is stored as hexadecimal-encoded character strings in the main bulk load data file, which increases the size of the file per unit of data stored. External files do not store binary data as hex character strings, and, therefore, require less space. On the other hand, more overhead is required to access external files than to access a single bulk load data file, so each bulk load situation must be considered individually.

The value of the Bulk Binary Threshold connection option determines the threshold, in KB, over which binary data is stored in external files instead of in the bulk load data file. Likewise, the Bulk Character Threshold connection option determines the threshold for character data.

In the case of an external character data file, the character set of the file is governed by the bulk load configuration file. If the bulk load data file is Unicode and the maximum character size of the source data is 1, then the data is stored in its source code page. See "Character set conversions".

The file name of the external file contains the bulk load data file name, a six-digit number, and a ".lob" extension in the following format: *CSVfilename_nnnnnn.lob*. Increments start at 000001.lob.

See also

[Character set conversions](#) on page 115

Summary of DataDirect Bulk Load related options

The following table summarizes how DataDirect Bulk Load related connection options work with the driver. See "Connection option descriptions" for details about configuring the options.

Table 16: Summary: Bulk Load Connection Options

Option	Characteristic
Bulk Load Asynchronous (BulkLoadAsync)	<p>Determines whether the driver treats bulk load operations as synchronous or asynchronous.</p> <p>If set to 0 (Disabled), bulk load operations are synchronous. The driver does not return from the function that invoked an operation until the operation is complete or the BulkLoadTimeout period has expired. If the operation times out, the driver returns an error.</p> <p>If set to 1 (Enabled), bulk load operations are asynchronous. The driver returns from the function that invoked an operation after the operation is submitted to the server. The driver does not verify the completion status of the bulk load operation.</p> <p>Default: 0 (Disabled)</p>

Option	Characteristic
Bulk Load Batch Size (BulkLoadBatchSize)	The number of rows that the driver sends to the database at a time during bulk operations. Default: 1024
Bulk Load Concurrency Mode (BulkLoadConcurrencyMode)	Determines whether multiple batches associated with a bulk load operation are processed by Salesforce in parallel or one at a time. If set to 0 (Serial), multiple batches associated with a bulk load operation are processed one at a time. If set to 1 (Parallel), multiple batches associated with a bulk load operation are processed in parallel. The order in which the batches are processed can vary. Default: 1 - Parallel
Bulk Load Poll Interval (BulkLoadPollInterval)	Specifies the number of seconds the driver waits to request bulk operation status. This interval is used by the driver the first time it requests status and for all subsequent status requests. Default: 10
Bulk Load Threshold (BulkLoadThreshold)	Determines when the driver uses bulk load for insert, update, delete, or batch operations. If set to 0, the driver always uses bulk load to execute insert, update, delete, or batch operations. If set to x , the driver only uses bulk load if the number of rows to be updated by an insert, update, delete, or batch operation exceeds the threshold. If the operation times out, the driver returns an error. Default: 4000
Bulk Load Timeout (BulkLoadTimeout)	The time, in seconds, that the driver waits for a Salesforce bulk job to complete. A value of zero means there is no timeout. Default: 0
Field Delimiter (BulkLoadFieldDelimiter)	Specifies the character that the driver will use to delimit the field entries in a bulk load data file. Default: None
Record Delimiter (BulkLoadRecordDelimiter)	Specifies the character that the driver will use to delimit the record entries in a bulk load data file. Default: None

See also

[Connection option descriptions](#) on page 123

Using Bulk API with SQL statements

The driver uses Salesforce Bulk API for executing selects, inserts, deletes, and updates based on the values of the Enable Bulk Load and Enable Bulk Fetch connection options.

For information on other connection options that influence the behavior of Salesforce Bulk API, see "Summary of options for using Bulk API with SQL statements."

Using Bulk API with Inserts, Updates, and Deletes:

When the Enable Bulk Load option is enabled (`EnableBulkLoad=1`), the driver uses the Salesforce Bulk API for inserts, updates, and deletes based on the values of the Bulk Load Threshold connection option. If the number of affected rows exceeds the value of Bulk Load Threshold option, the driver uses the Salesforce Bulk API to execute the insert, update, or delete operation.

You can further configure bulk load behavior by specifying the number of rows the driver loads at a time using the Bulk Load Batch Size connection option. Performance can be improved by increasing the number of rows the driver loads at a time because fewer network round trips are required. Be aware that increasing the number of rows that are loaded also causes the driver to consume more memory on the client.

Using Bulk API with Selects:

When the Enable Bulk Fetch option is enabled (`EnableBulkFetch=1`), the driver uses the Salesforce Bulk API for selects based on the values of the Bulk Fetch Threshold connection option. If the number of rows expected in the result set exceeds the value of Bulk Fetch Threshold option, the driver uses the Salesforce Bulk API to execute the select operation.

Note:

If there is a TOP or LIMIT clause in the select query, the driver considers the TOP or LIMIT clause value as the expected number of rows in the result set and compares it with the value of the Bulk Fetch Threshold option to choose either Bulk API or REST API for executing the select query.

If the value of TOP or LIMIT clause is greater than that of the Bulk Fetch Threshold option, but the actual row count in the result set is less, the performance of the select query might be affected adversely.

See also

[Enable Bulk Load](#) on page 151

[Enable Bulk Fetch](#) on page 150

[Summary of options for using the Bulk API with SQL statements](#) on page 118

Summary of options for using the Bulk API with SQL statements

The following table describes connection options related to using the Bulk API when executing selects, inserts, updates, and deletes. See "Connection option descriptions" in each driver chapter for details about configuring the options.

Table 17: Summary: Connection Options for Using Bulk API with SQL Statements

Option	Characteristic
Bulk Fetch Threshold (BulkFetchThreshold)	<p>Specifies a number of rows that, if exceeded, signals the driver to use the Salesforce Bulk API for select operations. For this behavior to take effect, the Enable Bulk Fetch option must be set to 1 (enabled).</p> <p>If set to 0, the driver uses the Salesforce Bulk API for all select operations.</p> <p>If set to x, the driver uses the Salesforce Bulk API for select operations when the value of x is exceeded.</p> <p>Default: 30000 (rows)</p>
Bulk Load Batch Size (BulkLoadBatchSize)	<p>The number of rows that the driver sends to the database at a time during bulk operations.</p> <p>Default: 1024</p>
Bulk Load Concurrency Mode (BulkLoadConcurrencyMode)	<p>Determines whether multiple batches associated with a bulk load operation are processed by Salesforce in parallel or one at a time.</p> <p>If set to 0 (Serial), multiple batches associated with a bulk load operation are processed one at a time.</p> <p>If set to 1 (Parallel), multiple batches associated with a bulk load operation are processed in parallel. The order in which the batches are processed can vary.</p> <p>Default: 1 (Parallel)</p>
Bulk Load Job Size (BulkLoadJobSize)	<p>Determines the number of rows to load into a single job of a bulk operation when BulkLoadVersion is set to V2. Performance can be improved by increasing the number of rows the driver loads at a time because fewer network round trips are required. Increasing the number of rows also causes the driver to consume more memory on the client.</p> <p>Default: 150000</p>
Bulk Load Threshold (BulkLoadThreshold)	<p>Determines when the driver uses bulk load for insert, update, and deletes.</p> <p>If set to 0, the driver always uses bulk load to execute insert, update, and deletes.</p> <p>If set to x, the driver only uses bulk load if the Enable Bulk Load option is set to a value of 1 (enabled) and the number of rows to be updated by an insert, update, or delete exceeds the threshold. If the operation times out, the driver returns an error.</p> <p>Default: 4000</p>
Bulk Load Timeout (BulkLoadTimeout)	<p>The time, in seconds, that the driver waits for a Salesforce bulk job to complete.</p> <p>A value of zero means there is no timeout.</p> <p>Default: 0</p>

Option	Characteristic
Bulk Load Version (BulkLoadVersion)	<p>Specifies which version of Salesforce Bulk Load API to be used for performing bulk load operations. This is applicable only if <code>EnableBulkLoad</code> is set to <code>true</code>.</p> <p>If set to <code>v1</code>, the driver uses the Salesforce Bulk API V1 for all insert, update, and delete operations.</p> <p>If set to <code>v2</code>, the driver uses the Salesforce Bulk API V2 to execute the insert, update, and delete operations.</p> <p>Default: <code>v1</code></p>
Enable Bulk Fetch (EnableBulkFetch)	<p>Specifies whether the driver can use the Salesforce Bulk API for selects based on the value of the Bulk Fetch Threshold connection option.</p> <p>If set to <code>1</code> (Enabled), the driver can use the Salesforce Bulk API for selects based on the value of the Bulk Fetch Threshold connection option. If the number of rows expected in the result set exceeds the value of Bulk Fetch Threshold option, the driver uses the Salesforce Bulk API to execute the select operation.</p> <p>If set to <code>0</code> (Disabled), the driver does not use the Salesforce Bulk API, and the Bulk Load Threshold option is ignored.</p> <p>Default: <code>1</code> (Enabled)</p> <hr/> <p>Note:</p> <p>If there is a TOP or LIMIT clause in the select query, the driver considers the TOP or LIMIT clause value as the expected number of rows in the result set and compares it with the value of the Bulk Fetch Threshold option to choose either Bulk API or REST API for executing the select query.</p> <p>If the value of TOP or LIMIT clause is greater than that of the Bulk Fetch Threshold option, but the actual row count in the result set is less, the performance of the select query might be affected adversely.</p> <hr/>
Enable Bulk Load (EnableBulkLoad)	<p>Specifies whether the driver can use the Salesforce Bulk API for inserts, updates, and deletes based on the value of the Bulk Load Threshold connection option.</p> <p>If set to <code>1</code> (Enabled), the driver can use the Salesforce Bulk API for inserts, updates, and deletes based on the value of the Bulk Load Threshold connection option. If the number of affected rows exceeds the value of Bulk Load Threshold option, the driver uses the Salesforce Bulk API to execute the insert, update, or delete operation.</p> <p>If set to <code>0</code> (Disabled), the driver does not use the Salesforce Bulk API, and the Bulk Load Threshold option is ignored.</p> <p>Default: <code>1</code> (Enabled)</p>

Option	Characteristic
Enable Primary Key Chunking (EnablePKChunking)	<p>Specifies whether the driver uses PK chunking for select operations. PK chunking breaks down bulk fetch operations into smaller, more manageable batches for improved performance.</p> <p>If set to 1 (Enabled), the driver uses PK chunking for select operations when the expected number of rows in the result set is greater than the values of the Bulk Fetch Threshold and Primary Key Chunk Size options. For this behavior to take effect, the Enable Bulk Fetch option must also be set to 1 (enabled).</p> <p>If set to 0 (Disabled), the driver does not use PK chunking when executing select operations, and the Primary Key Chunk Size option is ignored.</p> <p>Default: 1 (Enabled)</p>
Primary Key Chunk Size (PKChunkSize)	<p>Specifies the size, in rows, of a primary key (PK) chunk when PK chunking has been enabled via the Enable Primary Key Chunking option. The Salesforce Bulk API splits the query into chunks of this size.</p> <p>Default: 100000 (rows)</p>

See also

[Connection option descriptions](#) on page 123

Connection option descriptions

The following connection option descriptions are listed alphabetically by the GUI name that appears on the driver Setup dialog box. The connection string attribute name, along with its short name, is listed immediately underneath the GUI name.

In most cases, the GUI name and the attribute name are the same; however, some exceptions exist. If you need to look up an option by its connection string attribute name, please refer to the alphabetical table of connection string attribute names.

Also, a few connection string attributes, for example, Password, do not have equivalent options that appear on the GUI. They are in the list of descriptions alphabetically by their attribute names.

Note: The driver does not support specifying values for the same connection option multiple times in a connection string or DSN. If a value is specified using the same attribute multiple times or using both long and short attributes, the connection may fail or the driver may not behave as intended.

The following table lists the connection string attributes supported by the Salesforce driver.

Table 18: Salesforce Attribute Names

Attribute (Short Name)	Default
AccessToken (AT)	None
ApplicationUsingThreads (AUT)	1 (Enabled)
ApplyToLabel (ATL)	0 (Disabled)
Authentication Method (AM)	USERIDPASSWORD

Attribute (Short Name)	Default
BulkFetchThreshold (BFT)	30000 (rows)
BulkLoadAsync (BLA)	0 (Disabled)
BulkLoadBatchSize (BLBS)	1024
BulkLoadConcurrencyMode (BLCM)	1 (Parallel)
BulkLoadJobSize (BLJS)	150000
BulkLoadFieldDelimiter (BLFD)	None
BulkLoadPollInterval (BLPI)	10
BulkLoadRecordDelimiter (BLRD)	None
BulkLoadThreshold (BLTH)	4000 (rows)
BulkLoadTimeout (BLTO)	0
BulkLoadVersion (BLV)	V1
ClientID (CI)	None
ClientSecret (CS)	None
ConfigOptions (CFGO)	Empty string
<hr/> Important: The configuration options have been deprecated. However, the driver will continue to support them until the next major release of the driver. <hr/>	
ConnectionReset (CR)	0 (Disabled)
CreateDB (CDB)	None
<hr/> Note: The CreateDB option has been replaced by CreateMap. <hr/>	
CreateMap (CM)	2 (NotExist)
DataSourceName (DSN)	None
Database	None
<hr/> Note: The Database option has been replaced by SchemaMap. <hr/>	

Attribute (Short Name)	Default
Description (n/a)	None
EnableBulkFetch (EBF)	1 (Enabled)
EnableBulkLoad (EBL)	1 (Enabled)
FetchSize (FS)	100
HostName (HOST)	login.salesforce.com
IANAAppCodePage UNIX ONLY	4 (ISO 8559-1 Latin-1)
InitializationString (IS)	Empty string
JVMArgs (JVMA)	For the 32-bit driver when the SQL Engine Mode is set to 2 (Direct): -Xmx256m For all other configurations: -Xmx1024m
JVMClasspath (JVMC)	<i>install_dir\java\lib\sforce.jar</i>
LoadBalanceTimeout (LBT)	0
LogConfigFile (LCF)	Empty string
LoginTimeout (LT)	15
LogonDomain (LD)	None
Note: Support for the LogonDomain connection option has been deprecated. The full user name, including the domain, is now specified using the User Name option.	
LogonID (UID)	None
MaxPoolSize (MXPS)	100
MinPoolSize (MNPS)	0
Password (PWD)	None
Pooling (POOL)	0 (Disabled)
ProxyHost (PXHN)	Empty string
ProxyPassword (PXPW)	Empty string

Attribute (Short Name)	Default
ProxyPort (PXPT)	Empty string
ProxyUser (PXUN)	Empty string
ReadOnly (RO)	0 (Disabled)
RefreshDirtyCache (RDC)	NA
Note: Support for the RefreshDirtyCache option has been deprecated.	
RefreshSchema (RS)	0 (Disabled)
RefreshToken (RT)	None
ReportCodepageConversionErrors (RCCE)	0 (Ignore Errors)
SchemaMap (SMP)	<p>For Windows:</p> <ul style="list-style-type: none"> User Data Source: <code>user_profile\AppData\Local\Progress\DataDirect\Salesforce_Schema\user_name.config</code> System Data Source: <code>C:\Users\Default\AppData\Local\Progress\DataDirect\Salesforce_Schema\user_name.config</code> <p>For UNIX/Linux: <code>~/progress/datadirect/salesforce_schema/LogonID.config</code></p>
SecurityToken (STK)	Empty string
ServerPortNumber (SPN)	<p>For the 32-bit driver: 19938</p> <p>For the 64-bit driver: 19937</p>
SQLEngineMode (SEM)	<p>For Windows: 0 (Auto)</p> <p>For UNIX/Linux: The SQL engine runs in Direct mode by default.</p>
StmtCallLimit (SCL)	100
StmtCallLimitBehavior (SCLB)	1 (ErrorAlways)

Attribute (Short Name)	Default
TransactionMode (TM)	0 (No Transactions)
WSFetchSize (WSFS)	0
WSPoolSize (WSPS)	1
WSRetryCount (WSRC)	0
WSTimeout (WST)	120

For details, see the following topics:

- [Access Token](#)
- [Application Using Threads](#)
- [Apply ToLabel](#)
- [Authentication Method](#)
- [Bulk Fetch Threshold](#)
- [Bulk Load Asynchronous](#)
- [Bulk Load Batch Size](#)
- [Bulk Load Concurrency Mode](#)
- [Bulk Load Job Size](#)
- [Bulk Load Poll Interval](#)
- [Bulk Load Threshold](#)
- [Bulk Load Timeout](#)
- [Bulk Load Version](#)
- [Client ID](#)
- [Client Secret](#)
- [Config Options](#)
- [Connection Pooling](#)
- [Connection Reset](#)
- [Create Database](#)
- [Create Map](#)
- [Data Source Name](#)
- [Database](#)
- [Description](#)
- [Enable Bulk Fetch](#)

- [Enable Bulk Load](#)
- [Enable Primary Key Chunking](#)
- [Fetch Size](#)
- [Field Delimiter](#)
- [Host Name](#)
- [IANAAppCodePage](#)
- [Initialization String](#)
- [JVM Arguments](#)
- [JVM Classpath](#)
- [LoadBalance Timeout](#)
- [Log Config File](#)
- [Login Timeout](#)
- [Logon Domain](#)
- [Max Pool Size](#)
- [Min Pool Size](#)
- [Password](#)
- [Primary Key Chunk Size](#)
- [Proxy Host](#)
- [Proxy Password](#)
- [Proxy Port](#)
- [Proxy User](#)
- [Read Only](#)
- [Record Delimiter](#)
- [Refresh Dirty Cache](#)
- [Refresh Schema](#)
- [Refresh Token](#)
- [Report Codepage Conversion Errors](#)
- [Schema Map](#)
- [Security Token](#)
- [Server Port Number](#)
- [SQL Engine Mode](#)
- [Statement Call Limit](#)
- [Statement Call Limit Behavior](#)

- [Transaction Mode](#)
- [User Name](#)
- [WSFetch Size](#)
- [WSPoolSize](#)
- [WSRetry Count](#)
- [WSTimeout](#)

Access Token

Attribute

AccessToken (AT)

Purpose

Specifies the access token required to authenticate to a Salesforce instance when OAuth 2.0 is enabled (AuthenticationMethod=oauth2.0).

The access token acts as a session ID for the connection. Refer to the Salesforce documentation to know how to obtain an access token.

Valid Values

string

where:

string

is the access token you have obtained from Salesforce.

Notes

- If a value for the Access Token option is not specified, the driver uses the value of the Refresh Token option to make a connection.
- If both Access Token and Refresh Token values are not specified, the driver cannot make a successful connection.
- If both Access Token and Refresh Token values are specified, the driver ignores the Access Token value and uses the Refresh Token value to generate a new Access Token value.

Default

None

GUI Tab

[Security tab](#)

See also

[Configuring OAuth 2.0 authentication](#) on page 95

[Authentication Method](#) on page 131

[Refresh Token](#) on page 170

Application Using Threads

Attribute

ApplicationUsingThreads (AUT)

Purpose

Determines whether the driver works with applications using multiple ODBC threads.

Valid Values

0 | 1

Behavior

If set to 1 (Enabled), the driver works with single-threaded and multi-threaded applications.

If set to 0 (Disabled), the driver does not work with multi-threaded applications. If using the driver with single-threaded applications, this value avoids additional processing required for ODBC thread-safety standards.

Notes

- This connection option can affect performance.

Default

1 (Enabled)

GUI Tab

[Advanced tab](#)

See also

[Performance considerations](#) on page 78

Apply ToLabel

Attribute

ApplyToLabel (ATL)

Purpose

Determines whether the driver applies the toLabel() function to the picklist fields when executing queries. The toLabel() function translates the result set of a query into the language of the user.

Valid Values

0 | 1

Behavior

If set to 1 (Enabled), the driver applies the toLabel() function to the picklist fields when executing queries.

If set to 0 (Disabled), the driver does not apply the toLabel() function to the picklist fields when executing queries.

Notes

- The driver does not apply the toLabel() function to the picklist fields specified using the Order By clause.

Default

0 (Disabled)

GUI Tab

[Advanced tab](#)

Authentication Method

Attribute

AuthenticationMethod (AM)

Purpose

Determines which authentication method the driver uses when establishing a connection.

Valid Values

userIDPassword | oauth2.0

Behavior

If set to userIDPassword, the driver uses user ID/password authentication when establishing a connection.

If set to oauth2.0, the driver uses OAuth 2.0 authentication when establishing a connection.

Default

userIDPassword

GUI Tab

[Security tab](#)

See Also

[Authentication](#) on page 95

Bulk Fetch Threshold

Attribute

BulkFetchThreshold (BFT)

Purpose

Specifies a number of rows that, if exceeded, signals the driver to use the Salesforce Bulk API for select operations. For this behavior to take effect, the Enable Bulk Fetch option must be set to 1 (enabled).

Valid Values

0 | x

where:

x

is a positive integer that represents a number of rows.

Behavior

If set to 0, the driver uses the Salesforce Bulk API for all select operations.

If set to x , the driver uses the Salesforce Bulk API for select operations when the value of x is exceeded.

Notes

- If the Enable Bulk Fetch option is set to 0 (disabled), the Bulk Fetch Threshold option is ignored.
- The Enable Primary Key Chunking connection option can be used to break bulk fetch operations into smaller, more manageable batches for improved performance.

Default

30000 (rows)

GUI Tab

[Bulk tab](#)

See also

- [Enable Bulk Load](#) on page 151
- [Enable Primary Key Chunking](#) on page 152
- [Statement Call Limit](#) on page 176

Bulk Load Asynchronous

Attribute

BulkLoadAsync (BLA)

Purpose

Determines whether the driver treats bulk load operations as synchronous or asynchronous.

Valid Values

0 | 1

Behavior

If set to 0 (Disabled), bulk load operations are synchronous. The driver does not return from the function that invoked an operation until the operation is complete or the BulkLoadTimeout period has expired. If the operation times out, the driver returns an error.

If set to 1 (Enabled), bulk load operations are asynchronous. The driver returns from the function that invoked an operation after the operation is submitted to the server. The driver does not verify the completion status of the bulk load operation.

Default

0 (Disabled)

GUI Tab

[Bulk tab](#)

See Also

- [Bulk Load Timeout](#) on page 137
- [DataDirect Bulk Load](#) on page 109

Bulk Load Batch Size

Attribute

BulkLoadBatchSize (BLBS)

Purpose

The number of rows that the driver sends to the database at a time during bulk operations. This value applies to all methods of bulk loading.

Valid Values

x

where:

x

is a positive integer that specifies the number of rows to be sent.

Default

1024

GUI Tab

[Bulk tab](#)

See Also

- [DataDirect Bulk Load](#) on page 109

Bulk Load Concurrency Mode

Attribute

BulkLoadConcurrencyMode (BLCM)

Purpose

Determines whether multiple batches associated with a bulk load operation are processed by Salesforce in parallel or one at a time.

Valid Values

0 | 1

Behavior

If set to 0 (Serial), multiple batches associated with a bulk load operation are processed one at a time.

If set to 1 (Parallel), multiple batches associated with a bulk load operation are processed in parallel. The order in which the batches are processed can vary.

Default

1 (Parallel)

GUI Tab

[Bulk tab](#)

See Also

- [DataDirect Bulk Load](#) on page 109

Bulk Load Job Size

Attribute

BulkLoadJobSize (bljs)

Purpose

Determines the number of rows to load into a single job of a bulk operation when BulkLoadVersion is set to V2. Performance can be improved by increasing the number of rows the driver loads at a time because fewer network round trips are required. Increasing the number of rows also causes the driver to consume more memory on the client.

Valid Values

x

where:

x

is a positive integer less than or equal to 1000000 that represents a number of rows.

Behavior

The driver loads the specified number of rows into a single job.

Default Value

150000

Bulk Load Poll Interval

Attribute

BulkLoadPollInterval (BLPI)

Purpose

Specifies the number of seconds the driver waits to request bulk operation status. This interval is used by the driver the first time it requests status and for all subsequent status requests.

Valid Values

x

where:

x

is a positive integer that represents the number of seconds the driver waits before requesting bulk operation status.

Default

10

GUI Tab

[Bulk tab](#)

See Also

- [DataDirect Bulk Load](#) on page 109

Bulk Load Threshold

Attribute

BulkLoadThreshold (BLTH)

Purpose

Determines when the driver uses bulk load for inserts, updates, or deletes. If the Enable Bulk Load option is set to 1 (enabled) and the number of rows affected by an insert, update, delete, or batch operation exceeds the threshold specified by this option, the driver uses the Salesforce Bulk API to perform the operation.

Valid Values

0 | x

where:

x

is a positive integer that represents a threshold (number of rows).

Behavior

If set to 0, the driver always uses bulk load to execute inserts, updates, or deletes.

If set to x , the driver only uses bulk load if the Enable Bulk Load option is set to a value of 1 (enabled) and the number of rows to be updated by an inserts, updates, or deletes exceeds the threshold. If the operation times out, the driver returns an error.

Notes

- If the Enable Bulk Load option is set to 0 (disabled), this option is ignored.
- Do not set the Bulk Load Threshold option to a value greater than the Web service call limit set by the Statement Call Limit option. If the value set for Bulk Load Threshold is greater than the value of Statement Call Limit, the driver would never use the Salesforce Bulk API because the Web service call limit is reached before the driver reaches the threshold to switch to the Salesforce Bulk API.

Default

4000

GUI Tab

[Bulk tab](#)

See Also

- [Enable Bulk Load](#) on page 151
- [Statement Call Limit](#) on page 176
- [DataDirect Bulk Load](#) on page 109

Bulk Load Timeout

Attribute

BulkLoadTimeout (BLTO)

Purpose

The time, in seconds, that the driver waits for a Salesforce bulk job to complete. A value of zero means there is no timeout.

Valid Values

x

where:

x

is a positive integer that represents a number of seconds the driver waits before requesting bulk operation status.

Default

0

GUI Tab

[Bulk tab](#)

See Also

- [DataDirect Bulk Load](#) on page 109

Bulk Load Version

Attribute

BulkLoadVersion (blv)

Purpose

Specifies which version of Salesforce Bulk Load API to be used for performing bulk load operations.

Valid Values

v1 | v2

Behavior

If set to v1, the driver uses the Salesforce Bulk API V1 for all insert, update, and delete operations.

If set to v2, the driver uses the Salesforce Bulk API V2 for all insert, update, and delete operations.

Notes

- This is applicable only if EnableBulkLoad is set to 1 (Enabled).

Default Value

v1

GUI Tab

[Bulk tab](#)

Client ID

Attribute

ClientID (CI)

Purpose

Specifies the consumer key for your application. The driver uses this value when authenticating to a Salesforce instance using OAuth 2.0 (`AuthenticationMethod=oauth2.0`).

Refer to the Salesforce documentation to know how to obtain the consumer key for your application.

Valid Values

string

where:

string

is the consumer key for your application.

Default

None

GUI Tab

[Security tab](#)

See also

[Configuring OAuth 2.0 authentication](#) on page 95

[Authentication Method](#) on page 131

Client Secret

Attribute

ClientSecret (CS)

Purpose

Specifies the consumer secret for your application. This value can optionally be specified when authenticating to a Salesforce instance using OAuth 2.0 (`AuthenticationMethod=oauth2.0`).

Refer to the Salesforce documentation to know how to obtain the consumer secret for your application.

Valid Values

string

where:

string

is the consumer secret for your application.

Default

None

GUI Tab

[Security tab](#)

See Also

[Configuring OAuth 2.0 authentication](#) on page 95

[Authentication Method](#) on page 131

Config Options

Important: The configuration options have been deprecated. However, the driver will continue to support them until the next major release of the driver.

Attribute

ConfigOptions (CFG0)

Purpose

Determines how the mapping of the remote data model to the relational data model is configured, customized, and updated.

Notes

- This option is primarily used for initial configuration of the driver for a particular user. It is not intended for use with every connection. By default, the driver configures itself and this option is normally not needed. If Config Options is specified on a connection after the initial configuration, the values specified for Config Options must match the values specified for the initial configuration.

Valid Values

```
{ key = value [ ; key = value ] }
```

where:

key

is one of the following values:

- AuditColumns
- CustomSuffix
- MapSystemColumnNames
- NumberFieldMapping
- UppercaseIdentifiers

The value is a set of key value pairs separated by a semicolon (;). The value must be enclosed in curly brackets. For example:

```
{AuditColumns=All;CustomSuffix=strip;KeywordConflictSuffix=;
MapSystemColumnNames=1;NumberFieldMapping=alwaysDouble;
UppercaseIdentifiers=false}
```

Default

```
AuditColumns=All;CustomSuffix=Include;KeywordConflictSuffix=;
MapSystemColumnNames=0;NumberFieldMapping=emulateInteger;
UppercaseIdentifiers=true;
```

GUI Tab

[Advanced tab](#)

AuditColumns (Config Option)

Attribute

AuditColumns

Purpose

Determines whether the driver includes audit fields, which Salesforce adds to all objects defined in a Salesforce instance, as table columns when it defines the remote data model to relational table mapping.

The audit columns added by Salesforce are:

- IsDeleted
- CreatedById
- CreatedDate
- LastModifiedById
- LastModifiedDate
- SystemModstamp

Valid Values

All | AuditOnly | MasterOnly | None

The value for this option is specified as a key=value pair in the Config Options field. See "Config Options" for details.

Behavior

If set to `All`, the driver includes the all of the audit columns and the master record id column in its table definitions.

If set to `AuditOnly`, the driver adds only the audit columns in its table definitions.

If set to `MasterOnly`, the driver adds only the `MasterRecordId` column in its table definitions.

If set to `None`, the driver does not add the audit columns or the `MasterRecordId` column in its table definitions.

Default

All

Notes

- In a typical Salesforce instance, not all users are granted access to the Audit or MasterRecordId columns. If AuditColumns is set to a value other than `None` and the driver cannot include the columns requested, the connection fails and the driver generates a `SQLException` with a `SQLState` of 08001.

GUI Tab

The value for config options are specified in the Config Options field on the [Advanced tab](#).

See also

[Config Options](#) on page 139

CustomSuffix (Config Option)

Attribute

CustomSuffix

Purpose

Determines whether the driver includes or strips the `_c` suffix from the table and column names when mapping the remote data model to the relational data model. Salesforce adds the suffix to all custom objects and fields.

Valid Values

Include | Strip

The value for this option is specified as a *key=value* pair in the Config Options field. See "ConfigOptions" for details.

Behavior

If set to `Include`, the driver includes the `_c` suffix.

If set to `Strip`, the driver strips the `_c` suffix.

Default

Include

GUI Tab

The value for config options are specified in the Config Options field on the [Advanced tab](#).

See also

[Config Options](#) on page 139

KeywordConflictSuffix (Config Option)

Purpose

Specifies a string of up to five alphanumeric characters that the driver appends to any object or field name that conflicts with a SQL engine keyword.

Valid Values

string

where:

string

is a string of up to five alphanumeric characters.

The value for this option is specified as a *key=value* pair in the Config Options field. See "Config Options" for details.

Example

A field called `CASE` exists in the native Salesforce data. To avoid a naming conflict with the SQL engine keyword `CASE`, you could set `KeywordConflictSuffix=TAB`. In this scenario, the driver maps the `Case` object to the `CASETAB` column.

Notes

- Do not use a string that matches the suffix of a custom table, for example, `CASEOFICE`. If you specify `KeywordConflictSuffix=OFICE`, a name collision occurs with the standard object `CASE` and the custom table `CASEOFICE`, or a table with a column called `CASEOFICE`. In this situation, the standard object `CASE` is returned. The custom object is ignored.

Default

Empty string

GUI Tab

The value for config options are specified in the Config Options field on the [Advanced tab](#).

See also

[Config Options](#) on page 139

MapSystemColumnNames (Config Option)

Attribute

MapSystemColumnNames

Purpose

Determines how the driver maps Salesforce system columns.

Valid Values

0 | 1

Behavior

If set to 0, the driver does not change the names of the Salesforce system columns.

If set to 1, the driver changes the names of the Salesforce system columns as described in the following table:

Table 19: Mapped Names for Salesforce Field Names When Using MapSystemColumnNames

Field Name	Mapped Name
Id	ROWID
Name	SYS_NAME
IsDeleted	SYS_ISDELETED
CreatedDate	SYS_CREATEDDATE
CreatedById	SYS_CREATEDBYID
LastModifiedDate	SYS_LASTMODIFIEDDATE
LastModifiedId	SYS_LASTMODIFIEDID
SystemModstamp	SYS_SYSTEMMODSTAMP
LastActivityDate	SYS_LASTACTIVITYDATE
OwnerId	SYS_OWNERID

The value for this config option is specified as a *key=value* pair in the Config Options field. See "Config Options" for details.

Default

0

GUI Tab

The value for config options are specified in the Config Options field on the [Advanced tab](#).

See also

[Config Options](#) on page 139

NumberFieldMapping (Config Option)

Attribute

NumberFieldMapping

Purpose

Determines how the driver maps fields defined as NUMBER in Salesforce. The Salesforce API uses DOUBLE values to transfer data to and from NUMBER fields, which can cause problems when the precision of the NUMBER field is greater than the precision of a DOUBLE value. Rounding can occur when converting large values to and from DOUBLE. The NumberFieldMapping option allows you to map the NUMBER fields to the required SQL types based on their precision.

Valid Values

`emulateInteger` | `alwaysDouble` | `alwaysDecimal`

Behavior

If set to `emulateInteger`, the driver maps NUMBER fields with a precision of 9 or less and a scale of 0 to the INTEGER SQL type and maps all other NUMBER fields to the DOUBLE SQL type.

If set to `alwaysDouble`, the driver maps all NUMBER fields to the DOUBLE SQL type regardless of their precision.

If set to `alwaysDecimal`, the driver maps all NUMBER fields to the DECIMAL SQL type. It can be used when the precision of the NUMBER field is greater than the precision of a DOUBLE value.

The value for this option is specified as a *key=value* pair in the Config Options field. See "Config Options" for details.

Default

`emulateInteger`

GUI Tab

The value for config options are specified in the Config Options field on the [Advanced tab](#).

See also

[Config Options](#) on page 139

UppercaseIdentifiers (Config Option)

Attribute

UppercaseIdentifiers

Purpose

Specifies whether the driver maps all identifier names to uppercase.

Valid Values

true | false

The value for this option is specified as a key=value pair in the Config Options field. See "Config Options" for details.

Behavior

If set to `true`, the driver maps identifiers to uppercase.

If set to `false`, the driver maps identifiers to the mixed case name of the object being mapped. If mixed case identifiers are used, SQL statements must enclose those identifiers in double quotes and the case of the identifier must exactly match the case of the identifier name.

Example

For example, if `UppercaseIdentifiers=false`, to query the Account table you specify:

```
SELECT "Phone", "Website" FROM "Account"
```

Notes

- Do not change the value of `UppercaseIdentifiers` unless the data source you are connecting to has objects with names that differ only by case.

Default

true

GUI Tab

The value for config options are specified in the Config Options field on the [Advanced tab](#).

See also

[Config Options](#) on page 139

Connection Pooling

Attribute

Pooling (POOL)

Purpose

Specifies whether to use the driver's connection pooling.

Valid Values

0 | 1

Behavior

If set to 1 (Enabled), the driver uses connection pooling.

If set to 0 (Disabled), the driver does not use connection pooling.

Notes

- The application must be thread-enabled to use connection pooling.
- This connection option can affect performance.

Default

0 (Disabled)

GUI Tab

[Pooling tab](#)

See also

[Performance considerations](#) on page 78

Connection Reset

Attribute

ConnectionReset (CR)

Purpose

Determines whether the state of connections that are removed from the connection pool for reuse by the application is reset to the initial configuration of the connection.

Valid Values

0 | 1

Behavior

If set to 1 (Enabled), the state of connections removed from the connection pool for reuse by an application is reset to the initial configuration of the connection. Resetting the state can negatively impact performance because additional commands must be sent over the network to the server to reset the state of the connection.

If set to 0 (Disabled), the state of connections is not reset.

Notes

- This connection option can affect performance.

Default

0 (Disabled)

GUI Tab

[Pooling tab](#)

See also

This connection option can affect performance. See [Performance considerations](#) on page 78 for details.

Create Database

Attribute

CreateDB (CDB)

Purpose

Note: The Create Database option has been replaced by Create Map. The CreateDB attribute will continue to be supported for this release, but will be deprecated in subsequent versions of the product.

Determines whether the driver creates a new embedded database when establishing the connection.

Valid Values

0 | 1 | 2

Behavior

If set to 0 (No), the driver uses the current schema map specified by Database. If one does not exist, the connection fails.

If set to 1 (ForceNew), the driver deletes the current schema map specified by Database and creates a new one at the same location.

Warning: This causes all views, data caches, and map customizations defined in the current database to be lost.

If set to 2 (NotExist), the driver uses the current schema map specified by Database. If one does not exist, the driver creates one.

Default

None

GUI Tab

None

Create Map

Attribute

CreateMap (CM)

Purpose

Determines whether the driver creates a new schema map when establishing the connection.

Valid Values

0 | 1 | 2

Behavior

If set to 0 (No), the driver uses the current schema map specified by the Schema Map option. If one does not exist, the connection fails.

If set to 1 (ForceNew), the driver deletes the current schema map specified by the Schema Map option and creates a new one at the same location.

Warning: This causes all views, data caches, and map customizations defined in the current schema map to be lost.

If set to 2 (NotExist), the driver uses the current schema map specified by the Schema Map option. If one does not exist, the driver creates one.

Default

2 (NotExist)

GUI Tab

[Advanced tab](#)

See also

- [Schema Map](#) on page 172

Data Source Name

Attribute

DataSourceName (DSN)

Purpose

Specifies the name of a data source in your Windows Registry or `odbc.ini` file.

Valid Values

string

where:

string

is the name of a data source.

Default

None

GUI Tab

[General tab](#)

Database

Attribute

Database (DBN)

Purpose

Note: The Database option has been replaced by Schema Map. The Database attribute will continue to be supported for this release, but will be deprecated in subsequent versions of the product.

Specifies the file name prefix the driver uses to create or locate the set of files that define the Object mapping and the embedded database used by the connection.

Valid Values

prefix | *path+prefix*

where:

prefix

is the file name prefix for the embedded database. For example, if Database is set to a value of `JohnQPublic`, the embedded database files that are created or loaded have the form `johnqpublic.xxx`.

path+prefix

is a relative or absolute path appended to the file name prefix. The path defines the directory the driver uses to store the newly created database files or locate the existing database files. For example, if Database is set to a value of `C:\data\db\johnqpublic`, the driver either creates or looks for the database `johnqpublic.xxx` in the directory `C:\data\db`. If you do not specify a path, the current working directory is used.

Notes

- The driver parses the User ID value and removes all non-alphanumeric characters. For example, if User ID is specified as `John.Q.Public`, the value used for Database is `JohnQPublic`.

Default

None

GUI Tab

None

See Also

- [Schema Map](#) on page 172

Description

Attribute

Description (n/a)

Purpose

Specifies an optional long description of a data source. This description is not used as a runtime connection attribute, but does appear in the `ODBC.INI` section of the Registry and in the `odbc.ini` file.

Valid Values

string

where:

string

is a description of a data source.

Default

None

GUI Tab

[General tab](#)

Enable Bulk Fetch

Attribute

EnableBulkFetch (EBF)

Purpose

Specifies whether the driver can use the Salesforce Bulk API for selects based on the value of the Bulk Fetch Threshold connection option. If the number of rows expected in the result set exceeds the value of Bulk Fetch Threshold option, the driver uses the Salesforce Bulk API to execute the select operation. Using the Salesforce Bulk API may significantly reduce the number of Web service calls used to execute a statement and, therefore, may improve performance.

Note:

If there is a TOP or LIMIT clause in the select query, the driver considers the TOP or LIMIT clause value as the expected number of rows in the result set and compares it with the value of the Bulk Fetch Threshold option to choose either Bulk API or REST API for executing the select query.

If the value of TOP or LIMIT clause is greater than that of the Bulk Fetch Threshold option, but the actual row count in the result set is less, the performance of the select query might be affected adversely.

Valid Values

1 | 0

Behavior

If set to 1, the driver can use the Salesforce Bulk API for selects based on the value of the Bulk Fetch Threshold connection option. If the number of rows expected in the result set exceeds the value of Bulk Fetch Threshold option, the driver uses the Salesforce Bulk API to execute the select operation.

If set to 0, the driver does not use the Salesforce Bulk API, and the Bulk Fetch Threshold option is ignored.

Default

1

GUI Tab

[Bulk tab](#)

See Also

- [Bulk Fetch Threshold](#) on page 132
- [Performance considerations](#) on page 78

Enable Bulk Load

Attribute

EnableBulkLoad (EBL)

Purpose

Specifies whether the driver can use the Salesforce Bulk API for inserts, updates, and deletes based on the value of the Bulk Load Threshold connection option. If the number of affected rows exceeds the value of Bulk Load Threshold option, the driver uses the Salesforce Bulk API to execute the insert, update, or delete operation. Using the Salesforce Bulk API may significantly reduce the number of Web service calls used to execute a statement and, therefore, may improve performance.

Valid Values

1 | 0

Behavior

If set to 1, the driver can use the Salesforce Bulk API for inserts, updates, and deletes based on the value of the Bulk Load Threshold connection option. If the number of affected rows exceeds the value of Bulk Load Threshold option, the driver uses the Salesforce Bulk API to execute the insert, update, or delete operation.

If set to 0, the driver does not use the Salesforce Bulk API, and the Bulk Load Threshold option is ignored.

Default

1

GUI Tab

[Bulk tab](#)

See Also

- [Bulk Load Threshold](#) on page 136
- [Performance considerations](#) on page 78

Enable Primary Key Chunking

Attribute

EnablePKChunking (EPKC)

Purpose

Specifies whether the driver uses PK chunking for select operations. PK chunking breaks down bulk fetch operations into smaller, more manageable batches for improved performance.

Valid Values

1 | 0

Behavior

If set to 1 (Enabled), the driver uses PK chunking for select operations when the expected number of rows in the result set is greater than the values of the Bulk Fetch Threshold and Primary Key Chunk Size options. For this behavior to take effect, the Enable Bulk Fetch option must also be set to 1 (enabled).

If set to 0 (Disabled), the driver does not use PK chunking when executing select operations, and the Primary Key Chunk Size option is ignored.

Notes

- PK chunking is supported for all custom objects and the following standard objects: Account, Campaign, CampaignMember, Case, Contact, Lead, LoginHistory, Opportunity, Task, and User. In addition, PK chunking is supported for sharing objects as long as the parent object is supported.

Default

1 (Enabled)

GUI Tab

[Bulk tab](#)

See Also

- [Bulk Fetch Threshold](#) on page 132
- [Primary Key Chunk Size](#) on page 164
- [Enable Bulk Load](#) on page 151
- [Performance considerations](#) on page 78

Fetch Size

Attribute

FetchSize (FS)

Purpose

Specifies the maximum number of rows that the driver processes before returning data to the application. Smaller fetch sizes can improve the initial response time of the query. Larger fetch sizes improve overall fetch times at the cost of additional memory.

FetchSize is related to, but different than, WSFetchSize. WSFetchSize specifies the number of rows of raw data that the driver fetches from the remote data source, while FetchSize specifies how many of these raw data rows the driver processes before returning data to the application. Processing the data includes converting from the remote data source data type to the driver SQL data type used by the application. If FetchSize is greater than WSFetchSize, the driver makes multiple round trips to the data source to get the requested number of rows before returning control to the application.

Valid Values

0 | x

where:

x

is a positive integer.

Behavior

If set to 0, the driver fetches and processes all of the rows of the result before returning control to the application.

If set to x , the driver fetches and processes the specified number of rows before returning data to the application.

Notes

- WSFetchSize and FetchSize can be used to adjust the trade-off between throughput and response time. Smaller fetch sizes can improve the initial response time of the query. Larger fetch sizes can improve overall response times at the cost of additional memory.

Default

100

GUI Tab

[Advanced tab](#)

See Also

- [WSFetch Size](#) on page 178

Field Delimiter

Attribute

BulkLoadFieldDelimiter (BLFD)

Purpose

Specifies the character that the driver will use to delimit the field entries in a bulk load data file.

Valid Values

x

where:

x

is any printable character.

For simplicity, avoid using a value that can be in the data, including all alphanumeric characters, the dash(-), the colon(:), the period (.), the forward slash (/), the space character, the single quote (') and the double quote ("). You can use some of these characters as delimiters if all of the data in the file is contained within double quotes.

Notes

- The Bulk Load Field Delimiter character must be different from the Bulk Load Record Delimiter.

Default

None

GUI Tab

[Bulk tab](#)

See Also

- [DataDirect Bulk Load](#) on page 109

Host Name

Attribute

HostName (HOST)

Purpose

The base Salesforce URL or IP address of your Salesforce instance. If you are logging into a Salesforce instance other than the default, you must provide the root of the Salesforce URL or IP address.

Valid Values

url | *ip_address*

where:

url

is the is the root of the Salesforce URL to which you want to connect.

ip_address

is the is the IP address of the Salesforce instance to which you want to connect.

Example

Suppose you have a Salesforce instance that is configured with a production instance and a sandbox instance. You can specify `login.salesforce.com` as the value for the HostName attribute to connect to the production instance or `test.salesforce.com` to connect to the sandbox instance:

Salesforce Instance	URL
Production	<code>login.salesforce.com</code>
Sandbox	<code>test.salesforce.com</code>

Default

`login.salesforce.com`

GUI Tab

[General tab](#)

IANAAppCodePage

Attribute

IANAAppCodePage (IACP)

Purpose

An Internet Assigned Numbers Authority (IANA) value. You must specify a value for this option if your application is not Unicode enabled or if your database character set is not Unicode.

The driver uses the specified IANA code page to convert "W" (wide) functions to ANSI.

The driver and Driver Manager both check for the value of IANAAppCodePage in the following order:

- In the connection string
- In the Data Source section of the system information file (`odbc.ini`)
- In the ODBC section of the system information file (`odbc.ini`)

If the driver does not find an IANAAppCodePage value, the driver uses the default value of 4 (ISO 8859-1 Latin-1).

Valid Values

IANA_code_page

where:

IANA_code_page

is one of the valid values listed in "IANAAppCodePage values" in the *Progress DataDirect for ODBC Drivers Reference*. The value must match the database character encoding and the system locale.

Default

4 (ISO 8559-1 Latin-1)

GUI Tab

NA

See Also

Refer to "Internationalization, localization, and Unicode" in the *Progress DataDirect for ODBC Drivers Reference* for details.

Initialization String

Attribute

InitializationString (IS)

Purpose

One or multiple SQL commands to be executed by the driver after it has established the connection to the database and has performed all initialization for the connection. If the execution of a SQL command fails, the connection attempt also fails and the driver returns an error indicating which SQL command or commands failed.

Valid Values

string

where:

string

is one or multiple SQL commands.

Multiple commands must be separated by semicolons. In addition, if this option is specified in a connection URL, the entire value must be enclosed in parentheses when multiple commands are specified.

Example

Because fetching metadata and generating mapping files can significantly increase the time it takes to connect to Salesforce, the driver caches this information on the client the first time the driver connects on behalf of each user. The cached metadata is used in subsequent connections made by the user instead of re-fetching the metadata from Salesforce. To force the driver to re-fetch the metadata information for a connection, use the `InitializationString` property to pass the REFRESH MAP command in the connection URL. For example:

```
DSN=Salesforce;UID={test@abccorp.com};PWD=secret;InitializationString=(REFRESH MAP)
```

Default

None

GUI Tab

[Advanced tab](#)

JVM Arguments

Attribute

JVMArgs (JVMA)

Purpose

A string that contains the arguments that are passed to the JVM that the driver is starting. The location of the JVM must be specified on the driver library path. For information on setting the location of the JVM in your environment, see:

- [Setting the library path environment variable \(Windows\)](#) on page 30
- [Setting the library path environment variable \(UNIX and Linux\)](#) on page 32.

When specifying the heap size for the JVM, note that the JVM tries to allocate the heap memory as a single contiguous range of addresses in the application's memory address space. If the application's address space is fragmented so that there is no contiguous range of addresses big enough for the amount of memory specified for the JVM, the driver fails to load, because the JVM cannot allocate its heap. This situation is typically encountered only with 32-bit applications, which have a much smaller application address space. If you encounter problems with loading the driver in an application, try reducing the amount of memory requested for the JVM heap. If possible, switch to a 64-bit version of the application.

Valid Values

string

where:

string

contains arguments that are defined by the JVM. Values that include special characters or spaces must be enclosed in curly braces { } when used in a connection string.

Example

To set the heap size used by the JVM to 256 MB and the http proxy information, specify:

```
{-Xmx256m -Dhttp.proxyHost=johndoe -Dhttp.proxyPort=808}
```

To set the heap size to 256 MB and configure the JVM for remote debugging, specify:

```
{-Xmx256m  
-Xrunjdp:transport=dt_socket, address=9003,server=y,suspend=n -Xdebug}
```

Default

For the 32-bit driver when the SQL Engine Mode connection option is set to 2 (Direct):

```
-Xmx256m
```

For all other configurations:

```
-Xmx1024m
```

GUI Tab

[SQL Engine tab](#)

JVM Classpath

Attribute

JVMClasspath (JVMC)

Purpose

Specifies the CLASSPATH for the Java Virtual Machine (JVM) used by the driver. The CLASSPATH is the search string the JVM uses to locate the Java jar files the driver needs.

Valid Values

string

where:

string

specifies the CLASSPATH. Separate multiple jar files by a semi-colon on Windows platforms and by a colon on Linux and UNIX platforms. CLASSPATH values with multiple jar files must be enclosed in curly braces { } when used in a connection string.

If your process employs multiple drivers that use a JVM, the value of the JVM Classpath for all affected drivers must include an absolute path to all the jar files for drivers used in that process. In addition, the value specified must be identical for all drivers. For example, if you are using the Salesforce and MongoDB drivers on Windows, you would specify a value of

`{c:\install_dir\java\lib\sforce.jar; c:\install_dir\java\lib\mongodb.jar}`
for both drivers. If the value for any of the affected drivers is missing a file path or is different from the one specified for the other drivers, the drivers will return an error at connection that the JVM is already running.

Example

On Windows:

```
{.:c:\install_dir\java\lib\sforce.jar}
```

On UNIX:

```
{./home/user1/install_dir/java/lib/sforce.jar}
```

Default

`install_dir\java\lib\sforce.jar`

GUI Tab

[SQL Engine tab](#)

LoadBalance Timeout

Attribute

LoadBalanceTimeout (LBT)

Purpose

Specifies the number of seconds to keep inactive connections open in a connection pool. An inactive connection is a database session that is not associated with an ODBC connection handle, that is, a connection in the pool that is not in use by an application.

Valid Values

0 | *x*

where:

x

is a positive integer that specifies a number of seconds.

Behavior

If set to 0, inactive connections are kept open.

If set to *x*, inactive connections are closed after the specified number of seconds passes.

Notes

- The Min Pool Size option may cause some connections to ignore this value.
- This connection option can affect performance.

Default

0

GUI Tab

[Pooling tab](#)

See also

[Performance considerations](#) on page 78

Log Config File

Attribute

LogConfigFile (LCF)

Purpose

Specifies the filename of the configuration file used to initialize the driver logging mechanism.

If the driver cannot locate the specified file when establishing the connection, the connection fails and the driver returns an error.

Valid Values

string

where:

string

is the relative or fully qualified path of the configuration file used to initialize the driver logging mechanism. If the specified file does not exist, the driver continues searching for an appropriate configuration file as described in "Logging for Java components".

Default

Empty string

GUI Tab

[Advanced tab](#)

See Also

- For more information, refer to "Logging for Java components" in the *Progress DataDirect for ODBC Drivers Reference*.

Login Timeout

Attribute

LoginTimeout (LT)

Purpose

The number of seconds the driver waits for a connection to be established before returning control to the application and generating a timeout error. To override the value that is set by this connection option for an individual connection, set a different value in the `SQL_ATTR_LOGIN_TIMEOUT` connection attribute using the `SQLSetConnectAttr()` function.

Valid Values

0 | x

where:

x

is a positive integer that specifies a number of seconds.

Behavior

If set to 0, the connection request does not time out, but the driver responds to the `SQL_ATTR_LOGIN_TIMEOUT` attribute.

If set to x, the connection request times out after the specified number of seconds unless the application overrides this setting with the `SQL_ATTR_LOGIN_TIMEOUT` attribute.

Default

15

GUI Tab

[Advanced tab](#)

Logon Domain

Attribute

LogonDomain (LD)

Purpose

Note: The Logon Domain connection option has been deprecated. As a result, the user name option has been enhanced to accept the complete Salesforce user id value, including the domain.

Specifies the domain part of the Salesforce user id. If Logon Domain is not an empty string, the driver first appends the @ character to the end of the User Name value and then appends the value of Logon Domain.

Valid Values

string

where:

string

is a valid user ID domain.

Default

None

GUI Tab

None

See also

- [User Name](#) on page 178

Max Pool Size

Attribute

MaxPoolSize (MXPS)

Purpose

The maximum number of connections allowed within a single connection pool. When the maximum number of connections is reached, no additional connections can be created in the connection pool.

Valid Values

An integer from 1 to 65535

For example, if set to 20, the maximum number of connections allowed in the pool is 20.

Notes

This connection option can affect performance.

Default

100

GUI Tab

[Pooling tab](#)

See also

[Performance considerations](#) on page 78

Min Pool Size

Attribute

MinPoolSize (MNPS)

Purpose

Specifies the minimum number of connections that are opened and placed in a connection pool, in addition to the active connection, when the pool is created. The connection pool retains this number of connections, even when some connections exceed their Load Balance Timeout value.

Valid Values

0 | x

where:

x

is an integer from 1 to 65535.

For example, if set to 5, the start-up number of connections in the pool is 5 in addition to the current existing connection.

Behavior

If set to 0, no connections are opened in addition to the current existing connection.

Notes

- This connection option can affect performance.

Default

0

GUI Tab

[Pooling tab](#)

See Also

See [Performance considerations](#) on page 78 for details.

Password

Attribute

Password (PWD)

Purpose

Specifies the password to use to connect to your Salesforce instance. A password is required. Contact your system administrator to obtain your password.

Important: Setting the password using a data source is not recommended. The data source persists all options, including the Password option, in clear text.

Valid Values

password | *password+securitytoken*

where:

password

is a valid password. The password is case-sensitive.

password+securitytoken

is a valid password appended by the security token required to connect to the Salesforce instance, for example, *secretXaBARTsLZReM4Px47qPLOS*, where *secret* is the password and the remainder of the value is the security token. Both the password and security token are case-sensitive.

Optionally, you can specify the security token in the Security Token option. Do not specify the security token in both options.

Default

None

GUI Tab

[Logon dialog](#)

See also

[Security Token](#) on page 173

Primary Key Chunk Size

Attribute

PKChunkSize (PKCS)

Purpose

Specifies the size, in rows, of a primary key (PK) chunk when PK chunking has been enabled via the Enable Primary Key Chunking option. The Salesforce Bulk API splits the query into chunks of this size.

Valid Values

x

where:

x

is a positive integer less than or equal to 250,000 rows. The driver requests, via the Salesforce Bulk API, that Salesforce divide the query into chunks of this size.

Notes

- Fewer rows may be returned if a `WHERE` clause has been applied to the fetch operation, or if soft-deleted records are included within the boundaries of a given chunk.
- Each chunk is processed as a separate batch that counts toward your daily batch limit. Larger chunks will result in fewer batches, but may reduce performance. Primary Key Chunk Size may be set to a maximum value of 250,000 rows.
- For PK chunking to be used in select operations, the expected number of rows in the result set must be greater than the values of the Bulk Fetch Threshold and Primary Key Chunk Size options.

Default

100000 (rows)

GUI Tab

[Bulk tab](#)

See Also

- [Enable Primary Key Chunking](#) on page 152
- [Bulk Fetch Threshold](#) on page 132

Proxy Host

Attribute

ProxyHost (PXHN)

Purpose

Specifies the Hostname of the Proxy Server. The value specified can be a host name, a fully qualified domain name, or an IPv4 or IPv6 address.

Valid Values

server_name | *IP_address*

where:

server_name

is the name of the server or a fully qualified domain name to which you want to connect.

The IP address can be specified in either IPv4 or IPv6 format, or a combination of the two.

Default

Empty string

GUI Tab

[SQL Engine tab](#)

Proxy Password

Attribute

ProxyPassword (PXPW)

Purpose

Specifies the password needed to connect to the Proxy Server.

Valid Values

String

where:

String

specifies the password to use to connect to the Proxy Server. Contact your system administrator to obtain your password.

Default

Empty string

GUI Tab

[SQL Engine tab](#)

Proxy Port

Attribute

ProxyPort (PXPT)

Purpose

Specifies the port number where the Proxy Server is listening for HTTP and/or HTTPS requests.

Valid Values

port_name

where:

port_name

is the port number of the server listener. Check with your system administrator for the correct number.

Default

None

GUI Tab

[SQL Engine tab](#)

Proxy User

Attribute

ProxyUser

Purpose

Specifies the user name needed to connect to the Proxy Server.

Valid Values

The default user ID that is used to connect to the Proxy Server.

Default

Empty string

GUI Tab

[SQL Engine tab](#)

Read Only

Attribute

ReadOnly (RO)

Purpose

Specifies whether the connection has read-only access to the data source.

Valid Values

0 | 1

Behavior

If set to 1 (Enabled), the connection has read-only access. The following commands are the only commands that you can use when a connection is read-only:

- Call (if the procedure does not update data)
- Explain Plan
- Select (except Select Into)

- Set Schema

The driver returns an error if any other command is executed.

If set to 0 (Disabled), the connection is opened for read/write access, and you can use all commands supported by the product.

Default

0 (Disabled)

GUI Tab

[Advanced tab](#)

Record Delimiter

Attribute

BulkLoadRecordDelimiter (BLRD)

Purpose

Specifies the character that the driver will use to delimit the record entries in a bulk load data file.

Valid Values

x

where:

x

is any printable character.

For simplicity, avoid using a value that can be in the data, including all alphanumeric characters, the dash(-), the colon(:), the period (.), the forward slash (/), the space character, the single quote (') and the double quote ("). You can use some of these characters as delimiters if all of the data in the file is contained within double quotes.

Notes

- The Bulk Load Record Delimiter character must be different from the Bulk Load Field Delimiter.

Default

None

GUI Tab

[Bulk tab](#)

See Also

- [DataDirect Bulk Load](#) on page 109

Refresh Dirty Cache

Attribute

RefreshDirtyCache (RDC)

Purpose

Note: The Refresh Dirty Cache option has been deprecated. Now, for every fetch operation, the driver refreshes the cached object to pick up changes made to tables and rows.

Specifies whether the driver refreshes a dirty cache on the next fetch operation from the cache. A cache is marked as dirty when a row is inserted into or deleted from a cached table or a row in the cached table is updated.

Valid Values

1 | 0

Behavior

If set to 1 (Enabled), a dirty cache is refreshed when the cache is referenced in a fetch operation. The cache state is set to initialized if the refresh succeeds.

If set to 0 (Disabled), a dirty cache is not refreshed when the cache is referenced in a fetch operation.

Default

NA

GUI Tab

[Advanced tab](#)

See also

[Refreshing cache data](#) on page 87

Refresh Schema

Attribute

RefreshSchema (RS)

Purpose

Determines whether the driver automatically refreshes the information in a remote schema (rebuilds the schema map for the schema) the first time a user connects to the specified embedded database. The database is opened when the user first makes a connection with the application. When all connections associated with that user are closed, then the driver closes the database. The database must be reopened before it can be used again.

Valid Values

1 | 0

Behavior

If set to 1 (Enabled), the driver automatically refreshes the schema the first time a user connects to the specified database. Any schema objects that have changed since the last time the schema map was rebuilt are reflected in the metadata.

If set to 0 (Disabled), the driver does not automatically refresh the schema the first time a user connects to the specified database.

Notes

- This connection option is functionally equivalent to executing the Refresh Map statement (see to "Refresh Map (EXT)"). You can refresh a schema manually at any time by using the Refresh Map statement.

Default

0 (Disabled)

GUI Tab

[Advanced tab](#)

See Also

- [Refresh Map \(EXT\)](#) on page 222

Refresh Token

Attribute

RefreshToken (RT)

Purpose

Specifies the refresh token used to either request a new access token or renew an expired access token. When the refresh token is specified, the access token generated at connection is used to authenticate to a Salesforce instance when OAuth 2.0 is enabled (`AuthenticationMethod=oauth2.0`).

Refer to the Salesforce documentation to know how to obtain a refresh token.

Valid Values

string

where:

string

is the refresh token you have obtained from Salesforce.

Notes

- If a value for the Access Token option is not specified, the driver uses the value of the Refresh Token option to make a connection.
- If both Access Token and Refresh Token values are not specified, the driver cannot make a successful connection.
- If both Access Token and Refresh Token values are specified, the driver ignores the Access Token value and uses the Refresh Token value to generate a new Access Token value.

Default

None

GUI Tab

[Security tab](#)

See also

[Configuring OAuth 2.0 authentication](#) on page 95

[Authentication Method](#) on page 131

[Access Token](#) on page 129

Report Codepage Conversion Errors

Attribute

ReportCodepageConversionErrors (RCCE)

Purpose

Specifies how the driver handles code page conversion errors that occur when a character cannot be converted from one character set to another.

An error message or warning can occur if an ODBC call causes a conversion error, or if an error occurs during code page conversions to and from the database or to and from the application. The error or warning generated is `Code page conversion error encountered`. In the case of parameter data conversion errors, the driver adds the following sentence: `Error in parameter x`, where `x` is the parameter number. The standard rules for returning specific row and column errors for bulk operations apply.

Valid Values

0 | 1 | 2

Behavior

If set to 0 (Ignore Errors), the driver substitutes 0x1A for each character that cannot be converted and does not return a warning or error.

If set to 1 (Return Error), the driver returns an error instead of substituting 0x1A for unconverted characters.

If set to 2 (Return Warning), the driver substitutes 0x1A for each character that cannot be converted and returns a warning.

Default

0 (Ignore Errors)

GUI Tab

[Advanced tab](#)

Schema Map

Attribute

SchemaMap (SMP)

Purpose

Specifies either the name or the absolute path and name of the configuration file where the relational map of native data is written. The driver looks for this file when connecting to a Salesforce instance. If the file does not exist, the driver creates one.

Valid Values

string

where:

string

is either the name or the absolute path and name (including the `.config` extension) of the configuration file. For example, if Schema Map is set to a value of:

- `ABC`, the driver either creates or looks for the configuration file `ABC` in the user's home directory, provided the `HOME` environment variable has been set.
- `C:\Users\Default\AppData\Local\Progress\DataDirect\Salesforce_Schema\jsmith@defcorp.config`, the driver either creates or looks for the configuration file `jsmith@defcorp.config` in the directory `C:\Users\Default\AppData\Local\Progress\DataDirect\Salesforce_Schema`.

Notes

- If using OAuth 2.0 authentication, a value for the Schema Map option must be specified for every connection.
- When connecting to a Salesforce instance, the driver looks for the schema map configuration file. If the configuration file does not exist, the driver creates a schema map using the name and location you have provided. If you do not provide a name and location, the driver creates one using default values.
- The driver uses the path specified in this connection property to store additional internal files.
- You can refresh the internal files related to an existing view of your data by using the SQL extension Refresh Map. Refresh Map runs a discovery against your native data and updates your internal files accordingly.

Default

The default value is determined by the User Name connection option, platform, and data source type. The following is a list of default values by platform:

- Windows platforms:
 - User Data Source:


```
user_profile\AppData\Local\Progress\DataDirect\Salesforce_Schema\user_name.config
```
 - System Data Source:


```
C:\Users\Default\AppData\Local\Progress\DataDirect\Salesforce_Schema\user_name.config
```
- UNIX/Linux:


```
~/progress/datadirect/salesforce_schema/LogonID.config
```

GUI Tab

[General tab](#)

See Also

- [Refresh Map \(EXT\)](#) on page 222
- [User Name](#) on page 178
- [Configuring OAuth 2.0 authentication](#) on page 95

Security Token

Attribute

SecurityToken (STK)

Purpose

Specifies the security token required to make a connection to a Salesforce instance that is configured for a security token. If a security token is required and you do not supply one, the driver returns an error indicating that an invalid user or password was supplied. Contact your Salesforce administrator to find out if a security token is required.

Valid Values

string

where:

string

is the value of the security token assigned to the user.

Optionally, you can specify the security token in the Password option by appending the security token to the password, for example, `secretXaBARTsLZReM4Px47qPLOS`. Do not specify the security token in both options.

Notes

- When setting the security token using a data source on Windows, the Security Token option is encrypted.

Default

Empty string

GUI Tab

[General tab](#)

Server Port Number

Attribute

ServerPortNumber (SPN)

Purpose

Specifies a valid port on which the SQL engine listens for requests from the driver.

Valid Values

port_name

where:

port_name

is the port number of the server listener. Check with your system administrator for the correct number.

Notes

- This option is ignored when SQL Engine Mode is set to 2 (Direct).

Default

For the 32-bit driver:

19938

For the 64-bit driver:

19937

GUI Tab

[SQL Engine tab](#)

SQL Engine Mode

Attribute

SQLEngineMode (SEM)

Purpose

Specifies whether the driver's SQL engine runs in the same 32-bit process as the driver (direct mode) or runs in a process that is separate from the driver (server mode). You must be an administrator to modify the server mode configuration values, and to start or stop the SQL engine service.

Valid Values

0 | 1 | 2

Behavior

If set to 0 (Auto), the SQL engine attempts to run in server mode first; however, if server mode is unavailable, it runs in direct mode. To use server mode with this value, you must start the SQL Engine service before using the driver (see "Starting the SQL engine server" for more information).

If set to 1 (Server), the SQL engine runs in server mode. The SQL engine operates in a separate process from the driver within its own JVM. You must start the SQL Engine service before using the driver (see "Starting the SQL engine server" for more information).

If set to 2 (Direct), the SQL engine runs in direct mode. The driver and its SQL engine run in a single process within the same JVM.

Important: Changes you make to the server mode configuration affect all DSNs sharing the service.

Default

For Windows:

0 (Auto)

For UNIX/Linux:

2 (Direct)

GUI Tab

[SQL Engine tab](#)

See Also

- [Starting the SQL engine server](#) on page 82

Statement Call Limit

Attribute

StmtCallLimit (SCL)

Purpose

Specifies the maximum number of Web service calls the driver can make when executing any single SQL statement or metadata query.

Valid Values

0 | x

where:

x

is a positive integer that defines the maximum number of Web service calls the driver can make when executing any single SQL statement or metadata query.

Behavior

If set to 0, there is no limit.

If set to x , the driver uses this value to set the maximum number of Web service calls on a single connection that can be made when executing a SQL statement. This limit can be overridden by changing the STMT_CALL_LIMIT session attribute using the ALTER SESSION statement. For example, the following statement sets the statement call limit to 10 Web service calls:

```
ALTER SESSION SET STMT_CALL_LIMIT=10
```

If the Web service call limit is exceeded, the behavior of the driver depends on the value specified for the Stmt Call Limit Behavior option.

Default

100 (Web service calls)

GUI Tab

[Web Service tab](#)

Statement Call Limit Behavior

Attribute

StmtCallLimitBehavior (SCLB)

Purpose

Specifies the behavior of the driver when the maximum Web service call limit specified by the Statement Call Limit option is exceeded.

Valid Values

1 | 2

Behavior

If set to 1 (ErrorAlways), the driver returns an error if the maximum Web service call limit is exceed.

If set to 2 (ReturnResults), the driver returns any partial results it received prior to the call limit being exceeded. The driver generates a warning that not all of the results were fetched.

Default

1 (ErrorAlways)

GUI Tab

[Web Service tab](#)

Transaction Mode

Attribute

TransactionMode (TM)

Purpose

Specifies how the driver handles manual transactions.

Valid Values

0 | 1

Behavior

If set to 1 (Ignore), the data source does not support transactions and the driver always operates in auto-commit mode. Calls to set the driver to manual commit mode and to commit transactions are ignored. Calls to rollback a transaction cause the driver to return an error indicating that no transaction is started. Metadata indicates that the driver supports transactions and the ReadUncommitted transaction isolation level.

If set to 0 (No Transactions), the data source and the driver do not support transactions. Metadata indicates that the driver does not support transactions.

Default

0 (No Transactions)

GUI Tab

[Advanced tab](#)

User Name

Attribute

LogonID (UID)

Purpose

The default user ID, including the domain, that is used to connect to your the Salesforce instance. Your ODBC application may override this value or you may override it in the logon dialog box or connection string.

Valid Values

userid

where:

userid

is a valid user ID with permissions to access the database.

Example

jsmith@defcorp.com

Default

None

GUI Tab

[General tab](#)

WSFetch Size

Attribute

WSFetchSize (WSFS)

Purpose

Specifies the number of rows of data the driver attempts to fetch for each ODBC call.

Valid Values

0 | *x*

where:

x

is a positive integer from 1 to 2000 that defines a number of rows.

Behavior

If set to 0, the driver attempts to fetch up to a maximum of 2000 rows. This value typically provides the maximum throughput.

If set to x , the driver attempts to fetch up to a maximum of the specified number of rows. Setting the value lower than 2000 can reduce the response time for returning the initial data. Consider using a smaller WSFetch Size for interactive applications only.

Notes

- WSFetchSize and FetchSize can be used to adjust the trade-off between throughput and response time. Smaller fetch sizes can improve the initial response time of the query. Larger fetch sizes can improve overall response times at the cost of additional memory.

Default

0 (up to a maximum of 2000 rows)

GUI Tab

[Web Service tab](#)

See also

[Fetch Size](#) on page 153

[WSTimeout](#) on page 181

WSPoolSize

Attribute

WSPoolSize (WSPS)

Purpose

Specifies the maximum number of Salesforce sessions the driver uses. This allows the driver to have multiple web service requests active when multiple ODBC connections are open, thereby improving throughput and performance.

Valid Values

x

where:

x

is the number of Salesforce sessions the driver uses to distribute calls. This value should not exceed the number of sessions permitted by your Salesforce account.

Notes

- You can improve performance by increasing the number of sessions specified by this option. By increasing the number of sessions the driver uses, you can improve throughput by distributing calls across multiple sessions when multiple connections are active.

- The maximum number of sessions is determined by the setting of WSPoolSize for the connection that initiates the session. For subsequent connections to an active session, the setting is ignored and a warning is returned. To change the maximum number of sessions, close all connections using the Salesforce ODBC driver; then, open a new ODBC Salesforce connection with desired limit specified for this option.

Default

1

GUI Tab

[Web Service tab](#)

See also

[Performance considerations](#) on page 78

WSRetry Count

Attribute

WSRetryCount (WSRC)

Purpose

The number of times the driver retries a timed-out Select request. Insert, Update, and Delete requests are never retried. The timeout period is specified by the WSTimeout (WST) connection option.

Valid Values

0 | x

where:

x

is a positive integer.

Behavior

If set to 0, the driver does not retry timed-out requests after the initial unsuccessful attempt.

If set to x , the driver retries the timed-out request the specified number of times.

Default

0

GUI Tab

[Web Service tab](#)

See also

[WSTimeout](#) on page 181

WSTimeout

Attribute

WSTimeout (WST)

Purpose

Specifies the time, in seconds, that the driver waits for a response to a Web service request.

Valid Values

0 | x

where:

x

is a positive integer that defines the number of seconds the driver waits for a response to a Web service request.

Behavior

If set to 0, the driver waits indefinitely for a response; there is no timeout.

If set to x, the driver uses the value as the default timeout for any statement created by the connection.

If a Select request times out and WSRetryCount (WSRC) is set to retry timed-out requests, the driver retries the request the specified number of times.

Default

120 (seconds)

GUI Tab

[Web Service tab](#)

See also

[WSRetry Count](#) on page 180

6

Supported SQL statements and extensions

The Salesforce driver provides support for the SQL statements and the SQL extensions described in this chapter. SQL extensions are denoted by an (EXT) in the topic title.

For details, see the following topics:

- [Alter Cache \(EXT\)](#)
- [Alter Index](#)
- [Alter Sequence](#)
- [Alter Session \(EXT\)](#)
- [Alter Table](#)
- [Create Cache \(EXT\)](#)
- [Create Index](#)
- [Create Sequence](#)
- [Create Table](#)
- [Create View](#)
- [Delete](#)
- [Drop Cache \(EXT\)](#)
- [Drop Index](#)
- [Drop Sequence](#)

- [Drop Table](#)
- [Drop View](#)
- [Explain Plan](#)
- [Insert](#)
- [Refresh Cache \(EXT\)](#)
- [Refresh Map \(EXT\)](#)
- [Select](#)
- [Update](#)
- [SQL expressions](#)
- [Subqueries](#)

Alter Cache (EXT)

Purpose

The Alter Cache statement changes the definition of a cache on a remote table or view. An error is returned if the remote table or view specified does not exist.

Syntax

```
ALTER CACHE ON {remote_table | view}
  [REFERENCING (remote_table_ref[,remote_table_ref]...)]
  [REFRESH_INTERVAL {0 | -1 | interval_value [{M, H, D}]}]
  [INITIAL_CHECK [ONFIRSTCONNECT | FIRSTUSE | DEFAULT]]
  [PERSIST {TEMPORARY | MEMORY | DISK | DEFAULT}]
  [ENABLED {YES | TRUE | NO | FALSE}]
  [CALL_LIMIT {0 | -1 | max_calls}]
  [FILTER (expression)]
```

where:

remote_table

is the name of the remote table cache definition to be modified. The remote table name can be a two-part name: *schemaname.tablename*. When specifying a two-part name, the specified remote table must be defined in the specified schema, and you must have the privilege to alter objects in the specified schema. When altering a relational cache, *remote_table* must specify the primary table of the relational cache.

view

is the name of the view cache definition to be modified. The view name can be a two-part name: *schemaname.viewname*. When specifying a two-part name, the specified view must be defined in the specified schema, and you must have the privilege to alter objects in the specified schema. Caches on views are not currently supported in the product.

REFERENCING

is an optional clause that specifies the name of the remote table(s) for which a relationship cache is to be created. See "Relational caches" and "Referencing clause" for a complete explanation.

REFRESH_INTERVAL

is an optional clause that specifies the length of time the data in the cached table can be used before being refreshed. See "Refresh Interval clause" for a complete explanation.

INITIAL_CHECK

is an optional clause that specifies when the driver initially checks whether the data in the cache needs refreshed. See "Initial Check clause" for a complete explanation.

PERSIST

is an optional clause that specifies the life span of the data in the cached table or view. See "Persist clause" for a complete explanation.

ENABLED

is an optional clause that specifies whether the cache is enabled or disabled for use with SQL statements. See "Enabled clause" for a complete explanation.

CALL_LIMIT

is an optional clause that specifies the maximum number of Web service calls that can be used to populate or refresh the cache. See "Call Limit clause" for a complete explanation.

FILTER

is an optional clause that specifies a filter for the primary table to limit the number of rows that are cached in the primary table. See "Filter clause" for a complete explanation.

Notes

- At least one of the optional clauses must be used. If two or more are specified, they must be specified in the order shown in the grammar description.

See also

[Relational caches](#) on page 186
[Referencing clause](#) on page 196
[Refresh Interval clause](#) on page 197
[Initial Check clause](#) on page 197
[Persist clause](#) on page 198
[Enabled clause](#) on page 198
[Call Limit clause](#) on page 199
[Filter clause](#) on page 200

Relational caches

If the Referencing clause is specified, the Alter Cache statement drops the existing cache and any referenced caches and creates a new set of related caches, one for each of the tables specified in the statement. The cache attributes for the existing cache are the default cache attributes for the new relational cache. Any attributes specified in the Alter Cache statement override the default attributes. If the Referencing clause is not specified, the existing cache references, if any, are used.

If the cache being altered is a relational cache, the attributes specified in the Alter Cache statement apply to all of the caches that comprise the relational cache.

Alter Index

Purpose

The Alter Index statement changes the name of an existing index. Index names must not conflict with other user-defined or system-defined names.

Syntax

```
ALTER INDEX index_name RENAME TO new_name
```

where:

index_name

specifies an existing index name.

new_name

specifies the new index name.

Alter Sequence

Purpose

The Alter Sequence statement resets the next value of an existing sequence.

Syntax

```
ALTER SEQUENCE sequence_name RESTART WITH value
```

where:

sequence_name

specifies an existing sequence.

value

specifies the next value to be returned through the Next Value For clause (see "Next Value for clause").

Notes

- Indexes on remote tables cannot be created, altered or dropped. Indexes can only be defined on local tables by the driver.

See also

[Next Value For clause](#) on page 202

Alter Session (EXT)

Purpose

The Alter Session statement changes various attributes of a database session or a remote session. A database session maintains the state of the overall connection. A remote session maintains the state that pertains to a particular remote data source connection.

Syntax

```
ALTER SESSION SET attribute_name=value
```

where:

attribute_name

specifies the name of the attribute to be changed. Attributes apply to either database sessions or remote sessions.

value

refers to the specific value setting for that attribute.

The following table lists the database and remote session attributes, and provides their descriptions.

Table 20: Alter Session Attributes

Attribute Name	Session Type	Description
Current_Schema	Database	Sets the current schema for the database session. The current schema is the schema used when an identifier in a SQL statement is unqualified. The string value must be the name of a schema visible in the database session. For example: <pre>ALTER SESSION SET CURRENT_SCHEMA=<i>sforce</i></pre>

Attribute Name	Session Type	Description
Stmt_Call_Limit	Database	<p>Sets the maximum number of Web service calls the driver can make in executing a statement. Setting the Stmt_Call_Limit attribute has the same effect as setting the StmtCallLimit connection option. It sets the default Web service call limit used by any statement on the connection. Executing this command on a statement overrides the previously set StmtCallLimit for the connection. The value specified must be a positive integer or 0. The value 0 means that no call limit exists. For example:</p> <pre>ALTER SESSION SET STMT_CALL_LIMIT=10</pre>
Ws_Call_Count	Remote	<p>Resets the Web service call count of a remote session to the value specified. The value must be zero or a positive integer. WS_Call_Count represents the total number of Web service calls made to the remote data source instance for the current session. For example:</p> <pre>ALTER SESSION SET sforce.WS_CALL_COUNT=0</pre> <p>The current value of WS_Call_Count can be obtained by referring to the System_Remote_Sessions system table (see "SYSTEM_REMOTE_SESSIONS catalog table" for details). For example:</p> <pre>SELECT * FROM information_schema.system_remote_sessions WHERE session_id = cursessionid()</pre>

See also

[SYSTEM_REMOTE_SESSIONS catalog table](#) on page 91

Alter Table

Purpose

The Alter Table statement adds or removes a column. The table being altered can be either a remote or local table. A remote table is a Salesforce object and is exposed in the SFORCE schema. A local table is maintained by the driver and is local to the machine on which the driver is running. A local table is exposed in the PUBLIC schema.

- For information on altering a remote table, see "Altering a remote table."
- For information on altering a local table, see "Altering a local table."

Altering a remote table

Syntax

```
ALTER TABLE table_name[add_clause] [drop_clause]
```

where:

table_name

specifies an existing remote table.

add_clause

specifies a column or a foreign key constraint to be added to the table. See "Add clause: columns" and "Add clause: constraints" for a complete explanation.

drop_clause

specifies a column to be dropped from the table. See "Add clause columns" for a complete explanation.

Notes

- You cannot drop a constraint from a remote table.

See also

[Add clause: columns](#) on page 189

[Add clause: constraints](#) on page 190

[Drop clause: columns](#) on page 190

Add clause: columns

Purpose

Use the Add clause to add a column to an existing table. It is optional.

This clause adds a column to the table. It defines a column with the same syntax as the Create Table command (see "Column definition for remote tables").

Syntax

```
ADD [COLUMN] column_nameDatatype ... [DEFAULT default_value] [[NOT]NULL] [EXT_ID]
[PRIMARY KEY] [START WITH starting_value]
```

default_value

is the default value to be assigned to the column. See "Column definition for remote tables" for details.

starting_value

is the starting value for the Identity column. The default start value is 0.

Notes

- If NOT NULL is specified and the table is not empty, a default value must be specified. In all other respects, this command is the equivalent of a column definition in a Create Table statement.
- You cannot specify ANYTYPE, BINARY, COMBOBOX, or TIME data types in the column definition of Alter Table statements.

- If a SQL view includes `SELECT * FROM` for the table to which the column was added in the view's Select statement, the new column is added to the view.

Example A

Assuming the current schema is SFORCE, this example adds the `status` column with a default value of `ACTIVE` to the `test` table.

```
ALTER TABLE test ADD COLUMN status TEXT(30) DEFAULT 'ACTIVE'
```

Example B

Assuming the current schema is SFORCE, this example adds a `deptId` column that can be used as a foreign key column.

```
ALTER TABLE test ADD COLUMN deptId TEXT(18)
```

See also

[Column definition for remote tables](#) on page 203

Add clause: constraints

Purpose

Use the Add clause to add a constraint to an existing table. It is optional.

This command adds a constraint using the same syntax as the Create Table command (see "Create Table").

Syntax

```
ADD [CONSTRAINT constraint_name] ...
```

Notes

- The only type of constraint you can add is a foreign key constraint.
- When adding a foreign key constraint, the table that contains the foreign key must be empty.

Example A

Assuming the current schema is SFORCE, a foreign key constraint is added to the `deptId` column of the `test` table, referencing the `rowId` of the `dept` table. For the operation to succeed, the `dept` table must be empty.

```
ALTER TABLE test ADD FOREIGN KEY (deptId) REFERENCES dept(rowId)
```

See also

[Create Table](#) on page 202

Drop clause: columns

Purpose

Use the Drop clause to drop a column from an existing table. It is optional.

Syntax

```
DROP {[COLUMN] column_name | [CONSTRAINT] constraint_name}
```

where:

column_name

specifies an existing column in an existing table.

Notes

- The column being dropped cannot have a constraint defined on it.
- Drop fails if a SQL view includes the column.

Example A

This example drops the *status* column. For the operation to succeed, the status column cannot have a constraint defined on it and cannot be used in a SQL view.

```
ALTER TABLE test DROP COLUMN status
```

Altering a local table

Syntax

```
ALTER TABLE table_name [add_clause] [drop_clause] [rename_clause]
```

where:

table_name

specifies an existing local table.

add_clause

specifies a column or constraint to be added to the table. See "Add clause: columns" and "Add clause: constraints" for a complete explanation.

drop_clause

specifies a column or constraint to be dropped from the table. See "Drop clause: columns" and "Drop clause: constraints" for a complete explanation.

rename_clause

specifies a new name for the table. See "Rename clause" for a complete explanation.

See also

[Add clause: columns](#) on page 192

[Add clause: constraints](#) on page 192

[Drop clause: columns](#) on page 193

[Drop clause: constraints](#) on page 193

[Rename clause](#) on page 194

Add clause: columns

Purpose

Use the Add clause to add a column to an existing table. It is optional.

This clause adds a column to the end of the column list. It defines a column with the same syntax as the Create Table command (see "Creating a local table"). If `NOT NULL` is specified and the table is not empty, a default value must be specified. In all other respects, this command is the equivalent of a column definition in a Create Table statement.

Syntax

```
ADD [COLUMN] column_nameDatatype ... [BEFORE existing_column]
```

Notes

- You cannot specify ANYTYPE, BINARY, COMBOBOX, or TIME data types in the column definition of Alter Table statements.
- The optional *Before existing_column* can be used to specify the name of an existing column so that the new column is inserted in a position just before the existing column.
- The optional *Before existing_column* can be used to specify the name of an existing column so that the new column is inserted in a position just before the existing column.
- If a SQL view includes `SELECT * FROM` for the table to which the column was added in the view's Select statement, the new column is added to the view.

Example A

Assuming the current schema is PUBLIC, this example adds the `status` column with a default value of `ACTIVE` to the `test` table.

```
ALTER TABLE test ADD COLUMN status VARCHAR(30) DEFAULT 'ACTIVE'
```

Example B

Assuming the current schema is PUBLIC, this example adds a `deptId` column that can be used as a foreign key column.

```
ALTER TABLE test ADD COLUMN deptId VARCHAR(18)
```

See also

[Creating a local table](#) on page 207

Add clause: constraints

Purpose

Use the Add clause to add a constraint to an existing table. It is optional.

This command adds a constraint using the same syntax as the Create Table command (see "Constraint definition for local tables").

Syntax

```
ADD [CONSTRAINT constraint_name] ...
```


Notes

- You cannot add a Unique constraint if one is already assigned to the same column list. A Unique constraint works only if the values of the columns in the constraint columns list for the existing rows are unique or include a Null value.
- Adding a foreign key constraint to the table fails if, for each existing row in the referring table, a matching row (with equal values for the column list) is not found in the referenced table.

Example A

Assuming the current schema is PUBLIC, this example adds a foreign key constraint to the `deptId` column of the `test` table that references the `rowId` of the `dept` table.

```
ALTER TABLE test ADD CONSTRAINT test_fk FOREIGN KEY (deptId) REFERENCES dept(id)
```

See also

[Constraint definition for local tables](#) on page 210

Drop clause: columns

Purpose

Use the Drop clause to drop a column from an existing table. It is optional.

Syntax

```
DROP {[COLUMN] column_name}
```

where:

column_name

specifies an existing column in an existing table.

Notes

- Drop fails if a SQL view includes the column.

Example A

This example drops the `status` column. For the operation to succeed, the `status` column cannot have a constraint defined on it and cannot be used in a SQL view.

```
ALTER TABLE test DROP COLUMN status
```

Drop clause: constraints

Purpose

Use the Drop clause to drop a constraint from an existing table. It is optional.

Syntax

```
DROP {[CONSTRAINT] constraint_name}
```

where:

constraint_name

specifies an existing constraint.

Notes

- The specified constraint cannot be a primary key constraint or unique constraint.

Example A

This example drops the `test_fk` constraint.

```
ALTER TABLE test DROP CONSTRAINT test_fk
```

Rename clause

Purpose

Use the Rename clause to rename an existing table. It is optional.

Syntax

```
RENAME TO new_name
```

where:

new_name

specifies the new name for the table.

Example A

This example renames the table to `test2`.

```
ALTER TABLE test RENAME TO test2
```

Create Cache (EXT)

Purpose

The Create Cache statement creates a cache that holds the data of a remote table. The data is not loaded into the cache when the Create Cache statement is executed; the data is loaded the first time that the remote table is executed or when a Refresh Cache statement on the remote table is executed. An error is returned if the remote table specified does not exist.

Syntax

```
CREATE CACHE ON {remote_table}  
  [REFERENCING (remote_table_ref[,remote_table_ref]...)]  
  [REFRESH_INTERVAL {0 | -1 | interval_value [{M, H, D}]}]  
  [INITIAL_CHECK [{ONFIRSTCONNECT | FIRSTUSE | DEFAULT}]  
  [PERSIST {TEMPORARY | MEMORY | DISK | DEFAULT}]  
  [ENABLED {YES | TRUE | NO | FALSE}]  
  [CALL_LIMIT {0 | -1 | max_calls}]  
  [FILTER (expression)]
```

where:

remote_table

is the name of the remote table from which data is to be cached on the client. The name of the cached table is the same as the name of the remote table. When the table name is specified in a query, the cached table is accessed, not the remote table.

The remote table name can be a two-part name: *schemaname . tablename*. When specifying a two-part name, the specified remote table must be defined in the specified schema, and you must have the privilege to create objects in the specified schema.

REFERENCING

is an optional clause that specifies the name of the remote table(s) for which a relationship cache is to be created. See "Relational caches" and "Referencing clause" for a complete explanation.

REFRESH_INTERVAL

is an optional clause that specifies the length of time the data in the cached table can be used before being refreshed. See "Refresh Interval clause" for a complete explanation.

INITIAL_CHECK

is an optional clause that specifies when the driver initially checks whether the data in the cache needs refreshed. See "Initial Check clause" for a complete explanation.

PERSIST

is an optional clause that specifies the life span of the data in the cached table or view. See "Persist clause" for a complete explanation.

ENABLED

is an optional clause that specifies whether the cache is enabled or disabled for use with SQL statements. See "Enabled clause" for a complete explanation.

CALL_LIMIT

is an optional clause that specifies the maximum number of Web service calls that can be used to populate or refresh the cache. See "Call Limit clause" for a complete explanation.

FILTER

is an optional clause that specifies a filter for the primary table to limit the number of rows that are cached in the primary table. See "Filter clause" for a complete explanation.

Notes

- Caches on views are not supported.
- If two or more optional clauses are specified, they must be specified in the order shown in the grammar description.

See also

[Relational caches](#) on page 196

[Referencing clause](#) on page 196

[Refresh Interval clause](#) on page 197

[Initial Check clause](#) on page 197

[Persist clause](#) on page 198

[Enabled clause](#) on page 198

[Call Limit clause](#) on page 199

[Filter clause](#) on page 200

Relational caches

If the Referencing clause is specified, the Create Cache statement creates a set of related caches, one for each of the tables specified in the statement. This set of caches is referred to as a related or relational cache. The set of caches in a relational cache is treated as a single entity. They are refreshed, altered, and dropped as a unit. Any attributes specified in the Create Cache statement apply to the cache created for the primary table and to the caches created for all of the referenced tables specified.

A database session can have both standalone and relational caches defined, but only one cache can be defined on a table. If a table is referenced in a relational cache definition, a standalone cache cannot be created on that table.

Referencing clause

Purpose

The Referencing clause specifies the name of the remote table(s) for which a relationship cache is to be created; it is optional. The specified remote table must be related to either the primary table being cached or one of the other specified related tables. The remote table name cannot include a schema name. The referenced tables must exist in the same schema as the primary table.

Syntax

```
REFERENCING (remote_table_ref[,remote_table_ref]...)
```

where:

remote_table_ref

represents *remote_table*[.*foreign_key_name*]

remote_table

specifies one or more tables related to the primary table that are to be cached in conjunction with the primary table.

foreign_key_name

specifies the name of the foreign key relationship between the remote table and the primary table (or, optionally, another related table). If a foreign key name is not specified, the driver attempts to find a relationship between the remote table and one of the other tables specified in the relational cache. The driver first looks for a relationship to the primary table. If a relationship to the primary table does not exist, the driver then looks for a relationship to other referenced tables.

Refresh Interval clause

Purpose

The Refresh Interval clause specifies the length of time the data in the cached table can be used before being refreshed; it is optional. The driver maintains a timestamp of when the data in a table was last refreshed. When a cached table is used in a query, the driver checks if the current time is greater than the last refresh time plus the value of Refresh_Interval. If it is, the driver refreshes the data in the cached table before processing the query.

Syntax

```
[REFRESH_INTERVAL {0 | -1 | interval_value [{M, H, D}]}]
```

where:

0

specifies that the cache is refreshed manually. You can use the Refresh Cache statement to refresh the cache manually.

-1

resets the refresh interval to the default value of 12 hours.

interval_value

is a positive integer that specifies the amount of time between refreshes. The default unit of time is hours (H). You can also specify M for minutes or D for days. For example, 60M would set the time between refreshes to 60 minutes. The default refresh interval is 12 hours.

Initial Check clause

Purpose

The Initial Check clause specifies when the driver performs its initial check of the data in the cache to determine whether it needs to be refreshed; it is optional.

Syntax

```
[INITIAL_CHECK [ONFIRSTCONNECT | FIRSTUSE | DEFAULT]]
```

where:

ONFIRSTCONNECT

specifies that the initial check is performed the first time a connection for a user is established. Subsequently, it is performed each time the table or view is used. A driver session begins on the first connection for a user and the session is active as long as at least one connection is open for the user.

FIRSTUSE

specifies that the initial check is performed the first time the table or view is used in a query. Subsequently, it is performed each time the table or view is used.

DEFAULT

resets the value back to its default, which is `FIRSTUSE`.

Persist clause

Purpose

The Persist clause specifies the life span of the data in the cached table or view; it is optional.

Syntax

```
[ PERSIST { TEMPORARY | MEMORY | DISK | DEFAULT } ]
```

where:

TEMPORARY

specifies that the data exists for the life of the driver session. When the driver session ends, the data is discarded. A driver session begins on the first connection for a user and the session is active if at least one connection is open for the user.

MEMORY

specifies that the data exists beyond the life of the connection. While the connection is active, the cached data is stored in memory. When the connection is closed, the cached data is persisted to disk. If the connection ends abnormally, changes to the cached data may not be persisted to disk. This is the default.

DISK

specifies that the data exists beyond the life of the connection. A portion of the cached data is stored in memory while the connection is active. If the size of the cached data exceeds the cache memory threshold, the remaining data is stored on disk. When the connection is closed, the portion of the cached data that is in memory is persisted to disk. If the connection ends abnormally, changes to the cached data held in memory may not be persisted to disk.

DEFAULT

resets the `PERSIST` value back to its default, which is `MEMORY`.

Notes

- If you specify a value of `MEMORY` or `DISK` for the Persist clause, the remote data remains on the client past the lifetime of the application.

Enabled clause

Purpose

The Enabled clause specifies whether the cache is enabled or disabled for use with SQL statements; it is optional.

Syntax

```
[ ENABLED { YES | TRUE | NO | FALSE } ]
```

where:

```
YES | TRUE
```

specifies that the cache is enabled. When a cache is enabled, the driver accesses the cached data for the remote table or view when a query is executed.

The driver does not check whether the cache needs to be refreshed when the Alter Cache statement is used to enable the cache. The check occurs the next time that the cache is accessed.

```
NO | FALSE
```

specifies that the cache is disabled, which means that the driver accesses the data in the remote table or view rather than the cache when a query is executed. The driver does not update the cache when inserts, updates, and deletes are performed on a remote table or view. To use the cache, you must enable it.

All data in an existing cache is persisted on the client even when the cache is disabled, except for the case where `PERSIST` is set to `TEMPORARY`.

The default is `TRUE`.

Call Limit clause

Purpose

The Call Limit clause specifies the maximum number of Web service calls that can be used to populate or refresh the cache; it is optional.

Syntax

```
[ CALL_LIMIT { 0 | -1 | max_calls } ]
```

where:

```
0
```

specifies no call limit.

```
-1
```

resets the call limit back to its default, which is 0 (no call limit).

```
max_calls
```

is a positive integer that specifies the maximum number of Web service calls.

The default value is 0.

Notes

- The call limit for a cache is independent of the `Stmt_Call_Limit` set on a database session. See "Alter Session (EXT)" for details.

If the call limit of a cache is exceeded during the population or refresh of the cache, the cache is marked as partially initialized. At the next refresh opportunity, the driver attempts to complete the population or refresh of the cache. If the call limit (or other error) occurs during this second attempt, the cache becomes invalid and is disabled. All data in the cache is discarded after the second attempt to populate or refresh the cache fails. Before re-enabling the cache, consider altering the cache definition to allow more Web service calls or specify a more restrictive filter, or both.

See also

[Alter Session \(EXT\)](#) on page 187

Filter clause

Purpose

Filter is an optional clause that specifies a filter for the primary table to limit the number of rows that are cached in the primary table. This clause is not supported for views.

Syntax

```
[FILTER (expression)]
```

where:

expression

is any valid Where clause. See "Where clause" for details. Do not include the Where keyword in the clause. The filter for an existing cache can be removed by specifying an empty string for the filter expression, for example, `FILTER()`.

The default value is that cached data is not filtered.

Example

The following example filters by last activity date.

```
FILTER (lastactivitydate >= {d'2010-01-01'})
```

Example A

The Referencing clause allows multiple related tables to be cached as a single entity. The following example creates a cache on the remote table `account`. The cache is populated with all accounts that had activity in 2010. Additionally, caches are created for the following remote tables: `opportunity`, `contact`, and `opportunitylineitem`. These caches are populated with the opportunities and contacts that are associated with the accounts stored in the `accounts` cache and the opportunity line items associated with the opportunities stored in the `opportunity` cache.

```
CREATE CACHE ON account REFERENCING (opportunity, contact, opportunitylineitem)  
FILTER (lastactivitydate >= {d'2010-01-01'})
```

Example B

The following example caches all rows of the `account` table with a refresh interval of 12 hours, checks whether data of the cached table needs to be refreshed on the first use, persists the data beyond the life of the connection, and stores the data in memory while the connection is active.

```
CREATE CACHE ON account
```


Example C

The following example caches all active accounts in the account table with a refresh interval of 1 day, checks whether data of the cached table needs to be refreshed when the connection is established, and discards the data when the connection is closed.

```
CREATE CACHE ON account REFRESH_INTERVAL 1d INITIAL_CHECK ONFIRSTCONNECT PERSIST
TEMPORARY FILTER(account.active = 'Yes')
```

See also

[Where clause](#) on page 228

Create Index

Purpose

The Create Index statement creates an index on one or more columns in a local table.

Syntax

```
CREATE [UNIQUE] INDEX index_name ON table_name (column_name [, ...])
```

where:

`UNIQUE`

means that key columns cannot have duplicate values.

index_name

specifies the name of the index to be created.

table_name

specifies an existing local table.

column_name

specifies an existing column.

Notes

- The driver cannot create an index in a remote table; the driver returns an error indicating that the operation cannot be performed on a remote table.
- Creating a unique constraint is the preferred way to specify that the values of a column must be unique.

Create Sequence

Purpose

The Create Sequence statement creates an auto-incrementing sequence for a local table.

Syntax

```
CREATE SEQUENCE sequence_name [AS {INTEGER|BIGINT}][START WITH start_value][INCREMENT BY increment_value]
```

where:

sequence_name

specifies the name of the sequence. By default, the sequence type is INTEGER.

start_value

specifies the starting value of the sequence. The default start value is 0.

increment_value

specifies the value of the increment; the value must be a positive integer. The default increment is 1.

Next Value For clause

Purpose

Use the Next Value For clause to specify the next value for a sequence that is used in a Select, Insert, or Update statement.

Syntax

```
NEXT VALUE FOR sequence_name
```

where:

sequence_name

specifies the name of the sequence from which to retrieve the value.

Example

The following example retrieves the next value or set of values in Sequence1:

```
SELECT NEXT VALUE FOR Sequence1 FROM Account
```

Create Table

- For information on creating remote tables, see "Altering a remote table."
- For information on creating local tables, see "Creating a local table."

Creating a remote table

Purpose

Use the Create Table statement to create a new table. You can create either a remote or local table. A remote table is a Salesforce object and is exposed in the SFORCE schema. Creating a table in the SFORCE schema creates a remote table. A local table is maintained by the driver and is local to the machine on which the driver is running. A local table is exposed in the PUBLIC schema. Creating a table in the PUBLIC schema creates a local table.

Syntax

```
CREATE TABLE table_name (column_definition [, ...] [, constraint_definition...])
```

where:

table_name

specifies the name of the new remote table. The table name can be qualified by a schema name using the format *schema.table*. If the schema is not specified, the table is created in the current schema. See "Alter Session (EXT)" for information about changing the current schema.

column_definition

specifies the definition of a column in the new table. See "Column definition for remote tables" for a complete explanation.

constraint_definition

specifies constraints on the columns of the new table. See "Constraint definition for remote tables" for a complete explanation.

Notes

- Creating tables in Salesforce is not a quick operation. It can take several minutes for Salesforce to create the table and its relationships.

See also

[Alter Session \(EXT\)](#) on page 187

[Column definition for remote tables](#) on page 203

[Constraint definition for remote tables](#) on page 205

Column definition for remote tables

Purpose

Defines the syntax to define a column for remote tables.

Syntax

```
column_name Datatype [(precision[,scale])...] [DEFAULT  
default_value][[NOT]NULL][EXT_ID][PRIMARY KEY] [START WITH starting_value]
```

where:

column_name

is the name to be assigned to the column.

Datatype

is the data type of the column to be created. See "Data types" for a list of supported Salesforce data types. You cannot specify ANYTYPE, BINARY, COMBOBOX, ENCRYPTEDTEXT, or TIME data types in the column definition of Create Table statements.

precision

is the total number of digits for DECIMAL columns, the number of seconds for DATETIME columns, and the length of HTML, LONGTEXTAREA, and TEXT columns.

scale

is the number of digits to the right of the decimal point for DECIMAL columns.

default_value

is the default value to be assigned to the column. The following default values are allowed in column definitions for remote tables:

- For character columns, a single-quoted string or NULL.
- For datetime columns, a single-quoted Date, Time, or Timestamp value or NULL. You can also use the following datetime SQL functions: CURRENT_DATE, CURRENT_TIMESTAMP, TODAY, or NOW.
- For boolean columns, the literals FALSE, TRUE, NULL.
- For numeric columns, any valid number or NULL.

starting_value

is the starting value for the Identity column. The default start value is 0.

[NOT]NULL

is used to specify whether NULL values are allowed or not allowed in a column. If NOT NULL is specified, all rows in the table must have a column value. If NULL is specified or if neither NULL or NOT NULL is specified, NULL values are allowed in the column.

EXT_ID

is used to specify that the column is an external ID column.

PRIMARY KEY

can only be specified when the data type of the column is ID. ID columns are always the primary key column for Salesforce.

START WITH

specifies the sequence of numbers generated for the Identity column. It can only be used when the data type of the column definition is AUTONUMBER.

Example A

Assuming the current schema is SFORCE, the remote table `Test` is created in the SFORCE schema. The `id` column has a starting value of 1000.

```
CREATE TABLE Test (id AUTONUMBER START WITH 1000, Name TEXT(30))
```

Example B

The table name is qualified with a schema name that is not the current schema, creating the `Test` table in the SFORCE schema. The table is created with the following columns: `id`, `Name`, and `Status`. The `Status` column contains a default value of `ACTIVE`.

```
CREATE TABLE SFORCE.Test (id NUMBER(9, 0), Name TEXT(30), Status TEXT(10) DEFAULT 'ACTIVE')
```

Example C

Assuming the current schema is SFORCE, the remote table `dept` is created with the `name` and `deptId` columns. The `deptId` column can be used as an external ID column.

```
CREATE TABLE dept (name TEXT(30), deptId NUMBER(9, 0) EXT_ID)
```

See also

[Data types](#) on page 26

Constraint definition for remote tables

Purpose

Defines the syntax to define a constraint for a remote table.

Syntax

```
[CONSTRAINT [constraint_name]  
  {foreign_key_constraint}]
```

where:

constraint_name

Is ignored. The driver uses the Salesforce relationship naming convention to generate the constraint name.

foreign_key_constraint

Defines a link between related tables. See "Foreign Key clause" for syntax.

A column defined as a foreign key in one table references a primary key in the related table. Only values that are valid in the primary key are valid in the foreign key. The following example is valid because the foreign key values of the `dept id` column in the `EMP` table match those of the `id` column in the referenced table `DEPT`:

Referenced Table	Main Table
DEPT	EMP
	(Foreign Key)

Referenced Table		Main Table		
id	name	id	name	dept id
1	Dev	1	Mark	1
2	Finance	1	Jim	3
3	Sales	1	Mike	2

The following example, however, is not valid. The value 4 in the dept id column does not match any value in the referenced id column of the DEPT table.

Referenced Table		Main Table		
DEPT		EMP		
id	name	id	name	dept id
1	Dev	1	Mark	1
2	Finance	1	Jim	3
3	Sales	1	Mike	4

(Foreign Key)

See also

[Foreign Key clause](#) on page 206

Foreign Key clause

Purpose

Defines the syntax to specify a foreign key for a constraint.

Syntax

```
FOREIGN KEY (fcolumn_name) REFERENCES ref_table (pcolumn_name)
```

where:

fcolumn_name

specifies the foreign key column to which the constraint is applied. The data type of this column must be the same as the data type of the column it references.

ref_table

specifies the table to which the foreign key refers.

pcolumn_name

specifies the primary key column in the referenced table. For Salesforce, the primary key column is always the `rowId` column.

Example

Assuming the current schema is SFORCE, the remote table `emp` is created with the `name`, `empId`, and `deptId` columns. The table contains a foreign key constraint on the `deptId` column, referencing the `rowId` in the `dept` table created in "Column definition for remote tables". For the operation to succeed, the data type of the `deptId` column must be the same as that of the `rowId` column.

```
CREATE TABLE emp (name TEXT(30), empId NUMBER(9, 0) EXT_ID, deptId TEXT(18),
FOREIGN KEY(deptId) REFERENCES dept(rowId))
```

See also

[Column definition for remote tables](#) on page 203

Creating a local table

Syntax

```
CREATE [{MEMORY | DISK | [GLOBAL] {TEMPORARY | TEMP}}]
TABLE table_name (column_definition [, ...]
[, constraint_definition...]) [ON COMMIT {DELETE | PRESERVE} ROWS
```

where:

MEMORY

creates the new table in memory. The data for a memory table is held entirely in memory for the duration of the database session. When the database is closed, the data for the memory table is persisted to disk.

DISK

creates the new table on disk. A disk table caches a portion of its data in memory and the remaining data on disk.

TEMPORARY and TEMP

are equivalent and create the new table as a global temporary table. The GLOBAL qualifier is optional. The definition of a global temporary table is visible to all connections. The data written to a global temporary table is visible only to the connection used to write the data.

Note: If MEMORY, DISK, or TEMPORARY/TEMP is not specified, the new table is created in memory.

table_name

specifies the name of the new table.

column_definition

specifies the definition of a column in the new table. See "Column definition for local tables" for a complete explanation.

constraint_definition

specifies constraints on the columns of the new table. See "Constraint definition for local tables" for a complete explanation.

ON COMMIT PRESERVE ROWS

preserves row values in a temporary table while the connection is open; this is the default action.

ON COMMIT DELETE ROWS

empties row values on each commit or rollback.

See also

[Column definition for local tables](#) on page 208

[Constraint definition for local tables](#) on page 210

Column definition for local tables

Purpose

Use the following syntax to define a column for local tables.

Syntax

```
column_name Datatype [(precision[,scale])]
[ {DEFAULT default_value | GENERATED BY DEFAULT AS IDENTITY
  (START WITH n[, INCREMENT BY m)]} ] | [[NOT] NULL]
[IDENTITY] [PRIMARY KEY]
```

where:

column_name

is the name to be assigned to the column.

Datatype

is the data type of the column to be created. See "Data types" for a list of supported Salesforce data types. You cannot specify ANYTYPE, BINARY, COMBOBOX, or TIME data types in the column definition of Create Table statements.

precision

is the number characters for CHAR and VARCHAR columns, the number of bytes for BINARY and VARBINARY columns, and the total number of digits for DECIMAL columns.

scale

is the number of digits to the right of the decimal point for DECIMAL columns and the number of seconds for DATETIME columns.

default_value

is the default value to be assigned to the column. The following default values are allowed in column definitions for local tables:

- For character columns, a single-quoted string or NULL. The only SQL function that can be used is CURRENT_USER.
- For datetime columns, a single-quoted Date, Time, or Timestamp value or NULL. You can also use the following datetime SQL functions: CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, TODAY, or NOW.
- For boolean columns, the literals FALSE, TRUE, NULL.
- For numeric columns, any valid number or NULL.
- For binary columns, any valid hexadecimal string or NULL.

IDENTITY | GENERATED BY DEFAULT AS IDENTITY

defines an auto-increment column. Either clause can be specified only on INTEGER or BIGINT columns. Identity columns are considered primary key columns, so a table can have only one Identity column.

The GENERATED BY DEFAULT AS IDENTITY clause is the standard SQL syntax for specifying an Identity column.

The IDENTITY operator is equivalent to GENERATED BY DEFAULT AS IDENTITY without the optional START WITH clause.

START WITH *n*[, INCREMENT BY *m*]

specifies the sequence of numbers generated for the Identity column. *n* and *m* are the starting and incrementing values, respectively, for an Identity column. The default start value is 0 and the default increment value is 1.

Example A

Assuming the current schema is PUBLIC, a local table is created. *id* is an identity column with a starting value of 0 and an increment value of 1 because no Start With and Increment By clauses are specified.

```
CREATE TABLE Test (id INTEGER GENERATED BY DEFAULT AS IDENTITY, name VARCHAR(30))
```

This example is equivalent to the previous example.

```
CREATE TABLE Test (id INTEGER IDENTITY, name VARCHAR(30))
```

Example B

Assuming the current schema is PUBLIC, a local table is created. *id* is an identity column with a starting value of 2 and an increment of 2.

```
CREATE TABLE Test (id INTEGER GENERATED BY DEFAULT AS IDENTITY (START WITH 2,
INCREMENT BY 2), name VARCHAR(30))
```

See also

[Data types](#) on page 26

Constraint definition for local tables

Purpose

Defines the syntax to define a constraint for a local table.

Syntax

```
[CONSTRAINT [constraint_name]
{unique_constraint |
primary_key_constraint |
foreign_key_constraint}]
```

where:

constraint_name

specifies a name for the constraint.

unique_constraint

specifies a constraint on a single column in the table. See [Unique Clause](#) for syntax.

Values in the constrained column cannot be repeated, except in the case of null values. For example:

```
ColA
1
2
NULL
4
5
NULL
```

A single table can have multiple columns with unique constraints.

primary_key_constraint

specifies a constraint on one or more columns in the table. See [Primary Key Clause](#) for syntax.

Values in a single column primary key column must be unique. Values across multiple constrained columns cannot be repeated, but values within a column can be repeated. Null values are not allowed. For example:

```
Col A  Col B
2      1
```

3	1
4	2
5	2
6	2

Only one primary key constraint is allowed in the table.

foreign_key_constraint

defines a link between related tables. See "Foreign Key clause" for syntax.

A column defined as a foreign key in one table references a primary key in the related table. Only values that are valid in the primary key are valid in the foreign key. The following example is valid because the foreign key values of the dept id column in the EMP table match those of the id column in the referenced table DEPT:

Referenced Table		Main Table		
DEPT		EMP		
		(Foreign Key)		
id	name	id	name	dept id
1	Dev	1	Mark	1
2	Finance	1	Jim	3
3	Sales	1	Mike	2

The following example, however, is not valid. The value 4 in the dept id column does not match any value in the referenced id column of the DEPT table.

Referenced Table		Main Table		
DEPT		EMP		
		(Foreign Key)		

Referenced Table		Main Table		
id	name	id	name	dept id
1	Dev	1	Mark	1
2	Finance	1	Jim	3
3	Sales	1	Mike	4

Unique Clause

```
UNIQUE (column_name [,column_name...])
```

where:

column_name

specifies the column to which the constraint is applied. Multiple columns names must be separated by commas.

Primary Key Clause

```
PRIMARY KEY (column_name [,column_name...])
```

where:

column_name

specifies the primary key column to which the constraint is applied. Multiple column names must be separated by commas.

Foreign Key Clause

```
FOREIGN KEY (fcolumn_name [,fcolumn_name...])  
REFERENCES ref_table (pcolumn_name [,pcolumn_name...])  
[ON {DELETE | UPDATE}  
{CASCADE | SET DEFAULT | SET NULL}]
```

fcolumn_name

specifies the foreign key column to which the constraint is applied. Multiple column names must be separated by commas.

ref_table

specifies the table to which a foreign key refers.

pcolumn_name

specifies the primary key column or columns referenced in the referenced table. Multiple column names must be separated by commas.

ON DELETE

is a clause that defines the operation performed when a row in the table referenced by a foreign key constraint is deleted. One of the following operators must be specified in the On Delete clause:

where:

- `CASCADE` specifies that all rows in the foreign key table that reference the deleted row in the primary key table are also deleted.
- `SET DEFAULT` specifies that the value of the foreign key column is set to the column default value for all rows in the foreign key table that reference the deleted row in the primary key table.
- `SET NULL` specifies that the value of the foreign key column is set to `NULL` for all rows in the foreign key table that reference the deleted row in the primary key table.

ON UPDATE

is a clause that defines the operation performed when the primary key of a row in the table referenced by a foreign key constraint is updated. One of the following operators must be specified in the On Update clause:

- `CASCADE` specifies that the value of the foreign key column for all rows in the foreign key table that reference the row in the primary key table that had the primary key updated are updated with the new primary key value.
- `SET DEFAULT` specifies that the value of the foreign key column is set to the column default value for all rows in the foreign key table that reference the row that had the primary key updated in the primary key table.
- `SET NULL` specifies that the value of the foreign key column is set to `NULL` for all rows in the foreign key table that reference the row that had the primary key updated in the primary key table.

Notes

- You must specify at least one constraint.
- Both the On Delete and On Update clauses can be used in a single foreign key definition.

Example

Assuming the current schema is `PUBLIC`, the `emp` table is created with the `name`, `empId`, and `deptId` columns. The table contains a foreign key constraint on the `deptId` column that references the `id` column in the `dept` table. In addition, it sets the value of any rows in the `deptId` column to `NULL` that point to a deleted row in the referenced `dept` table.

```
CREATE TABLE emp (name VARCHAR(30), empId INTEGER, deptId INTEGER, FOREIGN
KEY(deptId) REFERENCES dept(id)) ON DELETE SET NULL)
```

See also

[Foreign Key clause](#) on page 206

Create View

Purpose

The Create View statement creates a new view. A view is analogous to a named query. The view's query can refer to any combination of remote and local tables as well as other views. Views are read-only; they cannot be updated.

Syntax

```
CREATE VIEW view_name[(view_column,...)] AS
SELECT ... FROM ... [WHERE Expression]
  [ORDER BY order_expression [, ...]]
  [LIMIT limit [OFFSET offset]];
```

where:

view_name

specifies the name of the view.

view_column

specifies the column associated with the view. Multiple column names must be separated by commas.

The other commands used for Create View are the same as those used for Select (see "Select").

Notes

- A view can be thought of as a virtual table. A Select statement is stored in the database; however, the data accessible through a view is not stored in the database. The result set of the Select statement forms the virtual table returned by the view. You can use this virtual table by referring to the view name in SQL statements the same way you refer to a table. A view is used to perform any or all of these functions:
 - Restrict a user to specific rows in a table.
 - Restrict a user to specific columns.
 - Join columns from multiple tables so that they function like a single table.
 - Aggregate information instead of supplying details. For example, the sum of a column, or the maximum or minimum value from a column can be presented.
- Views are created by defining the Select statement that retrieves the data to be presented by the view.
- The Select statement in a View definition must return columns with distinct names. If the names of two columns in the Select statement are the same, use a column alias to distinguish between them. Alternatively, you can define a list of new columns for a view.

Example A

This example creates a view named myOpportunities that selects data from three database tables to present a virtual table of data.

```
CREATE VIEW myOpportunities AS
SELECT a.name AS AccountName,
       o.name AS OpportunityName,
```

```

        o.amount AS Amount,
        o.description AS Description
FROM Opportunity o INNER JOIN Account a
    ON o.AccountId = a.id
    INNER JOIN User u
    ON o.OwnerId = u.id
WHERE u.name = 'MyName'
    AND o.isClosed = 'false'
ORDER BY Amount desc

```

You can then refer to the `myOpportunities` view in statements just as you would refer to a table. For example:

```
SELECT * FROM myOpportunities;
```

Example B

The `myOpportunities` view contains a detailed description for each opportunity, which may not be needed when only a summary is required. A view can be built that selects only specific `myOpportunities` columns as shown in the following example:

```

CREATE VIEW myOpps_NoDesc as
SELECT AccountName,
       OpportunityName,
       Amount
FROM myOpportunities

```

The view selects the name column from both the opportunity and account tables. These columns are assigned the alias `OpportunityName` and `AccountName`, respectively.

See also

[Select](#) on page 222

Delete

Purpose

The Delete statement is used to delete rows from a table.

Syntax

```
DELETE FROM table_name [WHERE search_condition]
```

where:

table_name

specifies the name of the table from which you want to delete rows.

search_condition

is an expression that identifies which rows to delete from the table.

Notes

- The Where clause determines which rows are to be deleted. Without a Where clause, all rows of the table are deleted, but the table is left intact. See "Where Clause" for information about the syntax of Where clauses. Where clauses can contain subqueries.

Example A

This example shows a Delete statement on the emp table.

```
DELETE FROM emp WHERE emp_id = 'E10001'
```

Each Delete statement removes every record that meets the conditions in the Where clause. In this case, every record having the employee ID E10001 is deleted. Because employee IDs are unique in the employee table, at most, one record is deleted.

Example B

This example shows using a subquery in a Delete clause.

```
DELETE FROM emp WHERE dept_id = (SELECT dept_id FROM dept WHERE dept_name = 'Marketing')
```

The records of all employees who belong to the department named Marketing are deleted.

Notes

- To enable Insert, Update, and Delete, set the ReadOnly connection option to `false`.

See also

[Where clause](#) on page 228

Drop Cache (EXT)

Purpose

The Drop Cache statement drops the cache defined on a remote table. To drop a relational cache, the specified table must be the primary table of the relational cache. If a relational cache is specified, the cache for the primary table and all referenced caches are dropped.

Syntax

```
DROP CACHE ON {remote_table} [IF EXISTS]
```

where:

remote_table

is the name of the remote table cache to be dropped. The remote table name can be a two-part name: *schemaname.tablename*. When specifying a two-part name, the specified remote table must be mapped in the specified schema, and you must have the privilege to drop objects in the specified schema.

IF EXISTS

specifies that an error is not to be returned if a cache for the remote table or view does not exist.

Notes

- Caches on views are not supported.

Drop Index

Purpose

The Drop Index statement drops an index for a local table.

Syntax

```
DROP INDEX index_name [IF EXISTS]
```

where:

index_name

specifies an existing index.

```
IF EXISTS
```

specifies that an error is not to be returned if the index does not exist. The Drop Index command generates an error if an index that is associated with a UNIQUE or FOREIGN KEY constraint is specified.

Notes

- Indexes on a remote table cannot be dropped. Only indexes on local tables can be created, altered, and dropped.

Drop Sequence

Purpose

The Drop Sequence statement drops a sequence for local tables.

Syntax

```
DROP SEQUENCE sequence_name [IF EXISTS] [RESTRICT | CASCADE]
```

where:

sequence_name

specifies the name of a sequence to drop.

```
IF EXISTS
```

specifies that an error is not to be returned if the sequence does not exist.

RESTRICT

is in effect by default, meaning that the drop fails if any view refers to the sequence.

CASCADE

silently drops all dependent database objects.

Drop Table

Purpose

The Drop Table statement drops (removes) a remote or local table, its data, and its indexes. A remote table is a Salesforce object and is exposed in the SFORCE schema. Dropping a table in the SFORCE schema drops a remote table. A local table is maintained by the driver and is local to the machine on which the driver is running. A local table is exposed in the PUBLIC schema. Dropping a table in the PUBLIC schema drops a local table.

Syntax

```
DROP TABLE table_name [IF EXISTS] [RESTRICT | CASCADE]
```

where:

table_name

specifies the name of an existing table to drop.

IF EXISTS

specifies that an error is not to be returned if the table does not exist.

RESTRICT

is in effect by default, meaning that the drop fails if any tables or views reference this table.

CASCADE

specifies that the drop extends to linked objects. If the specified table is a local table, it drops all dependent views and any foreign key constraints that link this table to other tables. If the specified table is a remote table, any tables that reference the specified table are dropped also.

Drop View

Purpose

The Drop View statement drops a view.

Syntax

```
DROP VIEW view_name [IF EXISTS] [RESTRICT | CASCADE]
```

where:

view_name

specifies the name of a view.

IF EXISTS

specifies that an error is not to be returned if the view does not exist.

RESTRICT

is in effect by default, meaning that the drop fails if any other view refers to this view.

CASCADE

silently drops all dependent views.

Explain Plan

Purpose

The Explain Plan statement can be used with any query to retrieve a detailed list of the elements in the execution plan. Explain Plan generates a result set with a single column named `OPERATION`. The individual elements that comprise the plan are returned as rows in the result set.

Syntax

```
EXPLAIN PLAN FOR {SELECT ... | DELETE ... | INSERT ... | UPDATE ... }
```

The returned list of elements includes the indexes used for performing the query and can be used to optimize the query.

Insert

Purpose

The Insert statement is used to add new rows to a local table. You can specify either of the following options:

- List of values to be inserted as a new row
- Select statement that copies data from another table to be inserted as a set of new rows

Syntax

```
INSERT INTO table_name [(column_name [, column_name] ...)] {VALUES (expression [, expression] ...) | select_statement}
```

table_name

is the name of the table in which you want to insert rows.

column_name

is optional and specifies an existing column. Multiple column names (a column list) must be separated by commas. A column list provides the name and order of the columns, the values of which are specified in the Values clause. If you omit a *column_name* or a column list, the value expressions must provide values for all columns defined in the table and must be in the same order that the columns are defined for the table. Table columns that do not appear in the column list are populated with the default value, or with NULL if no default value is specified.

expression

is the list of expressions that provides the values for the columns of the new record. Typically, the expressions are constant values for the columns. Character string values must be enclosed in single quotation marks ('). See "Literals" for more information.

select_statement

is a query that returns values for each *column_name* value specified in the column list. Using a Select statement instead of a list of value expressions lets you select a set of rows from one table and insert it into another table using a single Insert statement. The Select statement is evaluated before any values are inserted. This query cannot be made on the table into which values are inserted. See "Select" for information about Select statements.

Notes

- To enable Insert, Update, and Delete, set the ReadOnly connection option to `false`.

See also

[Literals](#) on page 235

[Select](#) on page 222

Specifying an external ID column

To specify an external ID column to look up the value of a foreign key column, use the following syntax:

```
column_name EXT_ID [schema_name.table_name.]ext_id_column
```

where:

EXT_ID

is used to specify that the column specified by *ext_id_column* is used to look up the rowid to be inserted into the column specified by *column_name*.

schema_name

is the name of the schema of the table that contains the foreign key column being specified as the external ID column.

table_name

is the name of the table that contains the foreign key column being specified as the external ID column.

`ext_id_column`

is the external ID column.

Example A

The following example uses a list of expressions to insert records. Each Insert statement adds one record to the database table. In this case, one record is added to the table `emp`. Values are specified for five columns. The remaining columns in the table are assigned the default value or NULL if no default value is specified.

```
INSERT INTO emp (last_name,
                first_name,
                emp_id,
                salary,
                hire_date)
VALUES ('Smith', 'John', 'E22345', 27500, {1999-04-06})
```

Example B

The following example uses a Select statement to insert records. The number of columns in the result of the Select statement must match exactly the number of columns in the table if no column list is specified, or it must match the number of column names specified in the column list. A new entry is created in the table for every row of the Select result.

```
INSERT INTO emp1 (first_name,
                 last_name,
                 emp_id,
                 dept,
                 salary)
SELECT first_name, last_name, emp_id, dept, salary FROM emp
WHERE dept = 'D050'
```

Example C

The following example uses a list of expressions to insert records and specifies an external ID column (a foreign key column) named `accountId` that references a table that has an external ID column named `AccountNum`.

```
INSERT INTO emp (last_name,
                first_name,
                emp_id,
                salary,
                hire_date,
                accountId EXT_ID AccountNum)
VALUES ('Smith', 'John', 'E22345', 27500, {1999-04-06}, 0001)
```

Refresh Cache (EXT)

Purpose

The Refresh Cache statement forces the data in the cache for the specified remote table to be refreshed.

Syntax

```
REFRESH CACHE ON {remote_table | ALL} [CLEAN]
```

where:

remote_table

is the name of the remote table cache to be refreshed. The remote table name can be a two-part name:

schemaname.tablename

. When specifying a two-part name, the specified remote table must be mapped in the specified schema, and you must have the privilege to insert, update, and delete objects in the specified schema.

ALL

forces all caches to be refreshed.

CLEAN

is optional and discards the data in the cache for the specified table or view, or all cache data if ALL is specified, and repopulates the cache with the data in the remote table or view.

Notes

- Caches on views are not supported.

Refresh Map (EXT)

Purpose

The REFRESH MAP statement adds newly discovered objects to your relational view of native data. It also incorporates any configuration changes made to your relational view by reloading the schema map and associated files.

Syntax

```
REFRESH MAP
```

Notes

- REFRESH MAP is an expensive query since it involves the discovery of native data.

Select

Purpose

Use the Select statement to fetch results from one or more tables.

Syntax

```
SELECT select_clause from_clause  
[where_clause]  
[groupby_clause]  
[having_clause]  
[ {UNION [ALL | DISTINCT] |
```

```
{MINUS [DISTINCT] | EXCEPT [DISTINCT]} |  
INTERSECT [DISTINCT]} select_statement |  
limit_clause
```

where:*select_clause*

specifies the columns from which results are to be returned by the query. See "Select clause" for a complete explanation.

from_clause

specifies one or more tables on which the other clauses in the query operate. See "From clause" for a complete explanation.

where_clause

is optional and restricts the results that are returned by the query. See "Where clause" for a complete explanation.

groupby_clause

is optional and allows query results to be aggregated in terms of groups. See "Group By clause" for a complete explanation.

having_clause

is optional and specifies conditions for groups of rows (for example, display only the departments that have salaries totaling more than \$200,000). See "Having clause" for a complete explanation.

UNION

is an optional operator that combines the results of the left and right Select statements into a single result. See "Union operator" for a complete explanation.

INTERSECT

is an optional operator that returns a single result by keeping any distinct values from the results of the left and right Select statements. See "Intersect operator" for a complete explanation.

EXCEPT | MINUS

are synonymous optional operators that returns a single result by taking the results of the left Select statement and removing the results of the right Select statement. See "Except and Minus operators" for a complete explanation.

orderby_clause

is optional and sorts the results that are returned by the query. See "Order By clause" for a complete explanation.

limit_clause

is optional and places an upper bound on the number of rows returned in the result. See "Limit clause" for a complete explanation.

See also

[Select clause](#) on page 224

[From clause](#) on page 226

[Where clause](#) on page 228

[Group By clause](#) on page 229

[Having clause](#) on page 229

[Union operator](#) on page 230

[Intersect operator](#) on page 230

[Except and Minus operators](#) on page 231

[Order By clause](#) on page 232

[Limit clause](#) on page 233

Select clause

Purpose

Use the Select clause to specify with a list of column expressions that identify columns of values that you want to retrieve or an asterisk (*) to retrieve the value of all columns.

Syntax

```
SELECT [{LIMIT offsetnumber | TOP number}] [ALL | DISTINCT] {* | column_expression
[[AS] column_alias] [,column_expression [[AS] column_alias], ...]}
```

where:

LIMIT offset number

creates the result set for the Select statement first and then discards the first number of rows specified by *offset* and returns the number of remaining rows specified by *number*. To not discard any of the rows, specify 0 for *offset*, for example, `LIMIT 0 number`. To discard the first *offset* number of rows and return all the remaining rows, specify 0 for *number*, for example, `LIMIT offset 0`.

TOP number

is equivalent to `LIMIT 0 number`.

column_expression

can be simply a column name (for example, `last_name`). More complex expressions may include mathematical operations or string manipulation (for example, `salary * 1.05`). See "SQL expressions" for details. *column_expression* can also include aggregate functions. See "Aggregate Functions" for details.

column_alias

can be used to give the column a descriptive name. For example, to assign the alias `department` to the column `dep`:

```
SELECT dep AS department FROM emp
```


DISTINCT

eliminates duplicate rows from the result of a query. This operator can precede the first column expression. For example:

```
SELECT DISTINCT dep FROM emp
```

Notes

- Separate multiple column expressions with commas (for example, `SELECT last_name, first_name, hire_date`).
- Column names can be prefixed with the table name or table alias. For example, `SELECT emp.last_name` or `e.last_name`, where `e` is the alias for the table `emp`.
- NULL values are not treated as distinct from each other. The default behavior is that all result rows be returned, which can be made explicit with the keyword `ALL`.

See also

[SQL expressions](#) on page 234

[Aggregate functions](#) on page 225

Aggregate functions

Aggregate functions can also be a part of a Select clause. Aggregate functions return a single value from a set of rows. An aggregate can be used with a column name (for example, `AVG(salary)`) or in combination with a more complex column expression (for example, `AVG(salary * 1.07)`). The column expression can be preceded by the `DISTINCT` operator. The `DISTINCT` operator eliminates duplicate values from an aggregate expression.

The following table lists supported aggregate functions.

Note: Doubly nested aggregates, such as `SUM(COUNT(col1))`, are currently not permitted by the driver.

Table 21: Aggregate Functions

Aggregate	Returns
AVG	The average of the values in a numeric column expression. For example, <code>AVG(salary)</code> returns the average of all salary column values.
COUNT	<p>The number of values in any field expression. For example, <code>COUNT(name)</code> returns the number of name values. When using <code>COUNT</code> with a field name, <code>COUNT</code> returns the number of non-NULL column values. A special example is <code>COUNT(*)</code>, which returns the number of rows in the set, including rows with NULL values.</p> <hr/> <p>Note: The driver does not support <code>COUNT(DISTINCT *)</code>. For example, <code>SELECT COUNT(DISTINCT *) FROM mytable</code> results in a syntax error.</p> <hr/>
MAX	The maximum value in any column expression. For example, <code>MAX(salary)</code> returns the maximum salary column value.

MIN	The minimum value in any column expression. For example, <code>MIN(salary)</code> returns the minimum salary column value.
SUM	The total of the values in a numeric column expression. For example, <code>SUM(salary)</code> returns the sum of all salary column values.

Example A

In the following example, only distinct last name values are counted. The default behavior is that all duplicate values be returned, which can be made explicit with `ALL`.

```
COUNT (DISTINCT last_name)
```

Example B

The following example uses the `COUNT`, `MAX`, and `AVG` aggregate functions:

```
SELECT
    COUNT(amount) AS numOpportunities,
    MAX(amount) AS maxAmount,
    AVG(amount) AS avgAmount
FROM opportunity o INNER JOIN user u
    ON o.ownerId = u.id
WHERE o.isClosed = 'false' AND
    u.name = 'MyName'
```

From clause

Purpose

The From clause indicates the tables to be used in the Select statement.

Syntax

```
FROM table_name [table_alias] [, ...]
```

where:

table_name

is the name of a table or a subquery. Multiple tables define an implicit inner join among those tables. Multiple table names must be separated by a comma. For example:

```
SELECT * FROM emp, dep
```

Subqueries can be used instead of table names. Subqueries must be enclosed in parentheses. See "Subquery in a From clause" for an example.

table_alias

is a name used to refer to a table in the rest of the Select statement. When you specify an alias for a table, you can prefix all column names of that table with the table alias.

Example

The following example specifies two table aliases, e for emp and d for dep:

```
SELECT e.name, d.deptName
FROM emp e, dep d
WHERE e.deptId = d.id
```

table_alias is a name used to refer to a table in the rest of the Select statement. When you specify an alias for a table, you can prefix all column names of that table with the table alias. For example, given the table specification:

```
FROM emp E
```

you may refer to the last_name field as E.last_name. Table aliases must be used if the Select statement joins a table to itself. For example:

```
SELECT * FROM emp E, emp F WHERE E.mgr_id = F.emp_id
```

The equal sign (=) includes only matching rows in the results.

See also

[Subquery in a From clause](#) on page 228

Join in a From clause

Purpose

You can use a Join as a way to associate multiple tables within a Select statement. Joins may be either explicit or implicit. For example, the following is the example from the previous section restated as an explicit inner join:

```
SELECT * FROM emp INNER JOIN dep ON id=empId
SELECT e.name, d.deptName
FROM emp e INNER JOIN dep d ON e.deptId = d.id;
```

whereas the following is the same statement as an implicit inner join:

```
SELECT * FROM emp, dep WHERE emp.deptID=dep.id
```

Note: The ON clause in a join expression must evaluate to a true or false value.

Syntax

```
FROM table_name {RIGHT OUTER | INNER | LEFT OUTER | CROSS | FULL OUTER} JOIN table.key
ON search-condition
```

Example

In this example, two tables are joined using LEFT OUTER JOIN. T1, the first table named includes nonmatching rows.

```
SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.key = T2.key
```

If you use a CROSS JOIN, no ON expression is allowed for the join.

Subquery in a From clause

Subqueries can be used in the From clause in place of table references (*table_name*).

Example

```
SELECT * FROM (SELECT * FROM emp WHERE sal > 10000) new_emp, dept WHERE
new_emp.deptno = dept.deptno
```

See also

For more information about subqueries, see "Subqueries".

See also

[Subqueries](#) on page 242

Where clause

Purpose

Specifies the conditions that rows must meet to be retrieved.

Syntax

```
WHERE expr1rel_operatorexpr2
```

where:

expr1

is either a column name, literal, or expression.

expr2

is either a column name, literal, expression, or subquery. Subqueries must be enclosed in parentheses.

rel_operator

is the relational operator that links the two expressions.

Example

The following Select statement retrieves the first and last names of employees that make at least \$20,000.

```
SELECT last_name, first_name FROM emp WHERE salary >= 20000
```

See also

[Subqueries](#) on page 242

[SQL expressions](#) on page 234

Group By clause

Purpose

Specifies the names of one or more columns by which the returned values are grouped. This clause is used to return a set of aggregate values.

Syntax

```
GROUP BY column_expression [, ...]
```

where:

column_expression

is either a column name or a SQL expression. Multiple values must be separated by a comma. If *column_expression* is a column name, it must match one of the column names specified in the Select clause. Also, the Group By clause must include all non-aggregate columns specified in the Select list.

Example

The following example totals the salaries in each department:

```
SELECT dept_id, sum(salary) FROM emp GROUP BY dept_id
```

This statement returns one row for each distinct department ID. Each row contains the department ID and the sum of the salaries of the employees in the department.

See also

[Subqueries](#) on page 242

[SQL expressions](#) on page 234

Having clause

Purpose

Specifies conditions for groups of rows (for example, display only the departments that have salaries totaling more than \$200,000). This clause is valid only if you have already defined a Group By clause.

Syntax

```
HAVING expr1rel_operatorexpr2
```

where:

expr1 | *expr2*

can be column names, constant values, or expressions. These expressions do not have to match a column expression in the Select clause. See "SQL expressions" for details regarding SQL expressions.

rel_operator

is the relational operator that links the two expressions.

Example

The following example returns only the departments that have salaries totaling more than \$200,000:

```
SELECT dept_id, sum(salary) FROM emp GROUP BY dept_id HAVING sum(salary) > 200000
```

See also

[SQL expressions](#) on page 234

[Subqueries](#) on page 242

Union operator

Purpose

Combines the results of two Select statements into a single result. The single result is all the returned rows from both Select statements. By default, duplicate rows are not returned. To return duplicate rows, use the All keyword (`UNION ALL`).

Syntax

```
select_statement  
UNION [ALL | DISTINCT] | {MINUS [DISTINCT] | EXCEPT [DISTINCT]} | INTERSECT  
[DISTINCT]select_statement
```

Notes

- When using the Union operator, the Select lists for each Select statement must have the same number of column expressions with the same data types and must be specified in the same order.

Example A

The following example has the same number of column expressions, and each column expression, in order, has the same data type.

```
SELECT last_name, salary, hire_date FROM emp  
UNION  
SELECT name, pay, birth_date FROM person
```

Example B

The following example is *not* valid because the data types of the column expressions are different (`salary FROM emp` has a different data type than `last_name FROM raises`). This example does have the same number of column expressions in each Select statement but the expressions are not in the same order by data type.

```
SELECT last_name, salary FROM emp  
UNION  
SELECT salary, last_name FROM raises
```

Intersect operator

Purpose

Intersect operator returns a single result set. The result set contains rows that are returned by both Select statements. Duplicates are returned unless the Distinct operator is added.

Syntax

```
select_statement
INTERSECT [DISTINCT]
select_statement
```

where:

DISTINCT

eliminates duplicate rows from the results.

Notes

- When using the Intersect operator, the Select lists for each Select statement must have the same number of column expressions with the same data types and must be specified in the same order.

Example A

The following example has the same number of column expressions, and each column expression, in order, has the same data type.

```
SELECT last_name, salary, hire_date FROM emp
INTERSECT [DISTINCT]
SELECT name, pay, birth_date FROM person
```

Example B

The following example is *not* valid because the data types of the column expressions are different (`salary FROM emp` has a different data type than `last_name FROM raises`). This example does have the same number of column expressions in each Select statement but the expressions are not in the same order by data type.

```
SELECT last_name, salary FROM emp
INTERSECT
SELECT salary, last_name FROM raises
```

Except and Minus operators

Purpose

Return the rows from the left Select statement that are not included in the result of the right Select statement.

Syntax

```
select_statement
{EXCEPT [DISTINCT] | MINUS [DISTINCT]}
select_statement
```

where:

DISTINCT

eliminates duplicate rows from the results.

Notes

- When using one of these operators, the Select lists for each Select statement must have the same number of column expressions with the same data types and must be specified in the same order.

Example A

The following example has the same number of column expressions, and each column expression, in order, has the same data type.

```
SELECT last_name, salary, hire_date FROM emp
EXCEPT
SELECT name, pay, birth_date FROM person
```

Example B

The following example is *not* valid because the data types of the column expressions are different (`salary FROM emp` has a different data type than `last_name FROM raises`). This example does have the same number of column expressions in each Select statement but the expressions are not in the same order by data type.

```
SELECT last_name, salary FROM emp
EXCEPT
SELECT salary, last_name FROM raises
```

Order By clause

Purpose

The Order By clause specifies how the rows are to be sorted.

Syntax

```
ORDER BY sort_expression [DESC | ASC] [,...]
```

where:

sort_expression

is either the name of a column, a column alias, a SQL expression, or the positioned number of the column or expression in the select list to use.

The default is to perform an ascending (ASC) sort.

Example

To sort by `last_name` and then by `first_name`, you could use either of the following Select statements:

```
SELECT emp_id, last_name, first_name FROM emp
ORDER BY last_name, first_name
```

or

```
SELECT emp_id, last_name, first_name FROM emp
ORDER BY 2,3
```

In the second example, `last_name` is the second item in the Select list, so `ORDER BY 2,3` sorts by `last_name` and then by `first_name`.

See also

[SQL expressions](#) on page 234

Limit clause**Purpose**

Places an upper bound on the number of rows returned in the result.

Syntax

```
LIMIT number_of_rows [OFFSET offset_number]
```

where:

number_of_rows

specifies a maximum number of rows in the result. A negative number indicates no upper bound.

OFFSET

specifies how many rows to skip at the beginning of the result set. *offset_number* is the number of rows to skip.

Notes

- In a compound query, the Limit clause can appear only on the final Select statement. The limit is applied to the entire query, not to the individual Select statement to which it is attached.

Example

The following example returns a maximum of 20 rows.

```
SELECT last_name, first_name FROM emp WHERE salary > 20000 ORDER BY dept_idc LIMIT 20
```

Update

Purpose

An Update statement changes the value of columns in the selected rows of a table.

Syntax

```
UPDATE table_name SET column_name = expression  
[, column_name = expression] [WHERE conditions]
```

table_name

is the name of the table for which you want to update values.

column_name

is the name of a column, the value of which is to be changed. Multiple column values can be changed in a single statement.

expression

is the new value for the column. The expression can be a constant value or a subquery that returns a single value. Subqueries must be enclosed in parentheses.

Example A

The following example changes every record that meets the conditions in the Where clause. In this case, the salary and exempt status are changed for all employees having the employee ID E10001. Because employee IDs are unique in the emp table, only one record is updated.

```
UPDATE emp SET salary=32000, exempt=1
WHERE emp_id = 'E10001'
```

Example B

The following example uses a subquery. In this example, the salary is changed to the average salary in the company for the employee having employee ID E10001.

```
UPDATE emp SET salary = (SELECT avg(salary) FROM emp)
WHERE emp_id = 'E10001'
```

Notes

- To enable Insert, Update, and Delete, set the ReadOnly connection option to `false`.
- A Where clause can be used to restrict which rows are updated.

See also

[Subqueries](#) on page 242

[Where clause](#) on page 228

SQL expressions

An expression is a combination of one or more values, operators, and SQL functions that evaluate to a value. You can use expressions in the Where, and Having of Select statements; and in the Set clauses of Update statements.

Expressions enable you to use mathematical operations as well as character string manipulation operators to form complex queries.

The Salesforce driver supports both unquoted and quoted identifiers. An unquoted identifier must start with an ASCII alpha character and can be followed by zero or more ASCII alphanumeric characters. Unquoted identifiers are converted to uppercase before being used.

Quoted identifiers must be enclosed in double quotation marks ("). A quoted identifier can contain any Unicode character including the space character. The Salesforce driver recognizes the Unicode escape sequence `\uxxxx` as a Unicode character. You can specify a double quotation mark in a quoted identifier by escaping it with a double quotation mark.

The maximum length of both quoted and unquoted identifiers is 128 characters.

Valid expression elements are:

- Column names
- Literals
- Operators
- Functions

Column names

The most common expression is a simple column name. You can combine a column name with other expression elements.

Literals

Literals are fixed data values. For example, in the expression `PRICE * 1.05`, the value 1.05 is a constant. Literals are classified into types, including the following:

- Binary
- Character string
- Date
- Floating point
- Integer
- Numeric
- Time
- Timestamp

The following table describes the literal format for supported SQL data types.

Table 22: Literal Syntax Examples

SQL Type	Literal Syntax	Example
BIGINT	<i>n</i> where <i>n</i> is any valid integer value in the range of the INTEGER data type	12 or -34 or 0
BOOLEAN	Min Value: 0 Max Value: 1	0 1
DATE	DATE' <i>date</i> '	'2010-05-21'
DATETIME	TIMESTAMP' <i>ts</i> '	'2010-05-21 18:33:05.025'

SQL Type	Literal Syntax	Example
DECIMAL	$n.f$ where: n is the integral part f is the fractional part	0.25 3.1415 -7.48
DOUBLE	$n.fEx$ where: n is the integral part f is the fractional part x is the exponent	1.2E0 or 2.5E40 or -3.45E2 or 5.67E-4
INTEGER	n where n is a valid integer value in the range of the INTEGER data type	12 or -34 or 0
LONGVARBINARY	' <i>hex_value</i> '	'000482ff'
LONGVARCHAR	' <i>value</i> '	'This is a string literal'
TIME	TIME' <i>time</i> '	'2010-05-21 18:33:05.025'
VARCHAR	' <i>value</i> '	'This is a string literal'

Character string literals

Text specifies a character string literal. A character string literal must be enclosed in single quotation marks. To represent one single quotation mark within a literal, you must enter two single quotation marks. When the data in the fields is returned to the client, trailing blanks are stripped.

A character string literal can have a maximum length of 32 KB, that is, (32*1024) bytes.

Example

```
'Hello'  
'Jim''s friend is Joe'
```

Numeric literals

Unquoted numeric values are treated as numeric literals. If the unquoted numeric value contains a decimal point or exponent, it is treated as a real literal; otherwise, it is treated as an integer literal.

Example

+1894.1204

Binary literals

Binary literals are represented with single quotation marks. The valid characters in a binary literal are 0-9, a-f, and A-F.

Example

'00af123d'

Date/time literals

Date and time literal values are enclosed in single quotation marks (*'value'*). Review this section.

- The format for a Date literal is DATE'*yyyy-mm-dd*'.
- The format for a Time literal is TIME'*hh:mm:ss*'.
- The format for a Timestamp literal is TIMESTAMP'*yyyy-mm-dd hh:mm:ss.SSSSSS*'.

Integer literals

Integer literals are represented by a string of numbers that are not enclosed in quotation marks and do not contain decimal points.

Notes

- Integer constants must be whole numbers; they cannot contain decimals.
- Integer literals can start with sign characters (+/-).

Example

1994 or -2

Operators

This section describes the operators that can be used in SQL expressions.

Note: Numeric operators are restricted to numeric types. Numeric operators do not support non-numeric types.

Unary operator

A unary operator operates on only one operand.

*operator operand***Binary operator**

A binary operator operates on two operands.

operand1 operator operand2

If an operator is given a null operand, the result is always null. The only operator that does not follow this rule is concatenation (||).

Arithmetic operators

You can use an arithmetic operator in an expression to negate, add, subtract, multiply, and divide numeric values. The result of this operation is also a numeric value. The + and - operators are also supported in date/time fields to allow date arithmetic. The following table lists the supported arithmetic operators.

Table 23: Arithmetic Operators

Operator	Purpose	Example
+ -	Denotes a positive or negative expression. These are unary operators.	<code>SELECT * FROM emp WHERE comm = -1</code>
* /	Multiplies, divides. These are binary operators.	<code>UPDATE emp SET sal = sal + sal * 0.10</code>
+ -	Adds, subtracts. These are binary operators.	<code>SELECT sal + comm FROM emp WHERE empno > 100</code>

Concatenation operator

The concatenation operator manipulates character strings. The following table lists the only supported concatenation operator.

Table 24: Concatenation Operator

Operator	Purpose	Example
	Concatenates character strings.	<code>SELECT 'Name is' ename FROM emp</code>

The result of concatenating two character strings is the data type VARCHAR.

Comparison operators

Comparison operators compare one expression to another. The result of such a comparison can be TRUE, FALSE, or UNKNOWN (if one of the operands is NULL). The Salesforce driver considers the UNKNOWN result as FALSE.

The following table lists the supported comparison operators.

Table 25: Comparison Operators

Operator	Purpose	Example
=	Equality test.	<code>SELECT * FROM emp WHERE sal = 1500</code>
!=<>	Inequality test.	<code>SELECT * FROM emp WHERE sal != 1500</code>

Operator	Purpose	Example
><	"Greater than" and "less than" tests.	SELECT * FROM emp WHERE sal > 1500 SELECT * FROM emp WHERE sal < 1500
>=<=	"Greater than or equal to" and "less than or equal to" tests.	SELECT * FROM emp WHERE sal >= 1500 SELECT * FROM emp WHERE sal <= 1500
ESCAPE clause in LIKE operator LIKE 'pattern string' ESCAPE 'c'	The Escape clause is supported in the LIKE predicate to indicate the escape character. Escape characters are used in the pattern string to indicate that any wildcard character that is after the escape character in the pattern string should be treated as a regular character. The default escape character is backslash (\).	SELECT * FROM emp WHERE ENAME LIKE 'J%_%' ESCAPE '\' This matches all records with names that start with letter 'J' and have the '_' character in them. SELECT * FROM emp WHERE ENAME LIKE 'JOE_JOHN' ESCAPE '\' This matches only records with name 'JOE_JOHN'.
LIKE	% and _ wildcards can be used to search for a pattern in a column. The percent sign denotes zero, one, or multiple characters, while the underscore denotes a single character. The right-hand side of a LIKE expression must evaluate to a string or binary.	SELECT * FROM emp WHERE ENAME LIKE 'J%'
[NOT] IN	"Equal to any member of" test.	SELECT * FROM emp WHERE job IN ('CLERK', 'ANALYST') SELECT * FROM emp WHERE sal IN (SELECT sal FROM emp WHERE deptno = 30)
[NOT] BETWEEN x AND y	"Greater than or equal to x" and "less than or equal to y."	SELECT * FROM emp WHERE sal BETWEEN 2000 AND 3000
EXISTS	Tests for existence of rows in a subquery.	SELECT empno, ename, deptno FROM emp e WHERE EXISTS (SELECT deptno FROM dept WHERE e.deptno = dept.deptno)
IS [NOT] NULL	Tests whether the value of the column or expression is NULL.	SELECT * FROM emp WHERE ename IS NOT NULL SELECT * FROM emp WHERE ename IS NULL

Logical operators

A logical operator combines the results of two component conditions to produce a single result or to invert the result of a single condition. The following table lists the supported logical operators.

Table 26: Logical Operators

Operator	Purpose	Example
NOT	Returns TRUE if the following condition is FALSE. Returns FALSE if it is TRUE. If it is UNKNOWN, it remains UNKNOWN.	<pre>SELECT * FROM emp WHERE NOT (job IS NULL) SELECT * FROM emp WHERE NOT (sal BETWEEN 1000 AND 2000)</pre>
AND	Returns TRUE if both component conditions are TRUE. Returns FALSE if either is FALSE; otherwise, returns UNKNOWN.	<pre>SELECT * FROM emp WHERE job = 'CLERK' AND deptno = 10</pre>
OR	Returns TRUE if either component condition is TRUE. Returns FALSE if both are FALSE; otherwise, returns UNKNOWN.	<pre>SELECT * FROM emp WHERE job = 'CLERK' OR deptno = 10</pre>

Example

In the Where clause of the following Select statement, the AND logical operator is used to ensure that managers earning more than \$1000 a month are returned in the result:

```
SELECT * FROM emp WHERE jobtitle = manager AND sal > 1000
```

Operator precedence

As expressions become more complex, the order in which the expressions are evaluated becomes important. The following table shows the order in which the operators are evaluated. The operators in the first line are evaluated first, then those in the second line, and so on. Operators in the same line are evaluated left to right in the expression. You can change the order of precedence by using parentheses. Enclosing expressions in parentheses forces them to be evaluated together.

Table 27: Operator Precedence

Precedence	Operator
1	+ (Positive), - (Negative)
2	*(Multiply), / (Division)
3	+ (Add), - (Subtract)
4	(Concatenate)
5	=, >, <, >=, <=, <>, != (Comparison operators)
6	NOT, IN, LIKE

Precedence	Operator
7	AND
8	OR

Example A

The query in the following example returns employee records for which the department number is 1 or 2 and the salary is greater than \$1000:

```
SELECT * FROM emp WHERE (deptno = 1 OR deptno = 2) AND sal > 1000
```

Because parenthetical expressions are forced to be evaluated first, the OR operation takes precedence over AND.

Example B

In the following example, the query returns records for all the employees in department 1, but only employees whose salary is greater than \$1000 in department 2.

```
SELECT * FROM emp WHERE deptno = 1 OR deptno = 2 AND sal > 1000
```

The AND operator takes precedence over OR, so that the search condition in the example is equivalent to the expression `deptno = 1 OR (deptno = 2 AND sal > 1000)`.

Functions

The Salesforce driver supports a number of functions that you can use in expressions.

Refer to "ODBC API and scalar functions" in the *Progress DataDirect for ODBC Drivers Reference* for additional information.

Conditions

A condition specifies a combination of one or more expressions and logical operators that evaluates to either TRUE, FALSE, or UNKNOWN. You can use a condition in the Where clause of the Delete, Select, and Update statements; and in the Having clauses of Select statements. The following describes supported conditions.

Table 28: Conditions

Condition	Description
Simple comparison	Specifies a comparison with expressions or subquery results. = , !=, <>, < , >, <=, >=
Group comparison	Specifies a comparison with any or all members in a list or subquery. [= , !=, <>, < , >, <=, >=] [ANY, ALL, SOME]

Condition	Description
Membership	Tests for membership in a list or subquery. [NOT] IN
Range	Tests for inclusion in a range. [NOT] BETWEEN
NULL	Tests for nulls. IS NULL, IS NOT NULL
EXISTS	Tests for existence of rows in a subquery. [NOT] EXISTS
LIKE	Specifies a test involving pattern matching. [NOT] LIKE
Compound	Specifies a combination of other conditions. CONDITION [AND/OR] CONDITION

Subqueries

A query is an operation that retrieves data from one or more tables or views. In this reference, a top-level query is called a Select statement, and a query nested within a Select statement is called a subquery.

A subquery is a query expression that appears in the body of another expression such as a Select, an Update, or a Delete statement. In the following example, the second Select statement is a subquery:

```
SELECT * FROM emp WHERE deptno IN (SELECT deptno FROM dept)
```

IN predicate

Purpose

The In predicate specifies a set of values against which to compare a result set. If the values are being compared against a subquery, only a single column result set is returned.

Syntax

```
value [NOT] IN (value1, value2,...)
```

OR

```
value [NOT] IN (subquery)
```

Example

```
SELECT * FROM emp WHERE deptno IN
(SELECT deptno FROM dept WHERE dname <> 'Sales')
```

EXISTS predicate

Purpose

The Exists predicate is true only if the cardinality of the subquery is greater than 0; otherwise, it is false.

Syntax

```
EXISTS (subquery)
```

Example

```
SELECT empno, ename, deptno FROM emp e WHERE EXISTS
(SELECT deptno FROM dept WHERE e.deptno = dept.deptno)
```

UNIQUE predicate

Purpose

The Unique predicate is used to determine whether duplicate rows exist in a virtual table (one returned from a subquery).

Syntax

```
UNIQUE (subquery)
```

Example

```
SELECT * FROM dept d WHERE UNIQUE
(SELECT deptno FROM emp e WHERE e.deptno = d.deptno)
```

Correlated subqueries

Purpose

A correlated subquery is a subquery that references a column from a table referred to in the parent statement. A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a Select, Update, or Delete statement.

A correlated subquery answers a multiple-part question in which the answer depends on the value in each row processed by the parent statement. For example, you can use a correlated subquery to determine which employees earn more than the average salaries for their departments. In this case, the correlated subquery specifically computes the average salary for each department.

Syntax

```
SELECT select_list
  FROM table1 t_alias1
  WHERE expr rel_operator
    (SELECT column_list
      FROM table2 t_alias2
      WHERE t_alias1.columnrel_operatort_alias2.column)
UPDATE table1 t_alias1
  SET column =
    (SELECT expr
      FROM table2 t_alias2
      WHERE t_alias1.column = t_alias2.column)
DELETE FROM table1 t_alias1
  WHERE column rel_operator
    (SELECT expr
      FROM table2 t_alias2
      WHERE t_alias1.column = t_alias2.column)
```

Notes

- Correlated column names in correlated subqueries must be explicitly qualified with the table name of the parent.

Example A

The following statement returns data about employees whose salaries exceed their department average. This statement assigns an alias to `emp`, the table containing the salary information, and then uses the alias in a correlated subquery:

```
SELECT deptno, ename, sal FROM emp x WHERE sal >
  (SELECT AVG(sal) FROM emp WHERE x.deptno = deptno)
ORDER BY deptno
```

Example B

This is an example of a correlated subquery that returns row values:

```
SELECT * FROM dept "outer" WHERE 'manager' IN
  (SELECT managername FROM emp
  WHERE "outer".deptno = emp.deptno)
```

Example C

This is an example of finding the department number (`deptno`) with multiple employees:

```
SELECT * FROM dept main WHERE 1 <
  (SELECT COUNT(*) FROM emp WHERE deptno = main.deptno)
```

Example D

This is an example of correlating a table with itself:

```
SELECT deptno, ename, sal FROM emp x WHERE sal >
  (SELECT AVG(sal) FROM emp WHERE x.deptno = deptno)
```