



# **Progress DataDirect for JDBC for Oracle Sales Cloud User's Guide**

***6.0.0 Release***



# 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: 2025/10/16**



# Table of Contents

## Welcome to the Progress DataDirect for JDBC for Oracle Sales Cloud

<b>Driver</b> .....	<b>9</b>
What's New in this Release?.....	10
Requirements.....	10
Driver and DataSource classes.....	11
Connection URL.....	11
Version string information.....	12
Connection properties and configuration options.....	12
Data types.....	12
getTypeInfo().....	13
SQL escape sequences.....	15
Supported scalar functions.....	16
Mapping objects to tables.....	17
DataDirect tools.....	18
Troubleshooting.....	18
Additional information .....	18
Contacting Technical Support.....	18
 <b>Getting started</b> .....	 <b>21</b>
Driver and DataSource classes.....	21
Connecting using the DriverManager.....	22
Setting the class path .....	22
Passing the connection URL.....	22
Testing the connection.....	23
Connecting using data sources.....	26
How data sources are implemented.....	26
Creating data sources.....	27
Calling a DataSource in an application.....	27
 <b>Using the driver</b> .....	 <b>29</b>
Connecting from an application.....	30
Driver and DataSource classes.....	30
Connecting using the DriverManager.....	30
Connecting using data sources.....	34
Using connection properties.....	35
Required properties.....	36
Mapping properties.....	36

Web service properties.....	37
Proxy server properties.....	37
Additional properties.....	37
Proxy server support.....	38
Unicode support.....	38
Data encryption.....	39
FIPS (Federal Information Processing Standard).....	39
Catalog Tables.....	39
SYSTEM_REMOTE_SESSIONS catalog table.....	39
SYSTEM_SESSIONS catalog table.....	40
Identifiers.....	41
Timeouts.....	42
Scrollable cursors.....	42
Parameter metadata support.....	43
ResultSet metadata support.....	43
Error handling.....	44
Rowset support.....	44
<b>Connection property descriptions.....</b>	<b>47</b>
ConfigOptions.....	49
APIVersion (configuration option).....	50
CreateMap.....	50
LogConfigFile.....	51
Password.....	52
ProxyHost.....	52
ProxyPassword.....	53
ProxyPort.....	54
ProxyUser.....	54
SchemaMap.....	55
ServerName.....	57
SpyAttributes.....	58
User.....	60
WSFetchSize.....	61
WSRetryCount.....	61
WSTimeout.....	62
<b>Supported SQL functionality.....</b>	<b>65</b>
Alter Session (EXT).....	65
Refresh Map (EXT).....	67
Select.....	67
Select clause.....	68
SQL expressions.....	78
Column names.....	78

Literals.....78

Operators.....81

Functions.....84

Conditions.....84

Subqueries.....85

    IN predicate.....86

    EXISTS predicate.....86

    UNIQUE predicate.....86

    Correlated subqueries.....87



---

# Welcome to the Progress DataDirect for JDBC for Oracle Sales Cloud Driver

---

The Progress® DataDirect® for JDBC™ for Oracle Sales Cloud™ driver supports SQL read access to Oracle Sales Cloud. To support SQL access to Oracle Sales Cloud, the driver creates a relational map of the Oracle Sales Cloud data model and translates SQL statements provided by the application to Oracle Sales Cloud queries and web service calls.

The documentation for the driver also includes the *Progress DataDirect for JDBC Drivers Reference*. The reference provides general reference information for all DataDirect drivers for JDBC, including content on troubleshooting, supported SQL escapes, and DataDirect tools.

For the complete documentation set, visit the Progress DataDirect Connectors Documentation Hub: <https://docs.progress.com/category/datadirect-oracle-sales-cloud>.

For details, see the following topics:

- [What's New in this Release?](#)
- [Requirements](#)
- [Driver and DataSource classes](#)
- [Connection URL](#)
- [Version string information](#)
- [Connection properties and configuration options](#)
- [Data types](#)
- [SQL escape sequences](#)

- [Mapping objects to tables](#)
- [DataDirect tools](#)
- [Troubleshooting](#)
- [Additional information](#)
- [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/datadirect-connectors/whats-new#jdbc>
- DataDirect Product Compatibility Guide: <https://docs.progress.com/bundle/datadirect-product-compatibility/resource/datadirect-product-compatibility.pdf>

### Changes Since 6.0.0 GA

- **Enhancements**
  - The driver has been enhanced to comply with FIPS standards for data encryption. As part of this enhancement, the driver was tested with FIPS 140-3 enabled using a Red Hat OpenJDK 21 on a Red Hat Universal Base Image 9 instance. See [FIPS \(Federal Information Processing Standard\)](#) on page 39 for details.
- **Changed behavior**
  - The connection property `SpyAttributes` has been updated to exclude the attribute `load=classname`, which was previously used to load the driver specified by the given class name. See [SpyAttributes](#) on page 58 for details.

### Highlights of the 6.0.0 Release

- Supports SQL read access to the Oracle Sales Cloud. See [Supported SQL functionality](#) on page 65 for details.
- The driver supports JDBC core functions. For details, refer to "JDBC support" in the *Progress DataDirect for JDBC Drivers Reference*.
- Supports Oracle Sales Cloud data types. See [Data types](#) on page 12 and [getTypeInfo\(\)](#) on page 13 for details.
- Supports HTTP proxy. See [Proxy server support](#) on page 38 for details.
- Supports timeout functionality. See [Timeouts](#) on page 42 for details.

## Requirements

The driver is compatible with JDBC 2.0, 3.0, and 4.0.

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

## Driver and DataSource classes

### Driver class

The Driver class is:

```
com.ddtek.jdbc.oraclesalescloud.OracleSalesCloudDriver
```

### DataSource class

The DataSource class is:

```
com.ddtek.jdbcx.oraclesalescloud.OracleSalesCloudDataSource
```

## Connection URL

The connection URL takes the following form:

```
jdbc:datadirect:oraclesalescloud://base_url;User=user_id;  
Password=password[;property=value[;...]]
```

---

### Note:

- Connection property names are case-insensitive. For example, `Password` is the same as `password`.
- For connection properties that support string values, use the following escape sequence to specify values containing leading or trailing spaces and curly brackets: `{value}`. For example: `User={hello }` or `Password={{hello}}`.

---

where:

*base\_url*

specifies the base URL of the Oracle Sales Cloud site to which you are connecting, for example, `mysite.custhelp.com`. This value should not include an internet protocol such as `http://` or `https://`.

*user\_id*

specifies the user name that used to connect to the Oracle Sales Cloud instance.

*password*

specifies the password for the specified `User`.

*property=value*

specifies additional connection property settings.

The following is an example of a connection URL to an Oracle Sales Cloud instance.

```
jdbc:datadirect:oraclsalescloud://mysite.custhelp.com;User=123@abccorp.com;Password=secret
```

## Version string information

The `DatabaseMetaData.getDriverVersion()` method returns a Driver Version string in the format:

```
M.m.s.bbbbb(CXXXX.FYYYYYY.UZZZZZ)
```

where:

*M* is the major version number.

*m* is the minor version number.

*s* is the service pack number.

*bbbbbb* is the driver build number.

*XXXX* is the cloud adapter build number.

*YYYYYY* is the framework build number.

*ZZZZZZ* is the utl build number.

For example:

```
6.0.0.000002(C0003.F000001.U000002)
  |____| |___| |_____| |_____|
  Driver Cloud Frame   Utl
```

## Connection properties and configuration options

The driver includes over 15 connection properties and configuration options. You can use these properties and options to customize the driver for your environment. These properties and options can be used to accomplish different tasks, such as controlling driver functionality and optimizing performance.

See [Using connection properties](#) on page 35 and [Connection property descriptions](#) on page 47 for more information.

## Data types

The following table lists Oracle Sales Cloud data types supported by the driver and how they are mapped to JDBC data types.

See [getTypeInfo\(\)](#) on page 13 for `getTypeInfo()` results of data types supported by the driver.

**Table 1: Oracle Sales Cloud data types**

Oracle Sales Cloud data type	JDBC data type
BOOLEAN	BOOLEAN
DECIMAL	DECIMAL
INTEGER <sup>1</sup>	INTEGER <i>or</i> BIGINT <i>or</i> DECIMAL
LONG	BIGINT
LONGSTRING	LONGVARCHAR
STRING <sup>2</sup>	VARCHAR

## getTypeInfo()

The `DatabaseMetaData.getTypeInfo()` method returns information about data types. The following table provides `getTypeInfo()` results for supported Oracle Sales Cloud data types.

**Table 2: getTypeInfo()**

<b>TYPE_NAME = boolean</b> AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 16 (BOOLEAN) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = BOOLEAN MAXIMUM_SCALE = 0	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 1 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
--	--

<sup>1</sup> When precision is less than or equal to 9, INTEGER is mapped as INTEGER. When precision is greater than 9, INTEGER is mapped as BIGINT. When no precision is specified, INTEGER is mapped as DECIMAL with a precision of 19 and a scale of 4.

<sup>2</sup> When no precision for STRING fields is offered in the metadata, STRING is mapped as VARCHAR with a length of 4000 characters. When precision for STRING columns is available, the precision is maintained and STRING is mapped as VARCHAR.

<p><b>TYPE_NAME = decimal</b></p> <p>AUTO_INCREMENT = false  CASE_SENSITIVE = false  CREATE_PARAMS = NULL  DATA_TYPE = 3 (DECIMAL)  FIXED_PREC_SCALE = false  LITERAL_PREFIX = NULL  LITERAL_SUFFIX = NULL  LOCAL_TYPE_NAME = DECIMAL  MAXIMUM_SCALE = 4</p>	<p>MINIMUM_SCALE = 0  NULLABLE = 1  NUM_PREC_RADIX = 10  PRECISION = 19  SEARCHABLE = 3  SQL_DATA_TYPE = NULL  SQL_DATETIME_SUB = NULL  UNSIGNED_ATTRIBUTE = false</p>
<p><b>TYPE_NAME = integer<sup>3</sup></b></p> <p>AUTO_INCREMENT = false  CASE_SENSITIVE = false  CREATE_PARAMS = NULL  DATA_TYPE = 4 (INTEGER)  FIXED_PREC_SCALE = false  LITERAL_PREFIX = NULL  LITERAL_SUFFIX = NULL  LOCAL_TYPE_NAME = INTEGER  MAXIMUM_SCALE = 0</p>	<p>MINIMUM_SCALE = 0  NULLABLE = 1  NUM_PREC_RADIX = 10  PRECISION = 10  SEARCHABLE = 3  SQL_DATA_TYPE = NULL  SQL_DATETIME_SUB = NULL  UNSIGNED_ATTRIBUTE = false</p>
<p><b>TYPE_NAME = long</b></p> <p>AUTO_INCREMENT = false  CASE_SENSITIVE = false  CREATE_PARAMS = NULL  DATA_TYPE = -5 (BIGINT)  FIXED_PREC_SCALE = false  LITERAL_PREFIX = NULL  LITERAL_SUFFIX = NULL  LOCAL_TYPE_NAME = LONG  MAXIMUM_SCALE = 0</p>	<p>MINIMUM_SCALE = 0  NULLABLE = 1  NUM_PREC_RADIX = 10  PRECISION = 19  SEARCHABLE = 3  SQL_DATA_TYPE = NULL  SQL_DATETIME_SUB = NULL  UNSIGNED_ATTRIBUTE = false</p>

<sup>3</sup> When precision is less than or equal to 9, INTEGER is mapped as INTEGER. When precision is greater than 9, INTEGER is mapped as BIGINT. When no precision is specified, INTEGER is mapped as DECIMAL with a precision of 19 and a scale of 4.

<p><b>TYPE_NAME = longstring</b></p> <p>AUTO_INCREMENT = false  CASE_SENSITIVE = true  CREATE_PARAMS = NULL  DATA_TYPE = -1 (LONGVARCHAR)  FIXED_PREC_SCALE = false  LITERAL_PREFIX = '  LITERAL_SUFFIX = '  LOCAL_TYPE_NAME = LONGVARCHAR  MAXIMUM_SCALE = 0</p>	<p>MINIMUM_SCALE = NULL  NULLABLE = 1  NUM_PREC_RADIX = 10  PRECISION = 2147483647  SEARCHABLE = 0  SQL_DATA_TYPE = NULL  SQL_DATETIME_SUB = NULL  UNSIGNED_ATTRIBUTE = NULL</p>
<p><b>TYPE_NAME =string<sup>4</sup></b></p> <p>AUTO_INCREMENT = false  CASE_SENSITIVE = true  CREATE_PARAMS = NULL  DATA_TYPE = 12 (VARCHAR)  FIXED_PREC_SCALE = false  LITERAL_PREFIX = '  LITERAL_SUFFIX = '  LOCAL_TYPE_NAME = STRING  MAXIMUM_SCALE = 0</p>	<p>MINIMUM_SCALE = NULL  NULLABLE = 1  NUM_PREC_RADIX = NULL  PRECISION = 4000  SEARCHABLE = 3  SQL_DATA_TYPE = NULL  SQL_DATETIME_SUB = NULL  UNSIGNED_ATTRIBUTE = NULL</p>

## SQL escape sequences

The driver supports the following SQL escape sequences.

- Date, Time, and Timestamp Escape Sequences
- Scalar Functions
- Outer Join Escape Sequences
- LIKE Escape Character Sequence for Wildcards

Refer to "SQL escape sequences" in the *Progress DataDirect for JDBC Drivers Reference* for information about SQL escape sequences.

<sup>4</sup> When no precision for `STRING` fields is offered in the metadata, `STRING` is mapped as `VARCHAR` with a length of 4000 characters. When precision for `STRING` columns is available, the precision is maintained and `STRING` is mapped as `VARCHAR`.

## Supported scalar functions

You can use scalar functions in SQL statements with the following syntax:

```
{fn scalar-function}
```

where:

*scalar-function*

is a scalar function supported by the drivers, as listed in the following table.

### Example:

```
SELECT id, name FROM emp WHERE name LIKE {fn UCASE('Smith')}
```

Refer to "Scalar functions" in the *Progress DataDirect for JDBC Drivers Reference* for more information.

**Table 3: Scalar functions**

String Functions	Numeric Functions	Timedate Functions	System Functions
ASCII	ABS	CURDATE	CURSESSIONID
BIT_LENGTH	ACOS	CURTIME	DATABASE
CHAR	ASIN	DATEDIFF	IDENTITY
CHAR_LENGTH	ATAN	DAY	USER
CHARACTER_LENGTH	ATAN2	DAYNAME	IFNULL
CONCAT	CEILING	DAYOFMONTH	
DIFFERENCE	BITAND	DAYOFWEEK	
HEXTORAW	BITOR	DAYOFYEAR	
INSERT	BITXOR	EXTRACT	
LCASE	COS	hour	
LEFT	COT	MINUTE	
LENGTH	DEGREES	MONTH	
LOCATE	EXP	MONTHNAME	
LOCATE_2	FLOOR	NOW	
LOWER	LOG	SECOND	
LTRIM	LOG10	TO_CHAR	
OCTET_LENGTH	MOD	WEEK	

String Functions	Numeric Functions	Timedate Functions	System Functions
RAWTOHEX	PI	YEAR	
REPEAT	POWER		
REPLACE	RADIANS		
RIGHT	RAND		
RTRIM	ROUND		
SOUNDEX	SIGN		
SPACE	SIN		
SUBSTR	SQRT		
SUBSTRING	TAN		
UCASE	TRUNCATE		
UPPER	ROUNDMAGIC		

## Mapping objects to tables

The driver automatically maps Oracle Sales Cloud objects and fields to tables and columns when it first connects to an Oracle Sales Cloud instance. The driver maps standard as well as custom objects. The driver uses a local schema map to instantiate the mapping of the remote data source objects to tables and the metadata associated with those tables. The driver creates a schema map for each user. By default, the schema map is created in the directory from which the application is run. The CreateMap connection property allows you to update or re-create the schema map. The schema map uses the user ID specified for the connection as the name of the schema map configuration 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 configuration file. You can set the SchemaMap connection property to override the default setting for the name and location of the database.

Note that the Refresh Map SQL extension can also be used to update the relational map of your data.

### See also

[CreateMap](#) on page 50

[SchemaMap](#) on page 55

[Refresh Map \(EXT\)](#) on page 67

## DataDirect tools

Progress DataDirect for JDBC drivers install the set of tools described in this section. For detailed instructions on using these tools, refer to the corresponding topics in the *Progress DataDirect for JDBC Drivers Reference*.

- DataDirect Test allows you to test your JDBC driver and learn the JDBC API.
- DataDirect Connection Pool Manager allows you to pool connections when accessing databases. When your applications use connection pooling, connections are reused rather than created each time a connection is requested. Because establishing a connection is among the most costly operations an application may perform, using Connection Pool Manager to implement connection pooling can significantly improve performance.
- Statement Pool Monitor loads statements into and remove statements from the statement pool as well as generate information to help you troubleshoot statement pooling performance.
- DataDirect Spy logs detailed information about calls your driver makes that can be used for troubleshooting.

## Troubleshooting

The *Progress DataDirect for JDBC Drivers Reference* provides information on troubleshooting problems should they occur. Refer to the "Troubleshooting" section in the *Reference* for details.

## Additional information

In addition to the content provided in this guide, the documentation set also contains detailed conceptual and reference information that applies to all the drivers. For more information in these topics, refer the *Progress DataDirect for JDBC Drivers Reference* or use the links below to view some common topics:

- "JDBC support" describes support for JDBC interfaces and methods for the Progress DataDirect for JDBC drivers.
- "JDBC extensions" describes the JDBC extensions provided by the `com.ddtek.jdbc.extensions` package.
- "SQL escape sequences for JDBC" provides an overview of SQL escape sequences for JDBC. In addition, it documents the scalar functions that you use in SQL statements.
- "Security best practices for JDBC applications" describes the security best practices you should employ when developing and deploying your application with the driver.

## 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

---

After the driver has been installed and defined on your class path, you can connect from your application to your database in either of the following ways.

- Using the JDBC `DriverManager`, by specifying the connection URL in the `DriverManager.getConnection()` method.
- Creating a JDBC `DataSource` that can be accessed through the Java Naming Directory Interface (JNDI).

For details, see the following topics:

- [Driver and DataSource classes](#)
- [Connecting using the DriverManager](#)
- [Connecting using data sources](#)

## Driver and DataSource classes

### Driver class

The `Driver` class is:

```
com.ddtek.jdbc.oraclesalescloud.OracleSalesCloudDriver
```

### DataSource class

The `DataSource` class is:

```
com.ddtek.jdbcx.oraclesalescloud.OracleSalesCloudDataSource
```

## Connecting using the DriverManager

You can connect to an Oracle Sales Cloud instance using the JDBC `DriverManager` with the `DriverManager.getConnection()` method. As the following example shows, this method specifies a string containing a connection URL.

```
Connection conn = DriverManager.getConnection  
( "jdbc:datadirect:oraclesalescloud://mysite.custhelp.com;User=123@abccorp.com;Password=secret" );
```

## Setting the class path

The driver must be defined on your class path before you can connect. The class path is the search string your Java Virtual Machine (JVM) uses to locate JDBC drivers on your computer. If the driver is not defined on your class path, you will receive a `class not found` exception when trying to load the driver. Set your system class path to include the `oraclesalescloud.jar` file as shown, where `install_dir` is the path to your product installation directory.

```
install_dir/lib/60/oraclesalescloud.jar
```

### Windows Example

```
CLASSPATH=.;C:\Program Files\Progress\DataDirect\JDBC\lib\60\oraclesalescloud.jar
```

### UNIX Example

```
CLASSPATH=./opt/Progress/DataDirect/JDBC/lib/60/oraclesalescloud.jar
```

## Passing the connection URL

After setting the class path, the required connection information needs to be passed in the form of a connection URL. The connection URL takes the form:

```
jdbc:datadirect:oraclesalescloud://base_url;User=user_id;  
Password=password[;property=value[;...]]
```

where:

*base\_url*

specifies the base URL of the Oracle Sales Cloud site to which you are connecting, for example, `mysite.custhelp.com`. This value should not include an internet protocol such as `http://` or `https://`.

*user\_id*

specifies the user name that used to connect to the Oracle Sales Cloud instance.

*password*

specifies the password for the specified `User`.

*property=value*

specifies additional connection property settings.

The following is an example of a connection URL to an Oracle Sales Cloud instance.

```
jdbc:datadirect:oraclesalescloud://mysite.custhelp.com;User=123@abccorp.com;Password=secret
```

## Testing the connection

You can also use DataDirect Test™ to establish and test a `DriverManager` connection. The screen shots in this section were taken on a Windows system.

**Take the following steps to establish a connection.**

1. Navigate to the installation directory. The default location is:

- Windows systems: `Program Files\Progress\DataDirect\JDBC\testforjdbc`
- UNIX and Linux systems: `/opt/Progress/DataDirect/JDBC/testforjdbc`

---

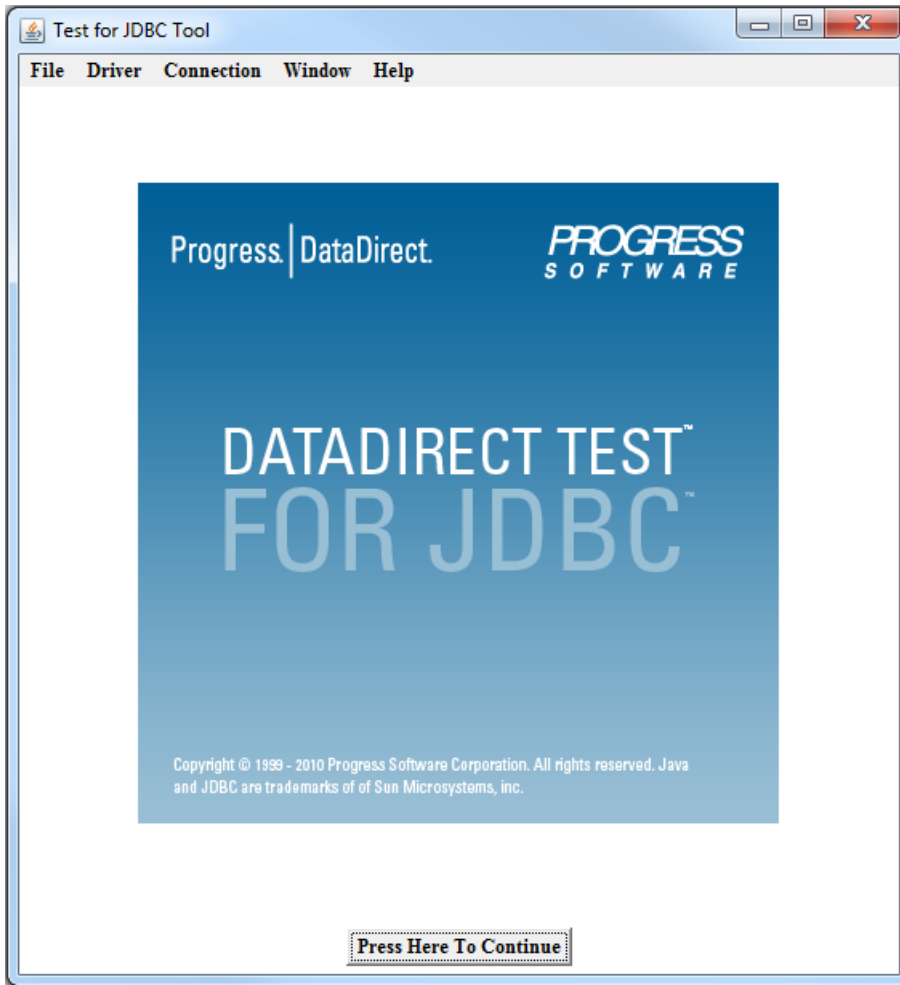
**Note:** For UNIX/Linux, if you do not have access to `/opt`, your home directory will be used in its place.

---

2. From the `testforjdbc` folder, run the platform-specific tool:

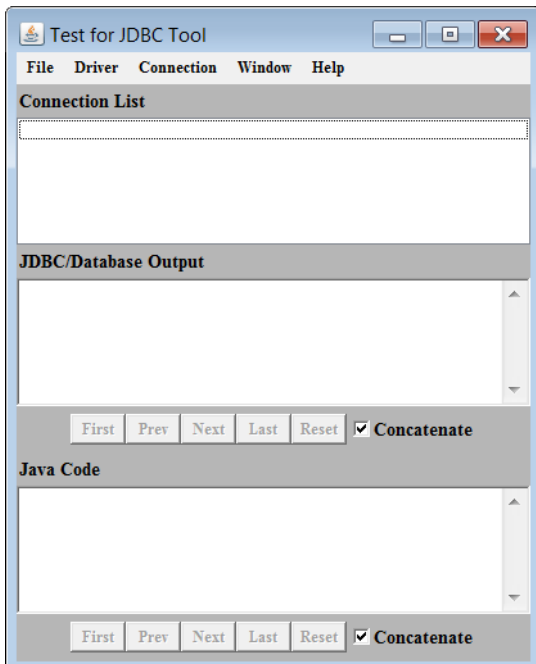
- `testforjdbc.bat` (on Windows systems)
- `testforjdbc.sh` (on UNIX and Linux systems)

The **Test for JDBC Tool** window appears:



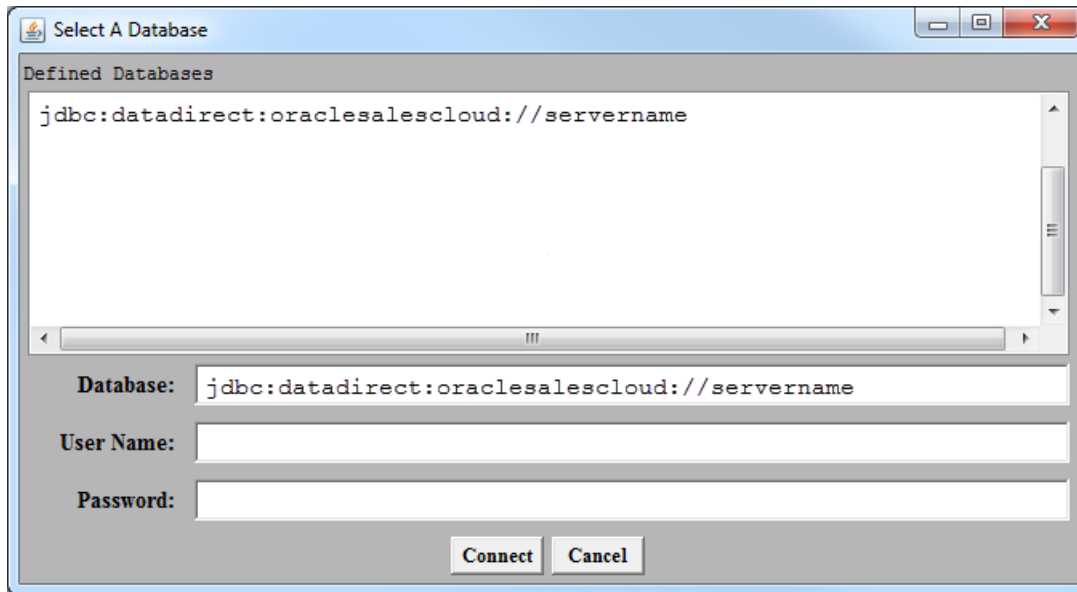
3. Click **Press Here to Continue**.

The main dialog appears:



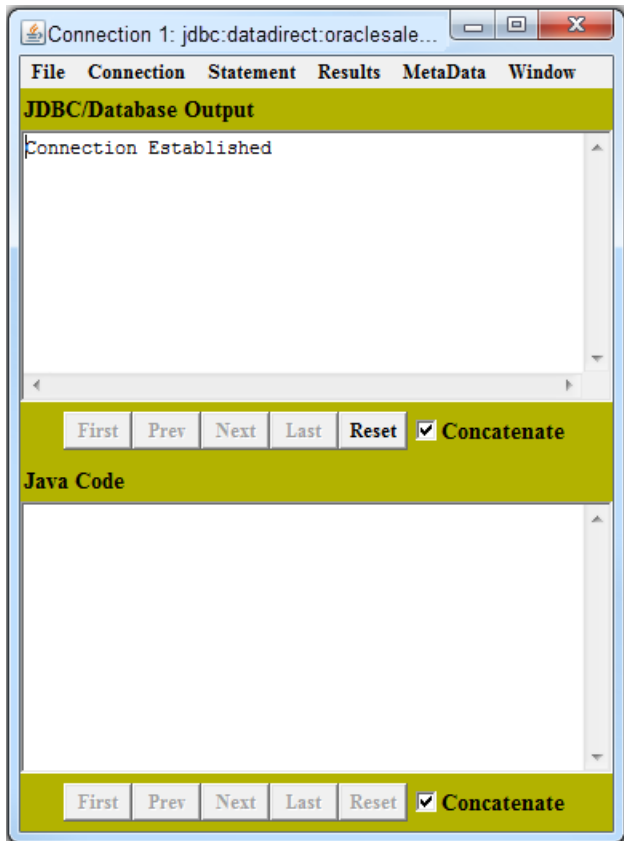
- From the menu bar, select **Connection > Connect to DB**.

The **Select A Database** dialog appears:



- Select the appropriate database template from the **Defined Databases** field.
- In the **Database** field, specify the base URL of the Oracle Sales Cloud site to which you are connecting. The base URL replaces `servername` in the sample string. For example:  
`jdbc:datadirect:oraclesalescloud://mysite.custhelp.com`
- If you are using user ID/password authentication, enter your user ID and password in the corresponding fields.
- Click **Connect**.

If the connection information is entered correctly, the **JDBC/Database Output** window reports that a connection has been established. (If a connection is not established, the window reports an error.)



Refer to "DataDirect Test" in the *Progress DataDirect for JDBC Drivers Reference* for more information about using DataDirect Test.

## Connecting using data sources

A JDBC `DataSource` is a Java object that defines connection information required for a JDBC driver to connect to the database. Each JDBC driver vendor provides their own data source implementation for this purpose. A Progress DataDirect data source is Progress DataDirect's implementation of a `DataSource` object that provides the connection information needed for the driver to connect to a database.

Because data sources work with the Java Naming Directory Interface (JNDI) naming service, data sources can be created and managed separately from the applications that use them. Because the connection information is defined outside of the application, the effort to reconfigure your infrastructure when a change is made is minimized. For example, if the database is moved to another database server, the administrator need only change the relevant properties of the `DataSource` object. The applications using the database do not need to change because they only refer to the name of the data source.

## How data sources are implemented

Data sources are implemented through a `DataSource` class. A `DataSource` class implements the `javax.sql.DataSource` interface.

## See also

[Driver and DataSource classes](#) on page 11

## Creating data sources

The following example files provide details on creating and using Progress DataDirect data sources with the Java Naming Directory Interface (JNDI), where *install\_dir* is the product installation directory.

- *install\_dir/Examples/JNDI/JNDI\_LDAP\_Example.java* can be used to create a JDBC `DataSource` and save it in your LDAP directory using the JNDI Provider for LDAP.
- *install\_dir/Examples/JNDI/JNDI\_FILESYSTEM\_Example.java* can be used to create a JDBC `DataSource` and save it in your local file system using the File System JNDI Provider.

To connect using a JNDI `DataSource`, the driver needs to access a JNDI data store to persist the data source information. For a JNDI file system implementation, you must download the File System Service Provider from the [Oracle Technology Network Java SE Support downloads page](#), unzip the files to an appropriate location, and add the `fscontext.jar` and `providerutil.jar` files to your class path. These steps are not required for LDAP implementations because the LDAP Service Provider has been included with Java SE since Java 2 SDK, v1.3.

## Calling a DataSource in an application

Applications can call a Progress DataDirect `DataSource` using a logical name to retrieve the `javax.sql.DataSource` object. This object loads the specified driver and can be used to establish a connection to the database.

Once the `DataSource` has been registered with JNDI, it can be used by your JDBC application as shown in the following code example.

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("EmployeeDB");
Connection con = ds.getConnection("domino", "spark");
```

In this example, the JNDI environment is first initialized. Next, the initial naming context is used to find the logical name of the data source (`EmployeeDB`). The `Context.lookup()` method returns a reference to a Java object, which is narrowed to a `javax.sql.DataSource` object. Then, the `DataSource.getConnection()` method is called to establish a connection.



## Using the driver

---

This section provides information on how to connect to your data store using either the JDBC DriverManager or DataDirect JDBC data sources, as well as information on how to implement and use functionality supported by the driver.

For details, see the following topics:

- [Connecting from an application](#)
- [Using connection properties](#)
- [Proxy server support](#)
- [Unicode support](#)
- [Data encryption](#)
- [Catalog Tables](#)
- [Identifiers](#)
- [Timeouts](#)
- [Scrollable cursors](#)
- [Parameter metadata support](#)
- [ResultSet metadata support](#)
- [Error handling](#)
- [Rowset support](#)

## Connecting from an application

Once the driver is installed and configured, you can connect from your application to your database in either of the following ways:

- Using the JDBC `DriverManager`, by specifying the connection URL in the `DriverManager.getConnection()` method.
- Creating a JDBC data source that can be accessed through the Java Naming Directory Interface (JNDI).

## Driver and DataSource classes

### Driver class

The `Driver` class is:

```
com.ddtek.jdbc.oraclesalescloud.OracleSalesCloudDriver
```

### DataSource class

The `DataSource` class is:

```
com.ddtek.jdbcx.oraclesalescloud.OracleSalesCloudDataSource
```

## Connecting using the DriverManager

You can connect to an Oracle Sales Cloud instance using the JDBC `DriverManager` with the `DriverManager.getConnection()` method. As the following example shows, this method specifies a string containing a connection URL.

```
Connection conn = DriverManager.getConnection  
( "jdbc:datadirect:oraclesalescloud://mysite.custhelp.com;User=123@abccorp.com;Password=secret" );
```

## Setting the class path

The driver must be defined on your class path before you can connect. The class path is the search string your Java Virtual Machine (JVM) uses to locate JDBC drivers on your computer. If the driver is not defined on your class path, you will receive a `class not found` exception when trying to load the driver. Set your system class path to include the `oraclesalescloud.jar` file as shown, where `install_dir` is the path to your product installation directory.

```
install_dir/lib/60/oraclesalescloud.jar
```

## Windows Example

```
CLASSPATH=.;C:\Program Files\Progress\DataDirect\JDBC\lib\60\oraclesalescloud.jar
```

## UNIX Example

```
CLASSPATH=.: /opt/Progress/DataDirect/JDBC/lib/60/oraclesalescloud.jar
```

## Passing the connection URL

After setting the class path, the required connection information needs to be passed in the form of a connection URL. The connection URL takes the form:

```
jdbc:datadirect:oraclesalescloud://base_url;User=user_id;
  Password=password[;property=value[;...]]
```

where:

*base\_url*

specifies the base URL of the Oracle Sales Cloud site to which you are connecting, for example, `mysite.custhelp.com`. This value should not include an internet protocol such as `http://` or `https://`.

*user\_id*

specifies the user name that used to connect to the Oracle Sales Cloud instance.

*password*

specifies the password for the specified `User`.

*property=value*

specifies additional connection property settings.

The following is an example of a connection URL to an Oracle Sales Cloud instance.

```
jdbc:datadirect:oraclesalescloud://mysite.custhelp.com;User=123@abccorp.com;Password=secret
```

## Testing the connection

You can also use DataDirect Test™ to establish and test a `DriverManager` connection. The screen shots in this section were taken on a Windows system.

**Take the following steps to establish a connection.**

1. Navigate to the installation directory. The default location is:

- Windows systems: `Program Files\Progress\DataDirect\JDBC\testforjdbc`
- UNIX and Linux systems: `/opt/Progress/DataDirect/JDBC/testforjdbc`

---

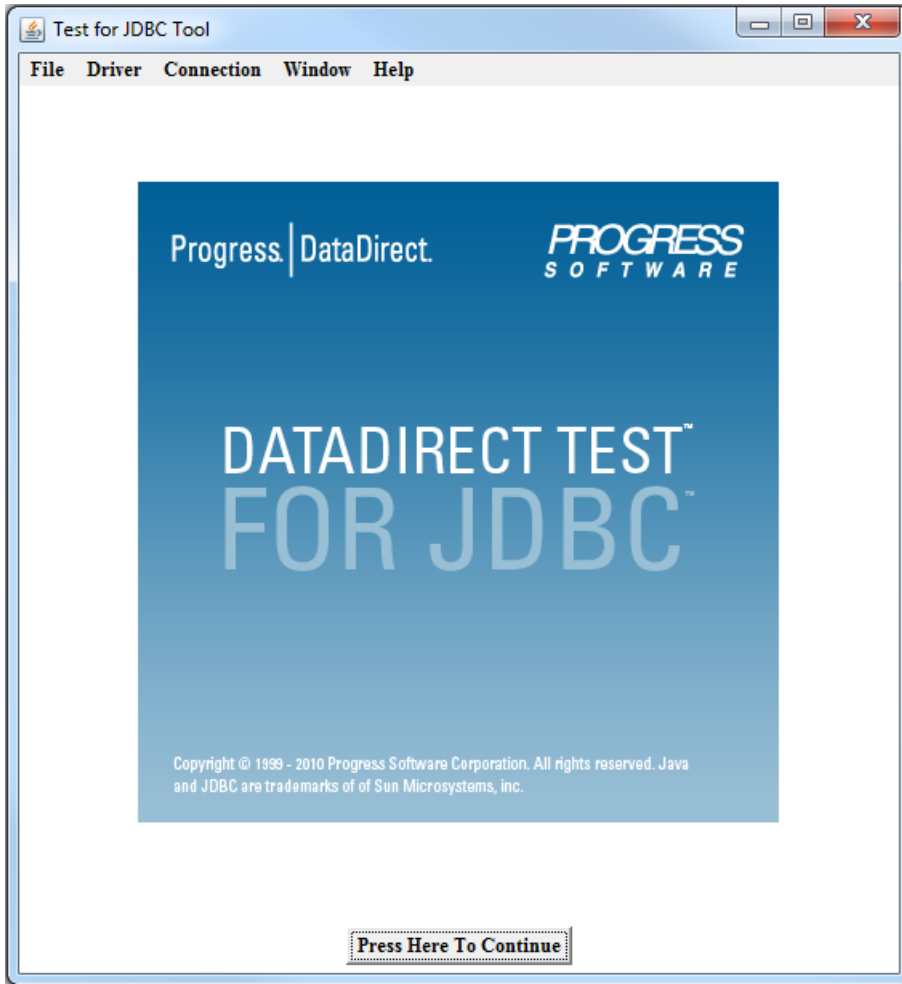
**Note:** For UNIX/Linux, if you do not have access to `/opt`, your home directory will be used in its place.

---

2. From the `testforjdbc` folder, run the platform-specific tool:

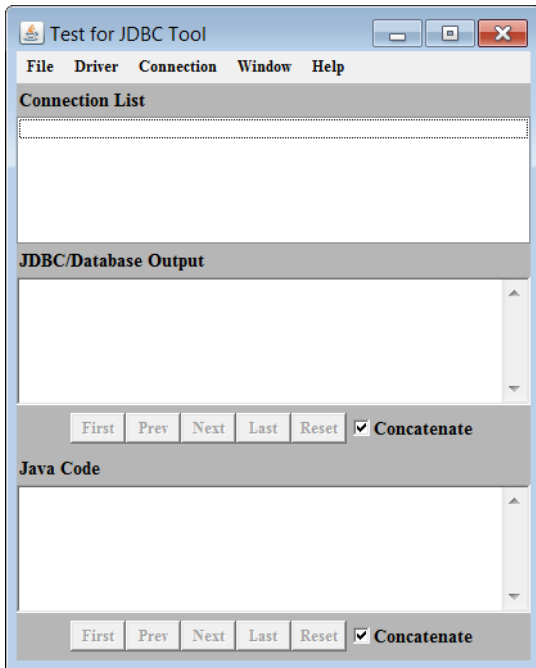
- `testforjdbc.bat` (on Windows systems)
- `testforjdbc.sh` (on UNIX and Linux systems)

The **Test for JDBC Tool** window appears:



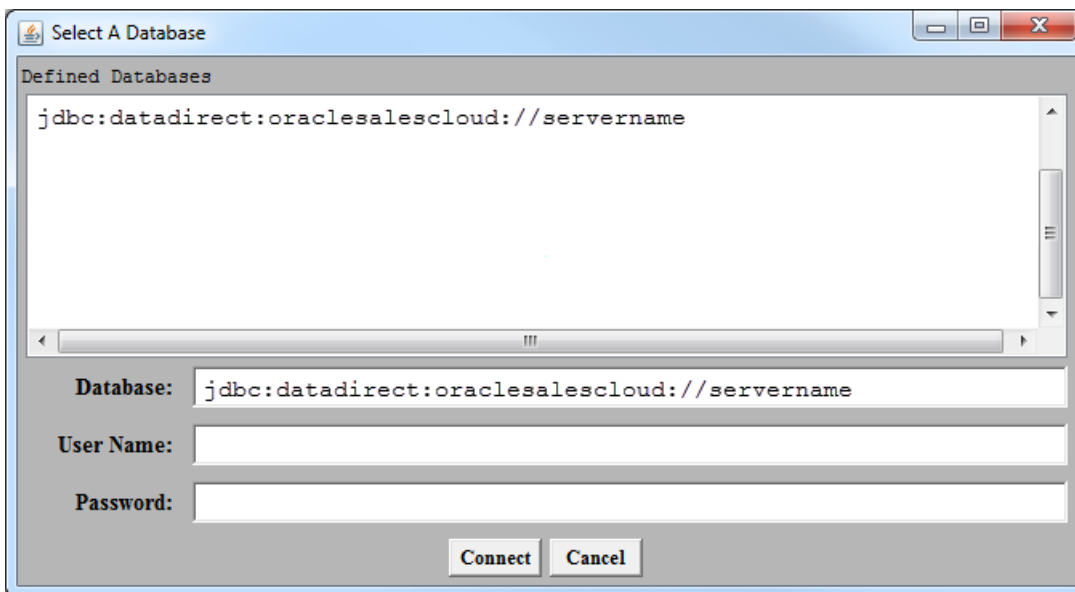
3. Click **Press Here to Continue**.

The main dialog appears:



- From the menu bar, select **Connection > Connect to DB**.

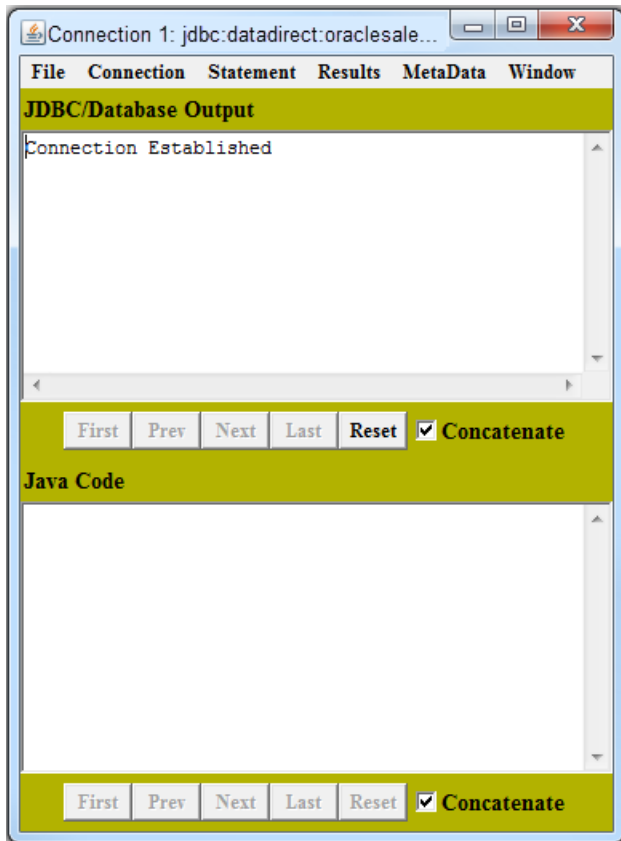
The **Select A Database** dialog appears:



- Select the appropriate database template from the **Defined Databases** field.
- In the **Database** field, specify the base URL of the Oracle Sales Cloud site to which you are connecting. The base URL replaces `servername` in the sample string. For example:  

```
jdbc:datadirect:oraclesalescloud://mysite.custhelp.com
```
- If you are using user ID/password authentication, enter your user ID and password in the corresponding fields.
- Click **Connect**.

If the connection information is entered correctly, the **JDBC/Database Output** window reports that a connection has been established. (If a connection is not established, the window reports an error.)



Refer to "DataDirect Test" in the *Progress DataDirect for JDBC Drivers Reference* for more information about using DataDirect Test.

## Connecting using data sources

A JDBC `DataSource` is a Java object that defines connection information required for a JDBC driver to connect to the database. Each JDBC driver vendor provides their own data source implementation for this purpose. A Progress DataDirect data source is Progress DataDirect's implementation of a `DataSource` object that provides the connection information needed for the driver to connect to a database.

Because data sources work with the Java Naming Directory Interface (JNDI) naming service, data sources can be created and managed separately from the applications that use them. Because the connection information is defined outside of the application, the effort to reconfigure your infrastructure when a change is made is minimized. For example, if the database is moved to another database server, the administrator need only change the relevant properties of the `DataSource` object. The applications using the database do not need to change because they only refer to the name of the data source.

## How data sources are implemented

Data sources are implemented through a `DataSource` class. A `DataSource` class implements the `javax.sql.DataSource` interface.

## See also

[Driver and DataSource classes](#) on page 11

## Creating data sources

The following example files provide details on creating and using Progress DataDirect data sources with the Java Naming Directory Interface (JNDI), where *install\_dir* is the product installation directory.

- *install\_dir/Examples/JNDI/JNDI\_LDAP\_Example.java* can be used to create a JDBC `DataSource` and save it in your LDAP directory using the JNDI Provider for LDAP.
- *install\_dir/Examples/JNDI/JNDI\_FILESYSTEM\_Example.java* can be used to create a JDBC `DataSource` and save it in your local file system using the File System JNDI Provider.

To connect using a JNDI `DataSource`, the driver needs to access a JNDI data store to persist the data source information. For a JNDI file system implementation, you must download the File System Service Provider from the [Oracle Technology Network Java SE Support downloads page](#), unzip the files to an appropriate location, and add the `fscontext.jar` and `providerutil.jar` files to your class path. These steps are not required for LDAP implementations because the LDAP Service Provider has been included with Java SE since Java 2 SDK, v1.3.

## Calling a DataSource in an application

Applications can call a Progress DataDirect `DataSource` using a logical name to retrieve the `javax.sql.DataSource` object. This object loads the specified driver and can be used to establish a connection to the database.

Once the `DataSource` has been registered with JNDI, it can be used by your JDBC application as shown in the following code example.

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("EmployeeDB");
Connection con = ds.getConnection("domino", "spark");
```

In this example, the JNDI environment is first initialized. Next, the initial naming context is used to find the logical name of the data source (`EmployeeDB`). The `Context.lookup()` method returns a reference to a Java object, which is narrowed to a `javax.sql.DataSource` object. Then, the `DataSource.getConnection()` method is called to establish a connection.

# Using connection properties

You can use connection properties to customize the driver for your environment. This section organizes connection properties according to functionality. You can use connection properties with either the JDBC `DriverManager` or a JDBC `DataSource`. For a `DriverManager` connection, a property is expressed as a key value pair and takes the form *property=value*. For a `DataSource` connection, a property is expressed as a JDBC method and takes the form *setProperty(value)*. Connection property names are case-insensitive. For example, `Password` is the same as `password`.

---

**Note:** In a JDBC `DataSource`, string values must be enclosed in double quotation marks, for example, `setUser("123@abccorp.com")`.

---

See [Connection property descriptions](#) on page 47 for an alphabetical list of connection properties and their descriptions.

## Required properties

The following table summarizes connection properties which are required to connect to an Oracle Sales Cloud instance.

**Table 4: Required properties**

Property	Characteristic
<a href="#">Password</a> on page 52	Specifies the password used to connect to your Oracle Sales Cloud instance for user ID/password authentication.
<a href="#">ServerName</a> on page 57	Specifies the base URL of the Oracle Sales Cloud instance to which the driver is connecting.
<a href="#">User</a> on page 60	Specifies the user ID used to connect to your Oracle Sales Cloud instance for user ID/password authentication.

### See also

[Connection property descriptions](#) on page 47

## Mapping properties

The following table summarizes connection properties which are used for the relational mapping of Oracle Sales Cloud data.

**Table 5: Required properties**

Property	Characteristic
<a href="#">ConfigOptions</a> on page 49	Allows you to specify which version of the Oracle Sales Cloud API is being used in your environment with the APIVersion configuration option.
<a href="#">CreateMap</a> on page 50	Determines whether the driver creates the internal files required for a relational map of the native data when establishing a connection.
<a href="#">SchemaMap</a> on page 55	Specifies the name and location of the configuration file used to create the relational map of native data. The driver looks for this file when connecting to an Oracle Sales Cloud instance. By default, if the file does not exist, the driver creates one.

### See also

[Mapping objects to tables](#) on page 17

[Connection property descriptions](#) on page 47

## Web service properties

The following table summarizes web service properties.

**Table 6: Web service properties**

Property	Characteristic
<a href="#">WSFetchSize</a> on page 61	Specifies the maximum number of rows of data the driver can attempt to fetch for each call. The default is 100 (rows).
<a href="#">WSRetryCount</a> on page 61	Specifies 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 connection property. The default is 0.
<a href="#">WSTimeout</a> on page 62	Specifies the time, in seconds, that the driver waits for a response to a web service request. The default is 120 seconds.

### See also

[Connection property descriptions](#) on page 47

## Proxy server properties

The following table summarizes connection properties used in the implementation of a proxy server.

**Table 7: Proxy server properties**

Property	Characteristic
<a href="#">ProxyHost</a> on page 52	Specifies a proxy server.
<a href="#">ProxyPassword</a> on page 53	Specifies the password needed to connect to a proxy server.
<a href="#">ProxyPort</a> on page 54	Specifies the port number where the proxy server is listening for HTTP or HTTPS requests.
<a href="#">ProxyUser</a> on page 54	Specifies the user name needed to connect to a proxy server.

### See also

[Connection property descriptions](#) on page 47

## Additional properties

The following table summarizes additional connection properties.

Table 8: Additional properties

Property	Characteristic
<a href="#">LogConfigFile</a> on page 51	Specifies the filename of the configuration file used to initialize driver logging. The default filename is <code>ddlogging.properties</code> .
<a href="#">SpyAttributes</a> on page 58	Enables DataDirect Spy to log detailed information about calls that are issued by the driver on behalf of the application. DataDirect Spy is not enabled by default.

**See also**

[Connection property descriptions](#) on page 47

## Proxy server support

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

The driver supports proxy server connections with the `ProxyHost`, `ProxyPort`, `ProxyUser`, and `ProxyPassword` connection properties. See [Using connection properties](#) on page 35 and [Proxy server properties](#) on page 37 for details.

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 8888, and provides a user ID and password for authentication:

```
java -Dhttp.proxyHost=pserver -Dhttp.proxyPort=8888 -Dhttp.proxyUser=smith
-Dhttp.proxyPassword=secret -cp oraclesalescloud.jar com.acme.myapp.Main
```

## Unicode support

Multilingual JDBC applications can be developed on any operating system using the driver to access both Unicode and non-Unicode enabled databases. Internally, Java applications use UTF-16 Unicode encoding for string data. When fetching data, the driver automatically performs the conversion from the character encoding used by the database to UTF-16. Similarly, when inserting or updating data in the database, the driver automatically converts UTF-16 encoding to the character encoding used by the database.

The JDBC API provides mechanisms for retrieving and storing character data encoded as Unicode (UTF-16) or ASCII. Additionally, the Java String object contains methods for converting UTF-16 encoding of string data to or from many popular character encodings.

# Data encryption

All communication between the driver and Oracle Sales Cloud, including user ID/password authentication, is encrypted using Secure Sockets Layer (SSL).

---

**Important:** The driver complies with FIPS when FIPS mode is enabled with the client JVM. See "FIPS (Federal Information Processing Standard)" for more information.

---

See [Using connection properties](#) on page 35 for information on specifying a user ID and password.

## FIPS (Federal Information Processing Standard)

The Federal Information Processing Standard (or FIPS) is a cryptography standard created by the U.S. government. FIPS specifications require certain secure algorithms, cryptographic modules, and random number generation. The driver is FIPS compliant for data encryption when FIPS is enabled for the JVM on the client machine.

The following applies when the driver is running in a FIPS environment:

- The driver complies with 140-3 and 140-2 standards.
- The driver uses PKCS #11 providers to access keystores.

The driver was tested with FIPS 140-3 enabled using Red Hat OpenJDK 21 on a Red Hat Universal Base Image 9 instance.

## Catalog Tables

The driver provides a standard set of catalog tables that maintain the information returned by the methods of the JDBC `DatabaseMetaData`, `ParameterMetaData`, and `ResultSetMetaData` interfaces. If possible, use JDBC metadata methods to obtain this information instead of querying the catalog tables directly.

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

## 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 <code>OracleSalesCloud</code> .
INSTANCE	VARCHAR(128)	The value for <code>INSTANCE</code> is null.
VERSION	VARCHAR(30), NOT NULL	The version of the remote data source to which the session is connected. This is the version of the web service API the driver is using to connect to Oracle Sales Cloud.
CONFIG_OPTIONS	LONGVARCHAR, NOT NULL	The configuration options used to define the remote data model to relational data model mapping.
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 <code>WS_CALL_COUNT</code> column can be reset using the <code>ALTER SESSION</code> 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\_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 10: SYSTEM\_SESSIONS catalog table

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. See <a href="#">Supported scalar functions</a> on page 16 for more information about the CURSESSIONID( ) system function.
CONNECTED	DATETIME, NOT NULL	The date and time the session was established.
USERNAME	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	The value for READONLY is True.
MAX_ROWS	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.

## 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 Oracle Sales Cloud 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

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. The following scenarios show how the driver handles timeouts.

### Session timeouts

If the driver receives a session timeout error from a data source, the driver automatically attempts to re-establish a new session. The driver uses the initial server name, 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 `WSTimeout` connection property for fetch requests. Additionally, in a case where requests might fail, you can configure the driver to retry the request a specified number of times by setting the `WSRetryCount` connection property. If all subsequent attempts to retry a request fail, the driver will return an error indicating that the service request timed out and that the subsequent requests failed.

See [Using connection properties](#) on page 35 and [Web service properties](#) on page 37 for details.

## Scrollable cursors

The driver supports scroll-insensitive result sets.

---

**Note:** When the driver cannot support the requested result set type or concurrency, it automatically downgrades the cursor and generates one or more `SQLWarnings` with detailed information.

---

## Parameter metadata support

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 = ?
```

## ResultSet metadata support

If your application requires table name information, the driver can return table name information in ResultSet metadata for Select statements. The Select statements for which ResultSet metadata is returned may contain aliases, joins, and fully qualified names. The following queries are examples of Select statements for which the ResultSetMetaData.getTableName() method returns the correct table name for columns in the Select list:

```
SELECT id, name FROM Employee
SELECT E.id, E.name FROM Employee E
SELECT E.id, E.name AS EmployeeName FROM Employee E
SELECT E.id, E.name, I.location, I.phone FROM Employee E, EmployeeInfo I
WHERE E.id = I.id
```

```
SELECT id, name, location, phone FROM Employee, EmployeeInfo WHERE id = empId
SELECT Employee.id, Employee.name, EmployeeInfo.location, EmployeeInfo.phone
FROM Employee, EmployeeInfo WHERE Employee.id = EmployeeInfo.id
```

The table name returned by the driver for generated columns is an empty string. The following query is an example of a Select statement that returns a result set that contains a generated column (the column named "upper").

```
SELECT E.id, E.name as EmployeeName, {fn UCASE(E.name)} AS upper FROM Employee E
```

The driver also can return catalog name information when the `ResultSetMetaData.getCatalogName()` method is called if the driver can determine that information. For example, for the following statement, the driver returns "test" for the catalog name and "foo" for the table name:

```
SELECT * FROM test.foo
```

The additional processing required to return table name and catalog name information is only performed if the `ResultSetMetaData.getTableName()` or `ResultSetMetaData.getCatalogName()` methods are called.

## Error handling

### SQLExceptions

The driver reports errors to the application by throwing `SQLExceptions`. Each `SQLException` contains the following information:

- Description of the probable cause of the error, prefixed by the component that generated the error
- Native error code (if applicable)
- String containing the XOPEN SQLstate

### Driver Errors

An error generated by the driver has the format shown in the following example:

```
[DataDirect][OracleSalesCloud JDBC Driver]Timeout expired.
```

You may need to check the last JDBC call your application made and refer to the JDBC specification for the recommended action.

### Database Errors

An error generated by the database has the format shown in the following example:

```
[DataDirect][OracleSalesCloud JDBC Driver][OracleSalesCloud]Invalid Object Name.
```

If you need additional information, use the native error code to look up details in your database documentation.

## Rowset support

The driver supports any JSR 114 implementation of the `RowSet` interface, including:

- `CachedRowSets`

- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

Visit <http://www.jcp.org/en/jsr/detail?id=114> for more information about JSR 114.



---

## Connection property descriptions

---

You can use connection properties to customize the driver for your environment. This section lists the connection properties supported by the driver and describes each property. You can use these connection properties with either the JDBC Driver Manager or a JDBC data source. For a Driver Manager connection, a property is expressed as a key value pair and takes the form *property=value*. For a data source connection, a property is expressed as a JDBC method and takes the form *setProperty(value)*.

---

### Note:

- In a JDBC data source, string values must be enclosed in double quotation marks, for example, `setUser("abc@defcorp.com")`.
- The data type listed for each connection property is the Java data type used for the property value in a JDBC data source.
- Connection property names are case-insensitive. For example, `Password` is the same as `password`.
- For connection properties that support string values, use the following escape sequence to specify values containing leading or trailing spaces and curly brackets: `{value}`. For example: `User={hello }` or `Password={{hello}}`.

---

The following table provides a summary of the connection properties supported by the driver, their corresponding data source methods, and their default values.

**Table 11: Driver Properties**

Connection String Property	Data Source Method	Default Value
<a href="#">ConfigOptions</a> on page 49	<code>setConfigOptions</code>	<code>APIVersion=latest</code>

Connection String Property	Data Source Method	Default Value
<a href="#">CreateMap</a> on page 50	setCreateMap	notExist
<a href="#">LogConfigFile</a> on page 51	setLogConfigFile	ddlogging.properties
<a href="#">Password</a> on page 52	setPassword	No default value
<a href="#">ProxyHost</a> on page 52	setProxyHost	No default value
<a href="#">ProxyPassword</a> on page 53	setProxyPassword	No default value
<a href="#">ProxyPort</a> on page 54	setProxyPort	No default value
<a href="#">ProxyUser</a> on page 54	setProxyUser	No default value
<a href="#">SchemaMap</a> on page 55	setSchemaMap	Default value depends on environment
<a href="#">ServerName</a> on page 57	setServerName	No default value
<a href="#">SpyAttributes</a> on page 58	setSpyAttributes	No default value
<a href="#">User</a> on page 60	setUser	No default value
<a href="#">WSFetchSize</a> on page 61	setWSFetchSize	100 (rows)
<a href="#">WSRetryCount</a> on page 61	setWSRetryCount	0
<a href="#">WSTimeout</a> on page 62	setWSTimeout	120 (seconds)

For details, see the following topics:

- [ConfigOptions](#)
- [CreateMap](#)
- [LogConfigFile](#)
- [Password](#)
- [ProxyHost](#)
- [ProxyPassword](#)
- [ProxyPort](#)
- [ProxyUser](#)
- [SchemaMap](#)
- [ServerName](#)
- [SpyAttributes](#)
- [User](#)

- [WSFetchSize](#)
- [WSRetryCount](#)
- [WSTimeout](#)

## ConfigOptions

### Purpose

Allows you to specify which version of the Oracle Sales Cloud API is being used in your environment with the `APIVersion` configuration option.

### Valid Values

```
( "key=value" )
```

where

*key*

is the `APIVersion` configuration option.

*value*

specifies a setting for the `APIVersion` configuration option.

The key value pair must be enclosed in quotation marks and then in parentheses. For example:

```
ConfigOptions=( "APIVersion=11.1.11" )
```

Similarly, when specifying configuration options using a `DataSource` method, key value pairs must be enclosed in quotation marks. For example:

```
setConfigOptions( "APIVersion=11.1.11" )
```

### Data Source Method

```
setConfigOptions
```

### Default

```
APIVersion=latest
```

### Data Type

String

### See also

[Mapping objects to tables](#) on page 17

[Mapping properties](#) on page 36

[CreateMap](#) on page 50

[SchemaMap](#) on page 55

## APIVersion (configuration option)

### Purpose

Identifies the version of Oracle Sales Cloud used in your environment.

### Valid Values

`latest` | `MM.mm.ss`

where

`latest`

is the latest version of Oracle Sales Cloud supported by the driver. When set to `latest`, the driver assumes the latest version of Oracle Sales Cloud is being used. This is the default setting.

`MM.mm.ss`

is the version of Oracle Sales Cloud used in your environment where `MM` is the major version number, `mm` is the minor version number, and `ss` is the service pack number. For example, `11.1.11`.

### Default

`latest`

### Data Type

String

## CreateMap

### Purpose

Determines whether the driver creates the internal files required for a relational map of the native data when establishing a connection.

### Valid Values

`forceNew` | `no` | `notExist`

### Behavior

If set to `forceNew`, the driver deletes the group of internal files specified by SchemaMap and creates a new group of these files at the same location.

If set to `no`, the driver uses the current group of internal files specified by SchemaMap. If the files do not exist, the connection fails.

If set to `notExist`, the driver uses the current group of internal files specified by SchemaMap. If the files do not exist, the driver creates them.

## Notes

- The internal files share the same directory as the schema map's configuration file. This directory is specified with the SchemaMap connection property.
- You can refresh the internal files related to an existing relational view of your data with the SQL extension Refresh Map. Refresh Map runs a discovery against your native data and updates your internal files accordingly.

## Data Source Method

setCreateMap

## Default

notExist

## Data Type

String

## See also

[Mapping objects to tables](#) on page 17

[Mapping properties](#) on page 36

[ConfigOptions](#) on page 49

[SchemaMap](#) on page 55

# LogConfigFile

## Purpose

Specifies the filename of the configuration file used to initialize driver logging. 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 properties file to load to initialize driver logging. If you do not specify a path, the driver looks for this file in the current working directory. If the specified file does not exist, the driver continues searching for an appropriate properties file.

## Data Source Method

setLogConfigFile

## Default

ddlogging.properties

## Data Type

String

## See also

Refer to "Using Java logging" in the *Progress DataDirect for JDBC Drivers Reference*.

# Password

## Description

Specifies the password used to connect to your Oracle Sales Cloud instance for user ID/password authentication.

---

**Important:** Setting the password using a data source is not recommended. The data source persists all properties, including Password, in clear text.

---

## Valid Values

*password*

where:

*password*

is a valid password. The password is case-sensitive.

## Data Source Method

setPassword

## Default

No default value

## Data Type

String

## See also

[Required properties](#) on page 36

[ServerName](#) on page 57

[User](#) on page 60

# ProxyHost

## Description

Identifies a proxy server.

## Valid Values

*server\_name* | *IP\_address*

where:

*server\_name*

is the name of the proxy server, which may be qualified with the domain name.

*IP\_address*

is an IP address, specified in either IPv4 or IPv6 format, or a combination of the two.

## Data Source Method

setProxyHost

## Default

No default value

## Data Type

String

## See also

[Proxy server properties](#) on page 37

[ProxyPassword](#) on page 53

[ProxyPort](#) on page 54

[ProxyUser](#) on page 54

# ProxyPassword

## Purpose

Specifies the password needed to connect to a proxy server.

## Valid Values

*password*

where:

*password*

is a valid password for that server. Contact your system administrator to obtain a valid password.

## Data Source Method

setProxyPassword

## Default

No default value

## Data Type

String

## See also

[Proxy server properties](#) on page 37

[ProxyHost](#) on page 52

[ProxyPort](#) on page 54

[ProxyUser](#) on page 54

# ProxyPort

## Purpose

Specifies the port number where the proxy server is listening for HTTP or HTTPS requests.

## Valid Values

*port*

where:

*port*

is the port number on which the proxy server is listening. Contact your system administrator to obtain the correct port.

## Data Source Method

`setProxyPort`

## Default

0

## Data Type

Int

## See also

[Proxy server properties](#) on page 37

[ProxyHost](#) on page 52

[ProxyPassword](#) on page 53

[ProxyUser](#) on page 54

# ProxyUser

## Purpose

Specifies the user name needed to connect to a proxy server.

## Valid Values

*user\_name*

where:

*user\_name*

is a valid user ID for the proxy server.

## Data Source Method

`setProxyUser`

## Default

No default value

## Data Type

String

## See also

[Proxy server properties](#) on page 37

[ProxyHost](#) on page 52

[ProxyPassword](#) on page 53

[ProxyPort](#) on page 54

# SchemaMap

## Purpose

Specifies the name and location of the configuration file used to create the relational map of native data. The driver looks for this file when connecting to an Oracle Sales Cloud instance. If the file does not exist, the driver creates one.

## Valid Values

*string*

where:

*string*

is the absolute path and filename of the configuration file, including the `.config` extension. For example, if `SchemaMap` is set to a value of

```
C:\Users\Default\AppData\Local\Progress\DataDirect\Oracle_Sales_Cloud_Schema\testuser.config,
```

the driver either creates or looks for the configuration file `testuser.config` in the directory

```
C:\Users\Default\AppData\Local\Progress\DataDirect\Oracle_Sales_Cloud_Schema\.
```

## Notes

- When connecting to a server, the driver looks for the SchemaMap configuration file. If the configuration file does not exist, the driver creates a SchemaMap configuration file using the name and location you have provided. If you do not provide a name and location for a SchemaMap configuration file, the driver creates it 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 relational 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.

## Example

As the following examples show, escapes are needed when specifying SchemaMap for a `DataSource` but are not used when specifying SchemaMap in a `DriverManager` connection URL.

### DriverManager example

```
jdbc:datadirect:oraclesalescloud://mysite.custhelp.com;  
SchemaMap=C:\Users\Default\AppData\Local\Progress\DataDirect  
  \Oracle_Sales_Cloud_Schema\testuser.config
```

### DataSource example

```
OracleSalesCloudDataSource mds = new OracleSalesCloudDataSource();  
mds.setDescription("My OracleSalesCloudDataSource");  
mds.setServerName("mysite.custhelp.com");  
mds.setSchemaMap("C:\\Users\\Default\\AppData\\Local\\Progress  
  \\DataDirect\\Oracle_Sales_Cloud_Schema\\testuser.config")
```

## Data Source Method

`setSchemaMap`

### Default

The default is determined by the environment. The driver attempts to create the files in a subdirectory of the first available directory in the following order:

- Windows
  - DD\_HOME environment variable
  - dd.home system property
  - LOCALAPPDATA environment variable
  - APPDATA environment variable
  - user.home system property

For Windows, the file path takes the following format:

```
available_location\Progress\DataDirect\Oracle_Sales_Cloud_Schema\user_name.config
```

- UNIX/Linux
  - DD\_HOME environment variable
  - dd.home system property
  - user.home system property

For UNIX/Linux, the file path takes the following format:

```
available_location/progress/datadirect/Oracle_Sales_Cloud_schema/user_name.config
```

## Data Type

String

## See also

[Mapping objects to tables](#) on page 17

[Mapping properties](#) on page 36

[ConfigOptions](#) on page 49

[CreateMap](#) on page 50

# ServerName

## Purpose

Specifies the base URL of the Oracle Sales Cloud instance to which the driver is connecting.

## Valid Values

*base\_url*

where:

*base\_url*

is the base URL of the Oracle Sales Cloud instance to which you are connecting, for example, `mysite.custhelp.com`. This value should not include an internet protocol such as `http://` or `https://`.

## Data Source Method

`setServerName`

## Default

No default value

## Data Type

String

## See also

[Required properties](#) on page 36

[Password](#) on page 52

[User](#) on page 60

# SpyAttributes

## Purpose

Enables DataDirect Spy to log detailed information about calls that are issued by the driver on behalf of the application. DataDirect Spy is not enabled by default.

## Valid Values

```
( spy_attribute [ ; spy_attribute ] ... )
```

where:

*spy\_attribute*

is any valid DataDirect Spy attribute.

## Behavior

Attribute	Description
<code>linelimit=<i>numberofchars</i></code>	Sets the maximum number of characters that DataDirect Spy logs on a single line. The default is 0 (no maximum limit).
<code>log=(<i>file</i>)<i>filename</i></code>	Directs logging to the file specified by <i>filename</i> . For Windows, if coding a path to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example: <code>log=(file)C:\\temp\\spy.log;logIS=yes;logITName=yes.</code>

Attribute	Description
<code>log=(filePrefix)file_prefix</code>	<p>Directs logging to a file prefixed by <i>file_prefix</i>. The log file is named <i>file_prefixX.log</i> where:</p> <p><i>X</i> is an integer that increments by 1 for each connection on which the prefix is specified.</p> <p>For example, if the attribute <code>log=(filePrefix)C:\\temp\\spy_</code> is specified on multiple connections, the following logs are created:</p> <pre>C:\temp\spy_1.log C:\temp\spy_2.log C:\temp\spy_3.log ...</pre> <p>If coding a path to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash.</p> <p>For example:  <code>log=(filePrefix)C:\\temp\\spy_;logIS=yes;logTName=yes.</code></p>
<code>log=System.out</code>	<p>Directs logging to the Java output standard, <code>System.out</code>.</p>
<code>logIS= { yes   no   nosingleread }</code>	<p>Specifies whether DataDirect Spy logs activity on <code>InputStream</code> and <code>Reader</code> objects.</p> <p>When <code>logIS=nosingleread</code>, logging on <code>InputStream</code> and <code>Reader</code> objects is active; however, logging of the single-byte read <code>InputStream.read</code> or single-character <code>Reader.read</code> is suppressed to prevent generating large log files that contain single-byte or single character read messages.</p> <p>The default is <code>no</code>.</p>
<code>logLobs= { yes   no }</code>	<p>Specifies whether DataDirect Spy logs activity on <code>BLOB</code> and <code>CLOB</code> objects.</p>
<code>logTName= { yes   no }</code>	<p>Specifies whether DataDirect Spy logs the name of the current thread.</p> <p>The default is <code>no</code>.</p>
<code>timestamp= { yes   no }</code>	<p>Specifies whether a timestamp is included on each line of the DataDirect Spy log.</p> <p>The default is <code>yes</code>.</p>

## Notes

- If coding a path on Windows to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example: `log=(file)C:\\temp\\spy.log`.
- If a log file name does not include the `.log` extension, the driver automatically appends it. For example, a file named `spy.jsp` is renamed to `spy.jsp.log` by the driver.

## Example

The following value instructs the driver to log all JDBC activity to a file using a maximum of 80 characters for each line.

```
(log=(file)/tmp/spy.log;linelimit=80)
```

## Data Source Method

`setSpyAttributes`

## Default

No default value

## Data Type

String

## See also

Refer to "Tracking JDBC Calls with DataDirect Spy" in the *Progress DataDirect for JDBC Drivers Reference* for more information about using DataDirect Spy.

# User

## Purpose

Specifies the user ID used to connect to your Oracle Sales Cloud instance for user ID/password authentication.

## Valid Values

*userid*

where:

*userid*

is a valid user ID used to connect to your Oracle Sales Cloud instance.

## Data Source Method

`setUser`

## Default

No default value

## Data Type

String

## See also

[Required properties](#) on page 36

[Password](#) on page 52

[ServerName](#) on page 57

# WSFetchSize

## Purpose

Specifies the maximum number of rows of data the driver can attempt to fetch for each call.

## Valid Values

x

where

x

is a positive integer from 1 to 100 that defines the maximum number of rows the driver can attempt to fetch for each call.

## Data Source Method

setWSFetchSize

## Default

100 (rows)

## Data Type

Int

## See also

[Web service properties](#) on page 37

[WSRetryCount](#) on page 61

[WSTimeout](#) on page 62

# WSRetryCount

## Description

Specifies the number of times the driver retries a timed-out Select request. The timeout period is specified by the WSTimeout connection property.

## 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.

## Data Source Method

`setWSRetryCount`

## Default

0

## Data Type

Int

## See also

[Web service properties](#) on page 37

[WSFetchSize](#) on page 61

[WSTimeout](#) on page 62

# WSTimeout

## 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  $\infty$ , the driver uses the value as the default timeout for any statement created by the connection. If a Select request times out and `WSRetryCount` is set to retry timed-out requests, the driver retries the request the specified number of times.

### Data Source Method

`setWSTimeout`

### Default

120 (seconds)

### Data Type

Int

### See also

[Web service properties](#) on page 37

[WSFetchSize](#) on page 61

[WSRetryCount](#) on page 61



## Supported SQL functionality

---

The driver provides support for SQL statements and extensions described in this section. SQL extensions are denoted by an (EXT) in the topic title.

For details, see the following topics:

- [Alter Session \(EXT\)](#)
- [Refresh Map \(EXT\)](#)
- [Select](#)
- [SQL expressions](#)
- [Subqueries](#)

### Alter Session (EXT)

#### Purpose

The `Alter Session` statement allows you to change various attributes of a connection session.

#### Syntax

```
ALTER SESSION SET attribute_name=value
```

where:

*attribute\_name*

Specifies the name of the attribute to be changed.

*value*

Refers to the specific value setting for that attribute.

The following table lists session attributes and describes them.

**Table 12: Alter Session Attributes**

Attribute Name	Session Type	Description
Current_Schema	Database	<p>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 session. For example:</p> <pre>ALTER SESSION SET CURRENT_SCHEMA=oraclesalescloud</pre>
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 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 data store 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. For example:</p> <pre>SELECT * FROM information_schema.system_remote_sessions WHERE session_id = cursessionid()</pre>

# 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 configuration file.

## 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](#) on page 68 for a complete explanation.

*from\_clause*

specifies one or more tables on which the other clauses in the query operate. See [From clause](#) on page 70 for a complete explanation.

*where\_clause*

is optional and restricts the results that are returned by the query. See [Where clause](#) on page 72 for a complete explanation.

*groupby\_clause*

is optional and allows query results to be aggregated in terms of groups. See [Group By clause](#) on page 73 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](#) on page 74 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](#) on page 74 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](#) on page 75 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](#) on page 76 for a complete explanation.

*orderby\_clause*

is optional and sorts the results that are returned by the query. See [Order By clause](#) on page 77 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](#) on page 77 for a complete explanation.

## 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 offset number | 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](#) on page 78 for details. *column\_expression* can also include aggregate functions. See [Aggregate functions](#) on page 69 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`.

## 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.

**Table 13: 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	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.
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](#) on page 72 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.

## Outer join escape sequences

### Purpose

The SQL-92 left, right, and full outer join syntax is supported.

### Syntax

```
{oj outer-join}
```

where *outer-join* is

```
table-reference {LEFT | RIGHT | FULL} OUTER JOIN {table-reference | outer-join} ON
search-condition
```

where *table-reference* is a database table name, and *search-condition* is the join condition you want to use for the tables.

```
Example: SELECT Customers.CustID, Customers.Name, Orders.OrderID, Orders.Status FROM {oj
Customers LEFT OUTER JOIN Orders ON Customers.CustID=Orders.CustID} WHERE
Orders.Status='OPEN'
```

The following outer join escape sequences are supported:

- Left outer joins
- Right outer joins
- Full outer joins
- Nested outer joins

## 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
```

### 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](#) on page 85.

## 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

- [SQL expressions](#) on page 78
- [Subqueries](#) on page 85

## 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

- [SQL expressions](#) on page 78
- [Subqueries](#) on page 85

## 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 expr1 rel_operator expr2
```

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](#) on page 78 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 78
- [Subqueries](#) on page 85

## 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 78

## 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
```

# 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 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 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 14: 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'
DECIMAL	<i>n.f</i> where: <i>n</i> is the integral part <i>f</i> is the fractional part	0.25 3.1415 -7.48
DOUBLE	<i>n.fEx</i> where: <i>n</i> is the integral part <i>f</i> is the fractional part <i>x</i> is the exponent	1.2E0 or 2.5E40 or -3.45E2 or 5.67E-4
INTEGER	<i>n</i> where <i>n</i> is a valid integer value in the range of the INTEGER data type	12 or -34 or 0
LONGVARBINARY	X' <i>hex_value</i> '	'000482ff'
LONGVARCHAR	' <i>value</i> '	'This is a string literal'

SQL Type	Literal Syntax	Example
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 and time literals

Date and time literal values are enclosed in single quotation marks ('*value*').

- The format for a Date literal is DATE'*date*'.
- The format for a Time literal is TIME'*time*'.
- The format for a Timestamp literal is TIMESTAMP'*ts*'.

## 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.

### 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 15: Arithmetic operators**

Operator	Purpose	Example
+ -	Denotes a positive or negative expression. These are unary operators.	SELECT * FROM emp WHERE comm = -1
* /	Multiplies, divides. These are binary operators.	UPDATE emp SET sal = sal + sal * 0.10
+ -	Adds, subtracts. These are binary operators.	SELECT sal + comm FROM emp WHERE empno > 100

### Concatenation operator

The concatenation operator manipulates character strings. The following table lists the only supported concatenation operator.

**Table 16: Concatenation operator**

Operator	Purpose	Example
	Concatenates character strings.	SELECT 'Name is'    ename FROM emp

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 driver considers the UNKNOWN result as FALSE.

The following table lists the supported comparison operators.

**Table 17: Comparison operators**

Operator	Purpose	Example
=	Equality test.	SELECT * FROM emp WHERE sal = 1500
!=<>	Inequality test.	SELECT * FROM emp WHERE sal != 1500
><	"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'.
[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

Operator	Purpose	Example
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 18: 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.	SELECT * FROM emp WHERE NOT (job IS NULL) SELECT * FROM emp WHERE NOT (sal BETWEEN 1000 AND 2000)
AND	Returns TRUE if both component conditions are TRUE. Returns FALSE if either is FALSE; otherwise, returns UNKNOWN.	SELECT * FROM emp WHERE job = 'CLERK' AND deptno = 10
OR	Returns TRUE if either component condition is TRUE. Returns FALSE if both are FALSE; otherwise, returns UNKNOWN.	SELECT * FROM emp WHERE job = 'CLERK' OR deptno = 10

### 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 19: Operator precedence**

Precedence	Operator
1	+ (Positive), - (Negative)
2	*(Multiply), / (Division)
3	+ (Add), - (Subtract)
4	(Concatenate)
5	=, >, <, >=, <=, <>, != (Comparison operators)
6	NOT, IN, LIKE
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 driver supports a number of functions that you can use in expressions, as listed and described in [Supported scalar functions](#) on page 16.

## 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 20: 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 ]
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 managename 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)
```