Progress® DataDirect®

Progress DataDirect for JDBC for SAP S/4HANA

User's Guide

Release 6.0.1

Copyright

Table of Contents

1

Welcome to the Progress DataDirect for JDBC for SAP S/4HANA Driver

The Progress® DataDirect® for JDBC™ for SAP S/4HANA™ driver supports SQL read-write access for JDBC applications to SAP ODATA V2 services exposed through the SAP Gateway, including S/4HANA and BW/4HANA. To support SQL access, the driver creates a relational map of the SAP data model and translates SQL statements to OData requests. In addition, the driver employs a SQL engine component that provides support to SQL constructs unavailable in SAP services. This functionality offers a number of advantages, including support for reporting data and metadata in a form that JDBC applications are ready to use.

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-sap-s4hana.

For details, see the following topics:

- CDS views
- 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 46 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 74 for details.

Changes for 6.0.1 GA

- **Enhancements**
  - The driver has been enhanced to support SAP BW/4HANA and other SAP ODATA V2 services exposed through the SAP Gateway.
    - For details, see Supported SQL statements and extensions on page 81.
    - For information on SAP services that use ODATA V2 APIs, refer to the SAP API documentation: ODATA V2.
  - The driver has been enhanced to support CDS views that require parameters. For details, see CDS views on page 25.
  - The driver has been enhanced to support HTTP Header authentication for use with the S/4HANA sandbox hosted at `https://api.sap.com`. See HTTP header authentication (S/4HANA only) on page 45 for details.
  - The driver has been enhanced to support a default value of `100000` and a maximum value of `1000000` for the connection property WSFetchSize on page 80.

- The driver has been enhanced to determine the maximum size of VARCHAR columns within the result set metadata using the new MaxVarcharSize on page 63 connection property.

- **Changed Behavior**
  - The HostName connection property has been renamed ServerName. To support existing configurations, HostName will continue to be supported as an alias for ServerName. See ServerName on page 71 for details.

Highlights of 6.0.0 Release

- The driver supports SQL read-write access to SAP S/4HANA. For details, see Supported SQL statements and extensions on page 81 and Introduction to the SAP S/4HANA data model on page 107.
- The driver supports JDBC core functions. For details, refer to "JDBC support" in the *Progress DataDirect for JDBC Drivers Reference*.
- The driver supports S/4HANA data types through data type inference. See Data types on page 17 and getTypeInfo() on page 18 for details.
- The driver supports custom fields, including many fields added through third-party plug-ins.
- The driver supports Basic authentication. See Basic authentication on page 45 for more details.
- The driver supports the handling of large result sets with paging, and the FetchSize on page 57 and WSFetchSize on page 80 connection properties.
- The driver includes the DataDirect JDBC Driver Configuration Manager for quick configuration and testing of your driver in a web browser. The tool allows you to:
  - Generate and edit connection URLs
  - Test connect your connection URLs
  - Execute SQL commands for testing

  For details, see Generating connection URLs with the Configuration Manager on page 37 and Testing connections and queries on page 38.

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.

Installing and setting up the driver

This section provides you with an overview of the steps required to install and set-up the driver. After completing this procedure, you will be able to begin accessing data with your application.

To begin accessing data with the driver:

1. Install the driver:

a) After downloading the product, unzip the installer files to a temporary directory.

b) From the installer directory, run the appropriate installer file to start the installer.

- **Windows**: `PROGRESS_DATADIRECT_JDBC_INSTALL.exe`
- **Non-Windows**: `PROGRESS_DATADIRECT_JDBC_INSTALL.jar`

c) Follow the prompts to complete installation.

The installer program supports multiple installation methods, including command-line and silent installations. For detailed instructions, refer to the *Progress DataDirect for JDBC Drivers Installation Guide*.

2. Set your system CLASSPATH to include the driver `.jar` file. The CLASSPATH is the search string your Java Virtual Machine (JVM) uses to locate JDBC drivers on your computer. The following examples demonstrate setting the CLASSPATH from a command line using the default installation directory.

- **Windows Example**

```
CLASSPATH=.;C:\Program Files\Progress\DataDirect\JDBC\lib\60\s4hana.jar
```

- **UNIX/LINUX Example**

```
CLASSPATH=.:/opt/Progress/DataDirect/JDBC/lib/60/s4hana.jar
```

Note: The same `.jar` file, `s4hana.jar`, is used for all SAP ODATA V2 services exposed through the SAP Gateway, including S/4HANA and BW/4HANA.

---

3. Configure your driver using one of the following methods:

- **Connection URL**: You can begin using the driver immediately by passing a connection URL with your application or tool. The following examples show how to connect using basic authentication.

  S/4HANA:

```
jdbc:datadirect:s4hana:ServerName=https://mycompany.s4hana.ondemand.com;
User=jsmith;Password=secret;
```

  BW/4HANA:

```
jdbc:datadirect:s4hana:ServerName=https://myinstance.company.com;
User=jsmith;Password=secret;
```

  You can also generate a connection string using the Progress DataDirect SAP S/4HANA Configuration Manager. For details, see Generating connection URLs with the Configuration Manager.
- **Data sources**: The driver also supports connecting using JDBC data sources. A JDBC data source is a Java object, specifically a DataSource object, that defines connection information required for a JDBC driver to connect to the database. See Connecting using data sources for more information.

Note: For most connections, specifying the minimum required connection properties is sufficient to begin accessing data; however, you can provide values for optional properties to use additional supported features and improve performance.

---

4. Set the values for any optional properties that you want to configure. For additional information on optional features and functionality, see the following resources:

- Connection URL Examples provides connection string examples that can be used to configure common functionality and features. You can modify and combine these examples to create a string that best suites your environment.
- Connection property descriptions provides a complete list of supported properties by functionality.
- Performance considerations describes connection properties that affect performance, along with recommended settings.

5. Connect to your service and begin accessing data with your applications, BI tools, database tools, and more. To help you get started, the following resources guide you through accessing data with some common tools:
   - Progress DataDirect SAP S/4Hana Configuration Manager: The SAP S/4HANA Configuration Manager is a browser-based tool that allows you to quickly generate connection URLs, test connections, and execute test queries.
   - DataDirect Test: DataDirect Test allows you to test connect, execute SQL statements, and practice using the JDBC API right out of the box.
   - Tableau: Tableau is a business intelligence software program that allows you to easily create reports and visualized representations of your data.
   - DbVisualizer: DB Visualizer is a database tool that allows you to connect and execute SQL statements against your data.
   - Supported SQL statements and extensions: This section describes the syntax used for SQL statements supported by the driver. You can modify and use the provided examples for your application or tool.

This completes the deployment of the driver.

Driver and DataSource classes

The following are the `Driver` and `DataSource` classes used by the driver:

Driver class:

com.ddtek.jdbc.s4hana.S4HanaDriver

DataSource class:

com.ddtek.jdbcx.s4hana.S4HanaDataSource

Note: The same classes are used for all SAP ODATA V2 services exposed through the SAP Gateway, including S/4HANA and BW/4HANA.

Connection URL examples

After setting the CLASSPATH, the connection information needs to be passed in the form of a connection URL. This section provides examples of connection strings configured to use common features and functionality. You can modify and/or combine these examples to create a connection string for your environment.

Note:

- You can also use the DataDirect Configuration Manager tool to generate and test connection URLs. For more information, see "Generating connection URLs with the Configuration Manager."
- 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}}`.

See also

Generating connection URLs with the Configuration Manager on page 37

Basic authentication example

This string includes the properties used to connect with basic authentication.

```
jdbc:datadirect:s4hana:ServerName=servername;User=user_name;Password=password;
[property=value[;...]];
```

where:

servername

specifies the base URL of the service to which you want to connect. It is comprised of either the domain name or the IP address of the service. For example, for S/4HANA, it can be either `https://mycompany.s4hana.ondemand.com` or `http://123.456.7.8`.

user_name

specifies the user name that is used to connect to the service. For example, `jsmith`.

password

specifies the password used to connect to the service.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

Note: The User and Password properties are not required to be stored in the connection string. They can also be passed separately by the application.

The following example connection strings include the properties required for connecting with basic authentication.

For S/4HANA:

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:s4hana:ServerName=https://mycompany.s4hana.ondemand.com;
   User=jsmith;Password=secret;");
```

For BW/4HANA:

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:s4hana:ServerName=https://myinstance.company.com
   User=jsmith;Password=secret;");
```

See also

Basic authentication on page 45
Connection property descriptions on page 49

HTTP header authentication (S/4HANA only)

This string includes the properties used to connect with HTTP header authentication. HTTP header authentication can only be used to authenticate with the S/4HANA sandbox hosted at `https://api.sap.com`.

```
jdbc:datadirect:s4hana:ServerName=servername;AuthenticationMethod=HttpHeader;AuthHeader=api_key;
SecurityToken=security_token;[property=value[;...]];
```

where:

servername

specifies the URL of the S/4HANA sandbox instance: `https://api.sap.com`.

api_key

specifies the API key used to connect to your SAP S/4HANA service.

security_token

specifies the token used to connect to your SAP S/4HANA service.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

The following example connection string includes the properties required for connecting with HTTP Header authentication.

```
Connection conn = DriverManager.getConnection
    ("jdbc:datadirect:s4hana:ServerName=https://api.sap.com;
    AuthenticationMethod=HttpHeader;AuthHeader=APIKey;
    SecurityToken=du1ZCRNJI0rANDc3pKxJ3AZTX0MxKexK;");
```

See also

HTTP header authentication (S/4HANA only) on page 45
Connection property descriptions on page 49

Proxy server example

This string includes the properties you may need to connect through a proxy server with basic authentication.

```
jdbc:datadirect:s4hana:ServerName=servername;ProxyHost=proxy_host;
ProxyPassword=proxy_password;ProxyPort=proxy_port;ProxyUser=proxy_user;
User=user_name;Password=password;[property=value[;...]];
```

where:

servername

specifies the base URL of the service to which you want to connect. It is comprised of either the domain name or the IP address of the service. For example, for S/4HANA, it can be either `https://mycompany.s4hana.ondemand.com` or `http://123.456.7.8`.

proxy_host

specifies the proxy server to use for the first connection.

proxy_password

specifies the password needed to connect to a proxy server for the first connection.

proxy_port

specifies the port number where the proxy server is listening for requests for the first connection. The default is `0`.

proxy_user

specifies the user name needed to connect to a proxy server for the first connection.

user_name

specifies the user name that is used to connect to the service. For example, `jsmith`.

password

specifies the password used to connect to the service.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

Note: The User and Password properties are not required to be stored in the connection string. They can also be passed separately by the application.

The following example connection strings include the properties required for using a proxy server with basic authentication.

For S/4HANA:

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:s4hana:ServerName=https://mycompany.s4hana.ondemand.com;
    ProxyHost=pserver;ProxyPassword=proxypwd;ProxyPort=808;ProxyUser=johndoe;
    User=jsmith;Password=secret;");
```

For BW/4HANA:

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:s4hana:ServerName=https://myinstance.company.com;
    ProxyHost=pserver;ProxyPassword=proxypwd;ProxyPort=808;ProxyUser=johndoe;
    User=jsmith;Password=secret;");
```

See also

Connection property descriptions on page 49

Data types

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

See "getTypeInfo()" for getTypeInfo() results of data types supported by the driver.

Note: This data type mapping is supported for all SAP ODATA V2 services exposed through the SAP Gateway, including S/4HANA and BW/4HANA.

Table 1: SAP S/4Hana Data Types

SAP S/4Hana Data Type	JDBC Data Type
BINARY	VARBINARY
BOOLEAN	BOOLEAN
BYTE	SMALLINT
DATETIME	TIMESTAMP
DATETIMEOFFSET	TIMESTAMP
DECIMAL	DECIMAL
DOUBLE	DOUBLE
GUID	CHAR
INT16	SMALLINT
INT32	INTEGER
INT64	BIGINT
SBYTE	TINYINT
SINGLE	REAL

SAP S/4Hana Data Type	JDBC Data Type
STRING	VARCHAR
TIMEOFDAY	TIME

getTypeInfo()

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

Table 2: getTypeInfo() Results

TYPE_NAME = BINARY AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = -3 (VARBINARY) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = X' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = BINARY MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 3 SQL_DATA_TYPE = 61 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = BOOLEAN 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 = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 1 SEARCHABLE = 3 SQL_DATA_TYPE = 16 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = BYTE AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 5 (SMALLINT) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = BYTE MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 5 SEARCHABLE = 3 SQL_DATA_TYPE = 5 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = FALSE
TYPE_NAME = DATETIME AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = TIMESTAMP ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = DATETIME MAXIMUM_SCALE = 6	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 26 SEARCHABLE = 3 SQL_DATA_TYPE = 93 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = DATETIMEOFFSET AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = TIMESTAMP ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = DATETIMEOFFSET MAXIMUM_SCALE = 6	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 26 SEARCHABLE = 3 SQL_DATA_TYPE = 93 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = DECIMAL 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 = 38	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 38 SEARCHABLE = 3 SQL_DATA_TYPE = 3 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = FALSE
TYPE_NAME = DOUBLE AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 8 (DOUBLE) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = DOUBLE MAXIMUM_SCALE = 15	MINIMUM_SCALE = -324 NULLABLE = 1 NUM_PREC_RADIX = 2 PRECISION = 15 SEARCHABLE = 3 SQL_DATA_TYPE = 8 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = FALSE
TYPE_NAME = GUID AUTO_INCREMENT = FALSE CASE_SENSITIVE = TRUE CREATE_PARAMS = NULL DATA_TYPE = 1 (CHAR) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = GUID MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 36 SEARCHABLE = 3 SQL_DATA_TYPE = 1 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = INT16 AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 5 (SMALLINT) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = INT16 MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 5 SEARCHABLE = 3 SQL_DATA_TYPE = 5 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = FALSE
TYPE_NAME = INT32 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 = INT32 MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 10 SEARCHABLE = 3 SQL_DATA_TYPE = 4 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = FALSE
TYPE_NAME = INT64 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 = INT64 MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 19 SEARCHABLE = 3 SQL_DATA_TYPE = 25 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = FALSE

TYPE_NAME = SBYTE AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = -6 (TINYINT) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = SBYTE MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 3 SEARCHABLE = 3 SQL_DATA_TYPE = -6 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = FALSE
TYPE_NAME = SINGLE AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 7 (REAL) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = SINGLE MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 2 PRECISION = 7 SEARCHABLE = 3 SQL_DATA_TYPE = 7 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = FALSE

TYPE_NAME = STRING AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = STRING MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 3 SQL_DATA_TYPE = 12 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = TIMEOFDAY AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 92 (TIME) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = TIME ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = TIMEOFDAY MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 8 SEARCHABLE = 3 SQL_DATA_TYPE = 92 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

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.

Supported scalar functions

The driver supports the scalar functions in the following table. Note that your database system may not support all these functions. Refer to the documentation for your database system to find out which functions are supported by your database.

In addition, you can also determine the supported scalar functions by using DatabaseMetaData methods.

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: Supported Scalar Functions

String Functions	Numeric Functions	Timedate Functions	System Functions
ASCII	ABS	CURDATE	CURSESSIONID
BIT_LENGTH	ACOS	CURRENT_DATE	DATABASE
CHAR	ASIN	CURRENT_TIME	IDENTITY
CHAR_LENGTH	ATAN	CURRENT_TIMESTAMP	USER
CHARACTER_LENGTH	ATAN2	CURTIME	
CONCAT	BITAND	DATEDIFF	
DIFFERENCE	BITOR	DATE_ADD	
HEXTORAW	BITXOR	DATE_SUB	
INSERT	CEILING	DAY	
LCASE	COS	DAYNAME	
LEFT	COT	DAYOFMONTH	
LENGTH	DEGREES	DAYOFWEEK	
LOCATE	EXP	DAYOFYEAR	
LOCATE_2	FLOOR	EXTRACT	

String Functions	Numeric Functions	Timedate Functions	System Functions
LOWER	LOG	HOUR	
LTRIM	LOG10	MINUTE	
OCTET_LENGTH	MOD	MONTH	
RAWTOHEX	PI	MONTHNAME	
REPEAT	POWER	NOW	
REPLACE	RADIANS	QUARTER	
RIGHT	RAND	SECOND	
RTRIM	ROUND	SECONDS_SINCE_MIDNIGHT	
SOUNDEX	ROUNDMAGIC	TIMESTAMPADD	
SPACE	SIGN	TIMESTAMPDIFF	
SUBSTR	SIN	TO_CHAR	
SUBSTRING	SQRT	WEEK	
UCASE	TAN	YEAR	
UPPER	TRUNCATE		

CDS views

The driver supports CDS views and CDS views that require parameters. The driver exposes CDS views as tables in a schema, as it does with other SAP services that use the ODATA V2 API. How a CDS view is queried depends on whether the CDS view requires parameters.

CDS views that do not require parameters

If querying a CDS view that does not require parameters, two-part notation should be used to specify the CDS view. In the following example, `YY1_CDSVIEW_SCHEMA` is the name of the schema and `CDSVIEW_ENTITY` is the entity or table name of the CDS view.

```
SELECT * FROM YY1_CDSVIEW_SCHEMA.CDSVIEW_ENTITY
```

In the following SQL statement, a `WHERE` clause is used to fetch records for the given region.

```
SELECT * FROM YY1_CDSVIEW_SCHEMA.CDSVIEW_ENTITY WHERE REGION = 'Northwest'
```

CDS views that require parameters

For a CDS view that requires a parameter, the required parameter must be specified in a WHERE clause and the values of the parameter must be specified in an equality expression. For example:

```
SELECT * FROM YY1_CDSVIEW_SCHEMA.CDSVIEW_ENTITY WHERE P_KEYDATE = {d '2017-03-28'}
```

You can determine which tables require WHERE clause filtering by querying the system table SYSTEM_COLUMNS_REQUIRED_FILTERS that resides in the driver's internal schema INFORMATION_SCHEMA. The following example searches the schema YY1_CDSVIEW_SCHEMA for this information.

```
SELECT * FROM INFORMATION_SCHEMA.SYSTEM_COLUMNS_REQUIRED_FILTERS WHERE TABLE_SCHEM =
'YY1_CDSVIEW_SCHEMA'
```

Note:

- Some CDS views may require more than one parameter. In these instances, each required parameter must be included in the WHERE clause using an AND operator to separate the expressions.
- Some CDS views may have optional parameters. When an optional parameter is included in the WHERE clause, the driver includes the optional parameter in the request.
- Currently, the OR operator is not supported for separating parameter values.

See also

Supported SQL statements and extensions on page 81
Introduction to the SAP S/4HANA data model on page 107

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.

2

Tutorials

The following sections guide you through using the driver to access your data with some common third-party applications. For information on installing your driver and setting the CLASSPATH, see "Installing and setting-up the driver."

For details, see the following topics:

- Tableau
- DbVisualizer
- Interactive SQL for JDBC (JDBCISQL)

Tableau

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

To use the driver to access data with Tableau:

1. Navigate to the `\lib\`*`xx`* subdirectory of the Progress DataDirect installation directory; then, locate the `jar` file for your driver:

   `s4hana.jar`

Note: The same `.jar` file, `s4hana.jar`, is used for all SAP ODATA V2 services exposed through the SAP Gateway, including S/4HANA and BW/4HANA.

2. Copy the `.jar` file for your driver into the following directory:

   Windows: `C:\Program Files\Tableau\Drivers`

   Linux: `/opt/tableau/tableau_driver/jdbc`

3. Open Tableau. From the **Connect** menu, select **Other Databases (JDBC)**.
4. In the **Other Databases (JDBC)** dialog, provide values for the following fields; then, click **Sign In**.

   - **URL**: Copy and paste your connection URL into this field. The following examples show how to connect using basic authentication.

     S/4HANA:

     ```
     jdbc:datadirect:s4hana:ServerName=https://mycompany.s4hana.ondemand.com;
     User=jsmith;Password=secret;
     ```

     BW/4HANA:

     ```
     jdbc:datadirect:s4hana:ServerName=https://myinstance.company.com;
     User=jsmith;Password=secret;
     ```

     Note: See Basic authentication on page 45 for details.

   - **Dialect**: Select **SQL92** (the default) from the drop-down box.
   - **Username**: If required by the authentication method being used, enter the user name. Alternatively, this value can be specified with the `User` property in the connection string.
   - **Password**: If required by the authentication method being used, enter the password. Alternatively, this value can be specified with the `Password` property in the connection string.

5. The **Data Source** window appears. In the **Schema** field, select the schema for the service you want to use.
6. In the **Table** field, the tables stored in the selected schema are now exposed and available for selection.

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

DbVisualizer

After you have installed your driver and defined it on the CLASSPATH, you can use the driver to access your data with the third-party DbVisualizer tool. The following topics guide you through using DbVisualizer to add your driver, connect, and execute SQL statements.

Adding a driver

To add a driver with DbVisualizer:

1. Open DbVisualizer.
2. From the menu, select **Tools**>**Driver Manager**. The Driver Manager window opens.
3. From the Driver Manager menu, select **Driver**>**Create Driver**.
4. Click the  button to navigate to the location of the driver jar file; then, click **OK**. The following are the default locations for the driver:

   Windows

   ```
   C:\Program Files\Progress\DataDirect\JDBC\lib\60\s4hana.jar
   ```

   Linux

   ```
   /opt/Progress/DataDirect/JDBC/lib/60/s4hana.jar
   ```

   Note: The same `.jar` file, `s4hana.jar`, is used for all SAP ODATA V2 services exposed through the SAP Gateway, including S/4HANA and BW/4HANA.

5. Provide values for the following fields; then, close the Driver Manager window.

   - **Name**: Type an alias for your driver. For example:

     S4HANA

   - **URL Format**: Optionally, specify the format of the connection string for your driver. For example:

     ```
     jdbc:datadirect:s4hana:ServerName=<server>
     ```

   - **Driver Class**: From the drop-down menu, select the driver class for your driver:

     ```
     com.ddtek.jdbc.s4hana.S4HanaDriver
     ```

     Note: The same driver class is used for all SAP ODATA V2 services exposed through the SAP Gateway, including S/4HANA and BW/4HANA.

You can now use your driver with DbVisualizer. Proceed to "Connecting and executing SQL statements" for information on connecting and executing SQL statements.

Connecting and executing SQL statements

To use the driver to access data with DbVisualizer:

1. Open DbVisualizer.
2. From the menu, select **Database**>**New Connection**. When prompted to use the Connection Wizard, click **OK**.
3. Provide the following information when prompted; then, click **Next** to proceed:

   - **Connection alias**: Type the name to be used when referring to this connection.
   - **Driver**: Select the alias that you provided for your driver from the drop-down menu.

4. Provide values for the following fields; then, click **Finish**.

   - **Database URL**: Copy and paste your connection URL into this field. The following examples show how to connect using basic authentication.

     Note: You can also generate connection strings using SAP S/4HANA Configuration Manager. For more information, see Generating connection URLs with the Configuration Manager on page 37.

     S/4HANA:

     ```
     jdbc:datadirect:s4hana:ServerName=https://mycompany.s4hana.ondemand.com;
     User=jsmith;Password=secret;
     ```

     BW/4HANA:

     ```
     jdbc:datadirect:s4hana:ServerName=https://myinstance.company.com;
     User=jsmith;Password=secret;
     ```

     Note: See Basic authentication on page 45 for details.

5. To execute SQL statements, select **SQL Commander>New SQL Commander**. A SQL Commander tab opens.
6. Select values for the following fields:

   - **Database Connection**: Select connection alias you provided for the connection from the drop-down menu.
   - **Schema**: Select the schema you want to execute queries against from the drop-down menu.

7. In the SQL Commander tab, enter SQL commands you want to execute; then select **SQL Commander>Execute**. For example:

   To select all of the rows from the A_CUSTOMER table:

   ```
   SELECT * FROM A_CUSTOMER
   ```

   To select the URLs for a specified issue :

   ```
   SELECT CUSTOMER FROM <SCHEMA_NAME>.A_CUSTOMER WHERE CUSTOMERNAME = <customer_name>
   ```

   See "Supported SQL statements and extensions" for the supported syntax used to execute SQL statements.

   Note: If you are fetching large sets of data, you may want to limit the results using the Max Rows and Max Chars fields.

You have successfully accessed your data with DbVisualizer.

Interactive SQL for JDBC (JDBCISQL)

After you have installed your driver and defined it on the CLASSPATH, you can use the driver to access your data with Interactive SQL for JDBC (JDBCISQL). JDBCISQL supports a command line interface that allows you to connect to a data source, execute SQL statements and retrieve results for display on a terminal.

To execute commands with JDBCISQL:

1. Start the ISQL tool. Do one of the following:
   - On Windows, double-click the`jdbcisql.bat` file in the *`install_dir`*`\jdbcisql` folder. Or, from a command prompt, navigate to the *`install_dir`*`\jdbcisql` directory and run the `jdbcisql.bat` file.
   - On Linux and UNIX, change to the *`install_dir`*`\isql` directory and run `jdbcisql.sh`.

   The Interactive SQL prompt appears.
2. Type the driver name class; then, press **Enter**:

   ```
   com.ddtek.jdbc.s4hana.S4HanaDriver
   ```

   Note: The same driver class is used for all SAP ODATA V2 services exposed through the SAP Gateway, including S/4HANA and BW/4HANA.
3. Type `connect` followed by the connection URL for the driver; then, press **Enter**. For example:

   S/4HANA:

   ```
   connect jdbc:datadirect:s4hana:ServerName=https://mycompany.s4hana.ondemand.com;
   User=jsmith;Password=secret;
   ```

   BW/4HANA:

   ```
   connect jdbc:datadirect:s4hana:ServerName=https://myinstance.company.com;
   User=jsmith;Password=secret;
   ```

   If successful, the tool will return the time required to connect.
4. At the `ISQL>` prompt, issue a SQL command to query or modify the data source; then, press **Enter**. For example:

   ```
   SELECT * FROM <SCHEMA_NAME>.A_BUSINESSPARTNER;
   ```

   Note: SQL commands must be terminated by a semi-colon.

   Note: In addition to SQL commands, JDBCISQL supports a set of proprietary commands. Type `Help` at the prompt for a list of supported commands and syntax.

   The results of the command are displayed in the terminal.
5. After you are finished executing queries and commands, you can disconnect from the data source by typing the following; then, pressing **Enter**:

   ```
   DISCONNECT;
   ```
6. Press any key to end the session.

3

Configuring and connecting

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

After the driver has been installed and defined on your classpath, you can connect from your application to your data 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).

For details, see the following topics:

Setting the classpath

The driver must be defined on your CLASSPATH before you can connect. The CLASSPATH 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 CLASSPATH, you will receive a `class not found` exception when trying to load the driver. Set your system CLASSPATH to include the driver jar file as shown, where *`install_dir`* is the path to your product installation directory.

`install_dir``/lib/60/s4hana.jar`

Windows Example

```
CLASSPATH=.;C:\Program Files\Progress\DataDirect\JDBC\lib\60\s4hana.jar
```

UNIX Example

```
CLASSPATH=.:/opt/Progress/DataDirect/JDBC/lib/60/s4hana.jar
```

Note: The same `.jar` file, `s4hana.jar`, is used for all SAP ODATA V2 services exposed through the SAP Gateway, including S/4HANA and BW/4HANA.

Connecting using the JDBC Driver Manager

One way to connect to a service is through the JDBC DriverManager using the `DriverManager.getConnection()` method. As the following example shows, this method specifies a string containing a connection URL.

S/4HANA:

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:s4hana:ServerName=https://mycompany.s4hana.ondemand.com;
    User=jsmith;Password=secret;");
```

BW/4HANA:

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:s4hana:ServerName=https://myinstance.company.com;
    User=jsmith;Password=secret;");
```

Passing the connection URL

After setting the CLASSPATH, the required connection information needs to be passed in the form of a connection URL. The following example includes the properties required for connecting with basic authentication.

Connection URL Syntax

The connection URL takes the following form:

```
jdbc:datadirect:s4hana:ServerName=servername;User=user_name;Password=password;
[property=value[;...]];
```

where:

servername

specifies the base URL of the service to which you want to connect. It is comprised of either the domain name or the IP address of the service. For example, for S/4HANA, it can be either `https://mycompany.s4hana.ondemand.com` or `http://123.456.7.8`.

user_name

specifies the user name that is used to connect to the service.

password

specifies the password used to connect to the service.

Important: The password is a confidential value used to authenticate the application to the server. To prevent unauthorized access, this value must be securely maintained.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

Connection URL Examples

S/4HANA:

```
Connection conn = DriverManager.getConnection
 ("jdbc:datadirect:s4hana:ServerName=https://mycompany.s4hana.ondemand.com;
   User=jsmith;Password=secret;");
```

BW/4HANA:

```
Connection conn = DriverManager.getConnection
 ("jdbc:datadirect:s4hana:ServerName=https://myinstance.company.com
   User=jsmith;Password=secret;");
```

See also

Connection property descriptions on page 49
Connection URL examples on page 13

Generating connection URLs with the Configuration Manager

The driver includes a browser-based tool, Progress DataDirect SAP S/4HANA Configuration Manager, that allows you to generate connection URLs, test connections, and execute test queries. This section will guide you through generating and testing a connection URL that can be used by your application.

Note: The Progress DataDirect SAP S/4HANA Configuration Manager can be used for all SAP ODATA V2 services exposed through the SAP Gateway, including S/4HANA and BW/4HANA.

To generate a connection URL:

1. Open the SAP S/4HANA Configuration Manager by double-clicking the driver jar file. Or, in a command line, navigate to the directory containing your driver jar file; then, execute the following command:

   ```
   java -jar s4hana.jar
   ```

   The SAP S/4HANA Configuration Manager opens in your default web browser.

   Note: The same `.jar` file, `s4hana.jar`, is used for all SAP ODATA V2 services exposed through the SAP Gateway, including S/4HANA and BW/4HANA.

2. From the browser window, provide values of the connection properties you want to configure in the corresponding fields. A connection URL will generate in the Connection String field as you provide settings. To view more properties, select the tabs at the top of the page. See "Connection URL examples" for a list of required properties and properties used for different configurations.

   Note: If you do not specify a value for an optional property, the property will be omitted from the string and the default value will be used.

3. Optionally, you can manually edit your string by clicking the Edit button ().
4. At any point during the process, you can click **Test Connect** to attempt to connect to the service using the string generated in the Connection String field. In the **Test Connection** window:
   a) Provide values for any fields required by your service.
   b) Optionally, in the Test Query field, enter any SQL queries you want to execute during the test. For example:

      ```
      SELECT * FROM INFORMATION_SCHEMA.SYSTEM_TABLES
      ```

   c) Click **Execute**.

      If successful, the window displays a confirmation message and, if a query was specified, the results of the query.

   Note: The information you enter in the logon dialog box during a test connect is not saved.

5. To use your string, click the Copy button () and paste the string to a location that can be used by your application.
6. Optionally, click **Save** to store your connection string for later use.

See also

Connection URL examples on page 13

Testing connections and queries

You can quickly test a connection string and queries using Progress DataDirect SAP S/4HANA Configuration Manager.

To test your connection and query:

1. Open the SAP S/4HANA Configuration Manager by double-clicking on the driver jar file. Or, in a command line, navigate to the directory containing your driver jar file; then, execute the following command:

   ```
   java -jar s4hana.jar
   ```

   The SAP S/4HANA Configuration Manager opens in your default web browser.

   Note: The same `.jar` file, `s4hana.jar`, is used for all SAP ODATA V2 services exposed through the SAP Gateway, including S/4HANA and BW/4HANA.

2. Provide a connection string to test using one of the following methods:

   - Entering a string: Click the Edit button (); then, paste your string into the Connection String field. If you prefer, you can also type a string directly into this field.
   - Generating a string: Provide values of the connection properties you want to configure into the corresponding fields. The SAP S/4HANA Configuration Manager will generate a string in the Connection String field based on the values you specify.

3. Click **Test Connect** to attempt to connect to the service using the string specified in the Connection String field. The **Test Connection** window appears.
4. Provide connection property values for any fields required by your service.
5. To execute a test query with the test connection, enter a SQL query into the Test Query field. For example:

   ```
   SELECT * FROM INFORMATION_SCHEMA.SYSTEM_TABLES
   ```

6. Click **Execute**.

   If successful, the window displays a confirmation message and, if a query was specified, the results of the query.

Connecting using data sources

A *JDBC data source* is a Java object, specifically a `DataSource` 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 data source class implements the following interfaces.

- `javax.sql.DataSource`
- `javax.sql.ConnectionPoolDataSource` (allows applications to use connection pooling)

Refer to "Connection Pool Manager" in the *Progress DataDirect for JDBC Drivers Reference* for more information.

See also

Driver and DataSource classes on page 13

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 data source 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 data source and save it in your local file system using the File System JNDI Provider.

See "Example data source" for an example data source definition for the example files.

To connect using a JNDI data source, 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 CLASSPATH. These steps are not required for LDAP implementations because the LDAP Service Provider is included with supported versions of Java SE.

See also

Example data source on page 40

Example data source

To configure a data source using the example files, you will need to create a data source definition. The content required to create a data source definition is divided into three sections.

First, you will need to import the data source class. For example:

```
import com.ddtek.jdbcx.s4hana.S4HanaDataSource;
```

Note: The data source class to be imported is the same for all SAP ODATA V2 services exposed through the SAP Gateway, including S/4HANA and BW/4HANA.

Next, you will need to set the values and define the data source. For example, the following definition contains the minimum properties required for a connection using the basic authentication method.

Note:

- Setting the password using a data source is generally not recommended. The data source persists all properties, including the Password property, in clear text.
- In a JDBC data source, string values must be enclosed in double quotation marks, for example, `setUser("abc@defcorp.com")`.

```
S4HanaDataSource mds = new S4HanaDataSource();
mds.setDescription("My S4Hana Data Source");
mds.setServerName("mycompany.s4hana.ondemand.com");
mds.setUser("username");
mds.setPassword("password");
```

Finally, you will need to configure the example application to print out the data source attributes. Note that this code is specific to the driver and should only be used in the example application. For example, you would add the following section for the minimum properties required to establish a connection:

```
if (ds instanceof S4HanaDataSource)
{
S4HanaDataSource jmds = (S4HanaDataSource) ds;
System.out.println("description=" + jmds.getDescription());
System.out.println("servername=" + jmds.getServerName());
System.out.println("user=" + jmds.getUser());
System.out.println("password=" + jmds.getPassword());
...
System.out.println();
}
```

Calling a data source in an application

Applications can call a Progress DataDirect data source 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 data source 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.

Testing a data source connection

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

Take the following steps to establish a connection.

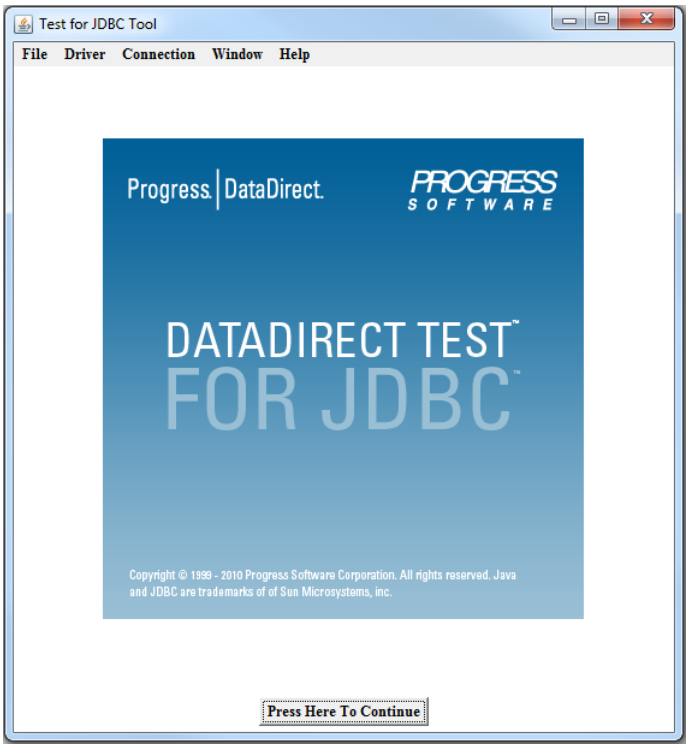
1. Navigate to the installation directory. The default location is:
   - Windows systems: `Program Files\Progress\DataDirect\JDBC\testforjdbc`

- 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.

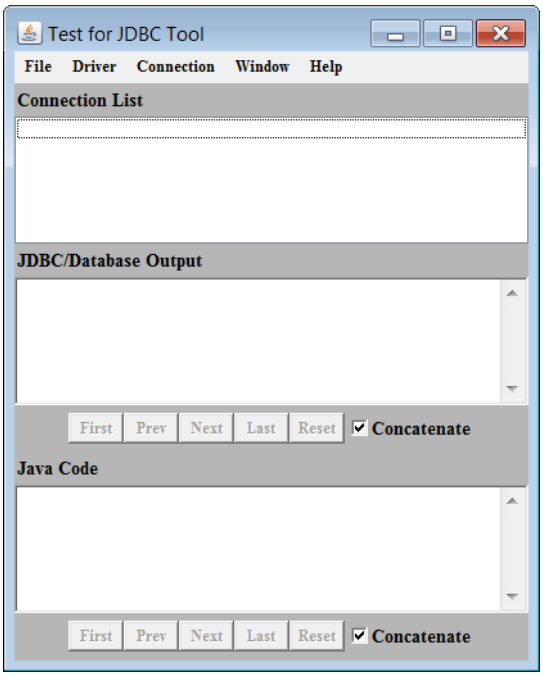
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

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



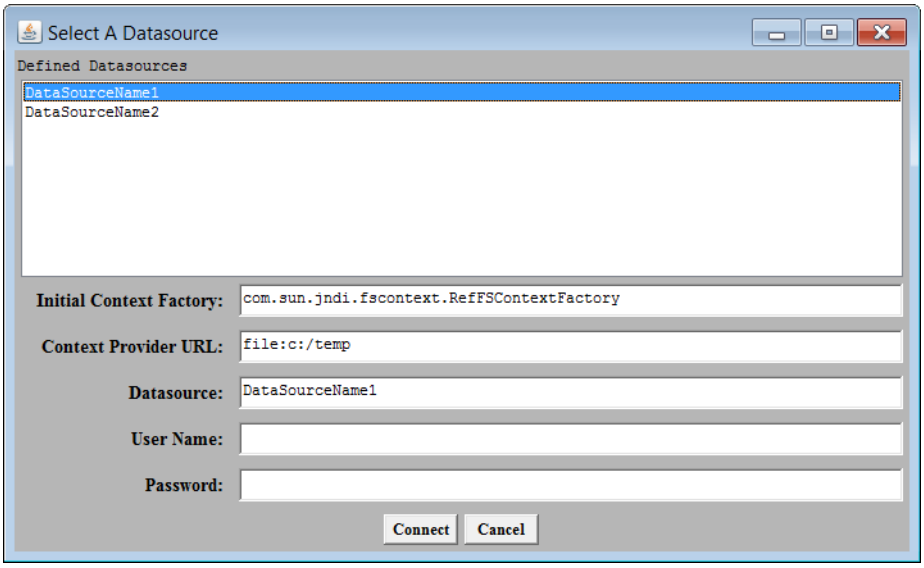
3. Click **Press Here to Continue**.

   The main dialog appears:



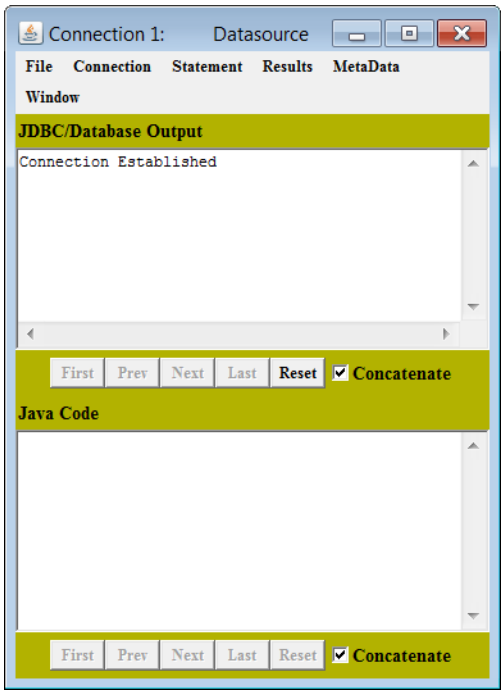
4. From the menu bar, select **Connection > Connect to DB via Data Source**.

   The **Select A Database** dialog appears:



5. Select a datasource template from the **Defined Datasources** field.
6. Provide the following information:
   a) In the **Initial Context Factory**, specify the location of the initial context provider for your application.
   b) In the **Context Provider URL**, specify the location of the context provider for your application.
   c) In the **Datasource** field, specify the name of your datasource.
7. If you are using user ID/password authentication, enter your user ID and password in the corresponding fields.
8. 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.



Authentication

The driver supports the following authentication methods:

- *Basic* authenticates using the specified user IDs and passwords.
- (S/4HANA only) *HTTP Header* passes security tokens via the HTTP headers to authenticate with the S/4HANA sandbox hosted at `https://api.sap.com`.

By default, the driver is configured to use basic authentication (`AuthenticationMethod=Basic`).

See also

AuthenticationMethod on page 54

Basic authentication

To configure the driver to use basic authentication:

- Set the ServerName property to specify the base URL of the service to which you want to connect. It is comprised of either the domain name or the IP address of the service. For example, for S/4HANA, it can be either `https://mycompany.s4hana.ondemand.com` or `http://123.456.7.8`.
- Set the User property to specify your logon ID.
- Set the Password property to specify your password.
- Optionally, specify values for any additional properties you want to configure.

The following examples demonstrate a session with basic authentication enabled.

For a connection URL:

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:s4hana:ServerName=https://myinstance.company.com
  User=jsmith;Password=secret;");
```

For a data source:

```
S4HanaDataSource mds = new S4HanaDataSource();
mds.setDescription("My S4Hana Data Source");
mds.setServerName("https://myinstance.company.com");
```

See also

Connection property descriptions on page 49

HTTP header authentication (S/4HANA only)

HTTP header authentication can be used to authenticate with the S/4HANA sandbox hosted at `https://api.sap.com`.

To configure the driver to use HTTP header authentication:

- Set the ServerName property to the S/4HANA sandbox instance: `https://api.sap.com`.
- Set the AuthenticationMethod property to `HTTP Header`.
- Set the AuthHeader property to specify the name of the HTTP header used for authentication.
- Set the SecurityToken to specify the security token required to make a connection to your endpoint. For example, `XaBARTsLZReM`.
- Optionally, specify values for any additional properties you want to configure.

The following examples demonstrate a session with HTTP header authentication enabled.

For a connection URL:

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:s4hana:ServerName=https://api.sap.com;
  AuthenticationMethod=HttpHeader;AuthHeader=APIKey;SecurityToken=XaBARTsLZReM;");
```

For a data source:

```
S4HanaDataSource mds = new S4HanaDataSource();
mds.setDescription("My S4Hana Data Source");
mds.setAuthenticationMethod("HttpHeader");
mds.setAuthHeader("APIKey");
mds.setSecurityToken("XaBARTsLZReM");
mds.setServerName("https://api.sap.com");
```

Note: If setting the security token using a data source, be aware that the SecurityToken property, like all data source properties, is persisted in clear text.

See also

Connection property descriptions on page 49

Data Encryption

All communication between the driver and SAP S/4HANA is encrypted using TLS/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.

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.

Performance considerations

CreateMap: CreateMap determines whether the driver creates the internal files required for a relational map of the native data when establishing a connection. When the driver creates these internal files, an initial connection can take a few minutes, depending on network speeds and the amount of metadata the driver must retrieve from the service. For example, when CreateMap is set to OnChange (the default) and changes have been made to the back-end schema, the connection may be delayed because the driver runs a discovery on the back-end schema and overwrites the internal files used to create a relational map of the data.

InsensitiveResultSetBufferSize: To improve performance when using scroll-insensitive result sets, the driver can cache the result set data in memory instead of writing it to disk. By default, the driver caches 2 MB of insensitive result set data in memory and writes any remaining result set data to disk. Performance can be improved by increasing the amount of memory used by the driver before writing data to disk or by forcing the driver to never write insensitive result set data to disk. The maximum cache size setting is 2 GB.

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

For example, if an application attempts to fetch 100,000 rows from the remote data source and WSFetchSize is set to `500`, the driver must make 200 Web service calls to get the 100,000 rows. If, however, WSFetchSize is set to `4000`, the driver only needs to make 25 Web service calls to retrieve 100,000 rows. Web service calls are expensive, so generally, minimizing Web service calls increases throughput.

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

Note that the values specified for the WSFetchSize and FetchSize properties provide only a suggestion as to the number of rows to be processed. The service will not exceed these values, but it will adjust page sizes to balance throughput amongst all its users. This behavior can result in different page sizes for successive queries.

MaxPooledStatements: To improve performance, the driver's own internal prepared statement pooling should be enabled when the driver does not run from within an application server or from within another application that does not provide its own prepared statement pooling. When the driver's internal prepared statement pooling is enabled, the driver caches a certain number of prepared statements created by an application. For example, if the MaxPooledStatements property is set to 20, the driver caches the last 20 prepared statements created by the application. If the value set for this property is greater than the number of prepared statements used by the application, all prepared statements are cached.

ServiceList: If your application does not require connecting to all the services to which you have access, you can improve performance using the ServiceList property. By default, the driver attempts to discover and connect to all the services to which it has access. However, you can limit the number of services the driver attempts to connect to by specifying only the required services with this property. This eliminates the discovery process and reduces the overhead associated with connecting to services you do not use, thereby improving performance.

ServiceVersion: When there are multiple versions of a service, ServiceVersion determines which versions the driver returns metadata for. If you only require metadata for the latest version of a service, set this property to `Latest`, the default. This improves performance at connection by eliminating the overhead associated with fetching metadata for multiple versions.

WSCompressData: Allows you to compress data sent to and from the Web server. By enabling this connection option, you can significantly improve performance by reducing the size of data transferred between the driver and the Web server.

4

Connection property descriptions

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 data source. For a `DriverManager` 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 `set`*`property`*`(`*`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 tables describe the connection properties by functionality:

- Basic authentication properties
- HTTP Header authentication properties
- Mapping properties
- Proxy server properties
- Web service properties

- Statement pooling properties
- Additional properties

Basic authentication properties

The following table summarizes the connection properties required to connect to a service when using basic authentication.

Table 4: Required properties

Property	Data Source Method	Default
AuthenticationMethod on page 54	`getAuthenticationMethod()` `setAuthenticationMethod(String)`	`Basic`
ServerName on page 71	`getServerName()` `setServerName(String)`	No default value
Password on page 64	`getPassword()` `setPassword(String)`	No default value
User on page 78	`getUser()` `setUser(String)`	No default value

HTTP Header properties (S/4HANA only)

The following table summarizes the connection properties required to connect to a service when using HTTP Header authentication. This authentication method is supported only for S/4HANA services.

Property	Data Source Method	Default
AuthenticationMethod on page 54	`getAuthenticationMethod()` `setAuthenticationMethod(String)`	`Basic`
AuthHeader on page 55	`getAuthHeader()` `setAuthHeader(String)`	No default value
ServerName on page 71	`getServerName()` `setServerName(String)`	No default value
SecurityToken on page 71	`getSecurityToken()` `setSecurityToken(String)`	No default value

Mapping properties

The following table summarizes the connection properties involved in mapping the SAP S/4HANA data model to a SQL model.

Property	Data Source Method	Default
CreateMap on page 56	`getCreateMap()` `setCreateMap(String)`	`NotExist`
KeywordConflictSuffix on page 61	`getKeywordConflictSuffix()` `setKeywordConflictSuffix(String)`	No default value
RefreshSchema on page 68	`getRefreshSchema()` `setRefreshSchema(Boolean)`	`false`
SchemaMap on page 69	`getSchemaMap()` `setSchemaMap(String)`	Default value depends on environment

Proxy server properties

The following table summarizes the proxy server connection properties.

Table 5: Proxy Server Properties

Property	Data Source Method	Default
ProxyHost on page 64	`getProxyHost()` `setProxyHost(String)`	No default value
ProxyPassword on page 65	`getProxyPassword()` `setProxyPassword(String)`	No default value
ProxyPort on page 66	`getProxyPort()` `setProxyPort(Integer)`	`0` which means the default is determined by the ProxyHost property. For HTTP URLs: `80` For HTTPS URLs: `443`
ProxyUser on page 66	`getProxyUser()` `setProxyUser(String)`	No default value

Web service properties

The following table summarizes the web service connection properties.

Table 6: Web Service Properties

Property	Data Source Method	Default
StmtCallLimit on page 76	`getStmtCallLimit()` `setStmtCallLimit(Integer)`	`0` (no limit)
StmtCallLimitBehavior on page 77	`getStmtCallLimitBehavior()` `setStmtCallLimitBehavior(String)`	`errorAlways`
WSCompressData on page 79	`getWsCompressData()` `setWsCompressData(String)`	`Compress`
WSFetchSize on page 80	`getWsFetchSize()` `setWsFetchSize(Integer)`	`100000`

Statement pooling properties

The following table summarizes the statement pooling connection properties.

Table 7: Statement Pooling Properties

Property	Data Source Method	Default
ImportStatementPool on page 58	`getImportStatementPool()` `setImportStatementPool(String)`	No default value
MaxPooledStatements on page 62	`getMaxPooledStatements()` `setMaxPooledStatements(Integer)`	`0`
RegisterStatementPoolMonitorMBean on page 68	`getRegisterStatementPoolMonitorMbean()` `setRegisterStatementPoolMonitorMbean(Boolean)`	`false`

Additional properties

The following table summarizes additional connection properties.

Table 8: Additional Properties

Property	Data Source Method	Default
ConvertNull on page 56	`getConvertNull()` `setConvertNull(Boolean)`	`true`
FetchSize on page 57	`getFetchSize()` `setFetchSize(Integer)`	`100` (rows)

Property	Data Source Method	Default
InitializationString on page 59	`getInitializationString()` `setInitializationString(String)`	No default value
InsensitiveResultSetBufferSize on page 60	`getInsensitiveResultsetBufferSize()` `setInsensitiveResultsetBufferSize(Integer)`	`2048` (KB of memory)
LogConfigFile on page 61	`getLogConfigFile()` `setLogConfigFile(String)`	`ddlogging.properties`
MaxVarcharSize on page 63	`getMaxVarcharSize()` `setMaxVarcharSize(Integer)`	`64000`
ReadOnly on page 67	`getReadOnly()` `setReadOnly(Boolean)`	`false`
ServiceList on page 72	`getServiceList()` `setServiceList(String)`	No default value
ServiceVersion on page 73	`getServiceVersion()` `setServiceVersion(String)`	`Latest`
SpyAttributes on page 74	`getSpyAttributes()` `setSpyAttributes(String)`	No default value
TransactionMode on page 78	`getTransactionMode()` `setTransactionMode(String)`	`NoTransactions`

For details, see the following topics:

- AuthenticationMethod
- AuthHeader
- ConvertNull
- CreateMap
- FetchSize
- ImportStatementPool
- InitializationString
- InsensitiveResultSetBufferSize
- KeywordConflictSuffix

AuthenticationMethod

Purpose

Determines which authentication method the driver uses during the course of a session.

Valid Values

`Basic` | `HTTPHeader`

Behavior

If set to `Basic`, the driver uses a hashed value, based on the concatenation of the user name and password, for authentication.

(S/4HANA only) If set to `HttpHeader`, the driver passes security tokens via HTTP headers for authentication. You must also configure `SecurityToken` property and, if the name of your HTTP header is not `Authorization` (the default), the `AuthHeader` property.

Data Source Methods

```
public String getAuthenticationMethod()
```

```
public void setAuthenticationMethod(String)
```

Default Value

`Basic`

Data Type

String

See also

Authentication on page 44

AuthHeader

Purpose

Specifies the name of the HTTP header used for authentication. or Header-based token authentication (`AuthenticationMethod=HTTPHeader`) is enabled; otherwise, this property is ignored.

Valid Values

`auth_header`

where:

`auth_header`

is the name of the HTTP header used for authentication. For example, `X-Api-Key`.

Data Source Methods

```
public String getAuthHeader()
```

```
public void setAuthHeader(String)
```

Default Value

No default value

Data Type

String

See also

HTTP header authentication (S/4HANA only) on page 45

ConvertNull

Purpose

Controls how data conversions are handled for null values.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver checks the data type being requested against the data type of the table column that stores the data. If a conversion between the requested type and column type is not defined, the driver generates an "unsupported data conversion" exception regardless of whether the column value is `NULL`.

If set to `false`, the driver does not perform the data type check if the value of the column is `NULL`. This allows null values to be returned even though a conversion between the requested type and the column type is undefined.

Data Source Methods

```
public Boolean getConvertNull()
```

```
public void setConvertNull(Boolean)
```

Default Value

`true`

Data Type

Boolean

CreateMap

Purpose

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

Valid Values

`NotExist` | `ForceNew` | `No` | `Session`

Behavior

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.

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.

Warning: `ForceNew` causes all views, data caches, and map customizations defined in the current schema map to be lost.

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 `Session`, the driver uses memory to store the internal configuration information and relational map of the native data model. After the session, the view is discarded.

Notes

- The internal files share the same directory as the SchemaMap 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 Methods

```
public String getCreateMap()
```

```
public void setCreateMap(String)
```

Default Value

`Session`

Data Type

String

See also

SchemaMap on page 69
Refresh Map (EXT) on page 83

FetchSize

Purpose

Specifies the maximum number of rows that the driver processes before returning data to the application when executing a Select. This value provides a suggestion to the driver as to the number of rows it should internally process before returning control to the application. The driver may fetch fewer rows to conserve memory when processing exceptionally wide rows.

Valid Values

`0` | *x*

where:

x

is a positive integer indicating the number of rows that should be processed.

Behavior

If set to `0`, the driver processes all the rows of the result before returning control to the application. When large data sets are being processed, setting FetchSize to `0` can diminish performance and increase the likelihood of out-of-memory errors.

If set to *x*, the driver limits the number of rows that may be processed for each fetch request before returning control to the application.

Notes

- To optimize throughput and conserve memory, the driver uses an internal algorithm to determine how many rows should be processed based on the width of rows in the result set. Therefore, the driver may process fewer rows than specified by FetchSize when the result set contains exceptionally wide rows. Alternatively, the driver processes the number of rows specified by FetchSize when the result set contains rows of unexceptional width.
- FetchSize can be used to adjust the trade-off between throughput and response time. Smaller fetch sizes can improve the initial response time of the query. Larger fetch sizes can improve overall response times at the cost of additional memory.
- You can use FetchSize to reduce demands on memory and decrease the likelihood of out-of-memory errors. Simply, decrease FetchSize to reduce the number of rows the driver is required to process before returning data to the application.

Data Source Methods

```
public Integer getFetchSize()
```

```
public void setFetchSize(Integer)
```

Default Value

`100`

Data Type

Integer

See also

Performance considerations on page 46
WSFetchSize on page 80

ImportStatementPool

Purpose

Specifies the path and file name of the file to be used to load the contents of the statement pool. When this property is specified, statements are imported into the statement pool from the specified file.

Valid Values

`String`

where:

String

is the path and file name of the file to be used to load the contents of the statement pool.

Notes

- If the driver cannot locate the specified file when establishing the connection, the connection fails and the driver throws an exception.
- For more information, refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference*.

Data Source Methods

```
public String getImportStatementPool()
```

```
public void setImportStatementPool(String)
```

Default Value

No default value

Data Type

String

InitializationString

Purpose

Specifies one or multiple SQL commands to be executed by the driver after it has established a connection and has performed all initialization for the connection. If the execution of a SQL command fails, the connection attempt also fails and the driver throws an exception indicating which SQL command or commands failed.

Valid Values

command[[;*command*]...]

where:

command

is a SQL command.

Notes

Multiple commands must be separated by semicolons. In addition, if this property is specified in a connection URL, the entire value must be enclosed in parentheses when multiple commands are specified.

Data Source Methods

```
public String getInitializationString()
```

```
public void setInitializationString(String)
```

Default Value

No default value

Data Type

String

InsensitiveResultSetBufferSize

Purpose

Determines the amount of memory that is used by the driver to cache insensitive result set data.

Valid Values

`-1` | `0` | *x*

where:

x

is a positive integer that represents the amount of memory.

Behavior

If set to `-1`, the driver caches insensitive result set data in memory. If the size of the result set exceeds available memory, an OutOfMemoryException is generated. With no need to write result set data to disk, the driver processes the data efficiently.

If set to `0`, the driver caches insensitive result set data in memory, up to a maximum of 2 MB. If the size of the result set data exceeds available memory, then the driver pages the result set data to disk, which can have a negative performance effect. Because result set data may be written to disk, the driver may have to reformat the data to write it correctly to disk.

If set to *x*, the driver caches insensitive result set data in memory and uses this value to set the size (in KB) of the memory buffer for caching insensitive result set data. If the size of the result set data exceeds available memory, then the driver pages the result set data to disk, which can have a negative performance effect. Because the result set data may be written to disk, the driver may have to reformat the data to write it correctly to disk. Specifying a buffer size that is a power of 2 results in efficient memory use.

Data Source Methods

```
public Integer getInsensitiveResultsetBufferSize()
```

```
public void setInsensitiveResultsetBufferSize(Integer)
```

Default Value

`2048`

Data Type

Integer

See also

Performance considerations on page 46

KeywordConflictSuffix

Purpose

Specifies a string of up to 5 alphanumeric characters that the driver appends to any object or field name that conflicts with a SQL engine keyword.

Valid Values

String

where:

String

is a string of up to 5 alphanumeric characters.

Example

A field called CASE exists in the data schema. To avoid a naming conflict with the SQL engine keyword CASE, you could set KeywordConflictSuffix=TAB. In this scenario, the driver maps the CASE field to the CASETAB column.

Data Source Methods

```
public String getKeywordConflictSuffix()
```

```
public void setKeywordConflictSuffix(String)
```

Default Value

No default value

Data Type

String

LogConfigFile

Purpose

Specifies the file name, and optionally, the path of the properties file used to initialize driver logging.

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 as described in "Using Java Logging" in the *Progress DataDirect for JDBC Drivers Reference*.

Data Source Methods

```
public String getLogConfigFile()
```

```
public void setLogConfigFile(String)
```

Default Value

`ddlogging.properties`

Data Type

String

MaxPooledStatements

Purpose

Specifies the maximum number of prepared statements to be pooled for each connection and enables the driver's internal prepared statement pooling when set to an integer greater than zero (0). The driver's internal prepared statement pooling provides performance benefits when the driver is not running from within an application server or another application that provides its own statement pooling.

Valid Values

`0` | *x*

where:

x

is a positive integer that represents a number of prepared statements to be cached.

Behavior

If set to `0`, the driver's internal prepared statement pooling is not enabled.

If set to *x*, the driver's internal prepared statement pooling is enabled and the driver uses the specified value to cache up to that many prepared statements created by an application. If the value set for this property is greater than the number of prepared statements that are used by the application, all prepared statements that are created by the application are cached. Because CallableStatement is a sub-class of PreparedStatement, CallableStatements also are cached.

Example

If the value of this property is set to `20`, the driver caches the last 20 prepared statements that are created by the application.

Notes

When you enable statement pooling, your applications can access the Statement Pool Monitor directly with DataDirect-specific methods. However, you can also enable the Statement Pool Monitor as a JMX MBean. To enable the Statement Pool Monitor as an MBean, statement pooling must be enabled with MaxPooledStatements and the Statement Pool Monitor MBean must be registered using the RegisterStatementPoolMonitorMBean connection property.

Data Source Methods

```
public Integer getMaxPooledStatements()
```

```
public void setMaxPooledStatements(Integer)
```

Default Value

```
0
```

Data Type

Integer

MaxVarcharSize

Purpose

Determines the maximum size of VARCHAR columns when described within the result set metadata.

Valid Values

x

where:

x

is an integer greater than or equal to 1 and less than or equal to 2147483647.

Notes

When string functions are used within the Select list of a Select statement, the driver describes the resulting string value with a size of 2147483647. For instance, concatenating two columns that are each defined as VARCHAR(10) will result in a VARCHAR(2147483647) rather than a VARCHAR(20). The unnecessarily long size can result in undesirable behavior in some JDBC applications.

Data Source Methods

```
public Integer getMaxVarcharSize()
```

```
public void setMaxVarcharSize(Integer)
```

Default Value

```
64000
```

Data Type

Integer

Password

Purpose

A password that is used to connect to the instance.

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

Behavior

password

where:

password

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

Data Source Methods

`public String getPassword()`

`public void setPassword(String)`

Default Value

No default value

Data Type

String

See also

Basic authentication example on page 14

ProxyHost

Purpose

Identifies a proxy server to use for the first connection.

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 Methods

```
public String getProxyHost()
```

```
public void setProxyHost(String)
```

Default Value

No default value

Data Type

String

See also

Proxy server example on page 16

ProxyPassword

Purpose

Specifies the password needed to connect to a proxy server for the first connection.

Valid Values

password

where:

password

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

Data Source Methods

```
public String getProxyPassword()
```

```
public void setProxyPassword(String)
```

Default Value

No default value

Data Type

String

See also

Proxy server example on page 16

ProxyPort

Purpose

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

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 Methods

```
public Integer getProxyPort()
```

```
public void setProxyPort(Integer)
```

Default Value

0 which means that the default value is determined by whether the value specified for the ProxyHost property is an HTTP or HTTPS URL.

For HTTP: 80

For HTTPS: 443

Data Type

Integer

See also

Proxy server example on page 16

ProxyUser

Purpose

Specifies the user name needed to connect to a proxy server for the first connection.

Valid Values

user_name

where:

user_name

is a valid user ID for the proxy server.

Data Source Methods

```
public String getProxyUser()
```

```
public void setProxyUser(String)
```

Default Value

No default value

Data Type

String

See also

Proxy server example on page 16

ReadOnly

Purpose

Specifies whether the connection has read-only access to the data source.

Valid Values

`true` | `false`

Behavior

If set to `true`, the connection has read-only access.

If set to `false`, the connection is opened for read/write access, and you can use all commands supported by the product.

Notes

You can use use the JDBC connection method setReadOnly to set a read-only state for a connection.

Data Source Methods

```
public Boolean getReadOnly()
```

```
public void setReadOnly(Boolean)
```

Default Value

`false`

Data Type

Boolean

RefreshSchema

Purpose

Specifies whether the driver automatically refreshes the relational map of the schema when a user connects to the instance.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver automatically refreshes the map when a user first connects to the instance. The driver rebuilds the relational map of the schema, and any changes that have been made to the schema since the last refresh will be shown in the metadata.

If set to `false`, the driver does not refresh the relational map when a user first connects.

Notes

- This property should not be enabled (`RefreshSchema=true`) when `CreateMap=Session`.
- This property is equivalent to executing the Refresh Schema statement.

Data Source Methods

```
public Boolean getRefreshSchema()
```

```
public void setRefreshSchema(Boolean)
```

Default Value

`false`

Data Type

Boolean

RegisterStatementPoolMonitorMBean

Purpose

Registers the Statement Pool Monitor as a JMX MBean when statement pooling has been enabled with MaxPooledStatements. This allows you to manage statement pooling with standard JMX API calls and to use JMX-compliant tools, such as JConsole.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver registers an MBean for the statement pool monitor for each statement pool. This gives applications access to the Statement Pool Monitor through JMX when statement pooling is enabled.

If set to `false`, the driver does not register an MBean for the Statement Pool Monitor for any statement pool.

Notes

- Registering the MBean exports a reference to the Statement Pool Monitor. The exported reference can prevent garbage collection on connections if the connections are not properly closed. When garbage collection does not take place on these connections, out of memory errors can occur.
- For more information, refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference*.

Data Source Methods

```
public Boolean getRegisterStatementPoolMonitorMbean()
```

```
public void setRegisterStatementPoolMonitorMbean(Boolean)
```

Default Value

`false`

Data Type

Boolean

See also

MaxPooledStatements on page 62

SchemaMap

Purpose

Specifies either the name or the absolute path and name of the configuration file where the map of the database data model is written. The driver looks for this file when connecting to a database instance. If the file does not exist, the driver creates one.

Valid Values

`string`

where:

`string`

is either the name or the absolute path and name (including the `.config` extension) of the configuration file. For example, if specifying a value of:

- `ABC`, the driver either creates or looks for the configuration file `ABC` in the working directory of your application.
- `C:\\Users\\Default\\AppData\\Local\\Progress\\DataDirect\\<driver>_Schema\\abc@defcorp.com.config`, the driver either creates or looks for the configuration file `abc@defcorp.com.config` in the directory `C:\Users\Default\AppData\Local\Progress\DataDirect\<driver>_Schema`.

Notes

- When connecting to a database instance, the driver looks for the schema map configuration file. If the configuration file does not exist, the driver creates the schema map configuration file using the name and location you have provided. If you do not provide a name and location for the 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 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.

Data Source Methods

```
public String getSchemaMap()
```

```
public void setSchemaMap(String)
```

Default Value

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
- UNIX/Linux
  - DD_HOME environment variable
  - dd.home system property
  - user.home system property

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

```
<available_location>/progress/datadirect/<driver>_schema/<user_name>.config
```

Data Type

String

See also

Refresh Map (EXT) on page 83

SecurityToken

Purpose

Specifies the API key required to make a connection to the server. This property is required when token-based authentication is enabled (`AuthenticationMethod=HTTPHeader`); otherwise, this property is ignored. If an API key is required and you do not supply one, the driver returns an error indicating that an invalid user or password was supplied.

Important: If setting the security token using a data source, be aware that the SecurityToken property, like all data source properties, is persisted in clear text.

Valid Values

`string`

where:

`string`

is the value of the API key assigned to the user.

Data Source Methods

```
public String getSecurityToken()
```

```
public void setSecurityToken(String)
```

Default Value

No default value

Data Type

String

See also

HTTP header authentication (S/4HANA only) on page 45

ServerName

Purpose

Specifies the base URL of the service to which you want to connect. The base URL is comprised of either the domain name or the IP address of the service.

Valid Values

`string`

where:

`string`

is the base URL of the service to which you want to connect. For example:

- For S/4HANA:
  - Base URL with a domain name: `https://mycompany.s4hana.ondemand.com`
  - Base URL with an IP address: `http://123.456.7.8`
- For BW/4HANA:
  - Base URL with a domain name: `https://myinstance.company.com`
  - Base URL with an IP address: `http://123.456.7.8`

Notes

- The HostName property is an alias for the ServerName property.

Data Source Methods

```
public String getServerName()
```

```
public void setServerName(String)
```

Default Value

No default value

Data Type

String

ServiceList

Purpose

Specifies a comma-separated list of standard and/or custom services to which the driver connects. This property allows you to limit the services to which the driver connects, thereby improving performance at connection. In addition, this property provides a method to connect to services not included in your communication scenario.

Valid Values

service_name[,*service_name*,...]

where:

service_name

is the name of the service to which you want to connect.

Example

```
API_BUSINESS_PARTNER,API_SALES_ORDER_SRV
```

Notes

- If no value is specified for this property, the driver attempts to discover all the services included in your communication scenario or to which you have access.

- This property is recommended when you do not have access to the *SAP Analytics Cloud for Planning Integration (SAP_COM_0087)* communication scenario.

Data Source Methods

`public String getServiceList()`

`public void setServiceList(String)`

Default Value

No default value

Data Type

String

ServiceVersion

Purpose

Determines which versions of a service the driver returns metadata for when the service has multiple versions.

Valid Values

`Latest|All`

Behavior

If set to `Latest`, the driver returns metadata only for the latest version of the service(s) to which it connects. For example, if the service has both V=001 and V=002, the driver returns metadata only for V=002.

If set to `All`, the driver returns metatdata for every version of the service to which it connects.

Notes

- If a service has only one version, the driver returns only the metadata for that version.
- If a value is specified for the ServiceList property, the driver returns metadata only for the services and versions specified by the ServiceList and ServiceVersion properties.
- If metadata is required only for the latest version of a service, set this property to `Latest`, the default. This improves performance at connection by limiting the overhead associated with fetching metadata for multiple versions.

Data Source Methods

`public String getServiceVersion()`

`public void setServiceVersion(String)`

Default Value

`Latest`

Data Type

String

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
linelimit=*numberofchars*	Sets the maximum number of characters that DataDirect Spy logs on a single line. The default is 0 (no maximum limit).
log=(file)*filename*	Directs logging to the file specified by *filename*. 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: log=(file)C:\\temp\\spy.log;logIS=yes;logTName=yes.

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

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)
```

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.
- For more information, refer to "Tracking JDBC calls with DataDirect Spy" in the *Progress DataDirect for JDBC Drivers Reference*.

Data Source Methods

```
public String getSpyAttributes()
```

```
public void setSpyAttributes(String)
```

Default Value

No default value

Data Type

String

StmtCallLimit

Purpose

Specifies the maximum number of web service calls the driver can make when executing any single SQL statement or metadata query.

Valid Values

`0` | *x*

where:

x is a positive integer that defines the maximum number of web service calls up to 2147483647 the driver can make when executing any single SQL statement or metadata query.

Behavior

If set to `0`, there is no limit.

If set to *x*, the driver uses this value to set the maximum number of web service calls on a single connection that can be made when executing a SQL statement. This limit can be overridden by changing the STMT_CALL_LIMIT session attribute using the ALTER SESSION statement. For example, the following statement sets the statement call limit to 10 web service calls:

```
ALTER SESSION SET STMT_CALL_LIMIT=10
```

If the web service call limit is exceeded, the behavior of the driver depends on the value specified for the StmtCallLimitBehavior property.

Data Source Methods

```
public Integer getStmtCallLimit()
```

```
public void setStmtCallLimit(Integer)
```

Default Value

0

Data Type

Integer

See also

StmtCallLimitBehavior on page 77

StmtCallLimitBehavior

Purpose

Specifies the behavior of the driver when the maximum web service call limit specified by the StmtCallLimit property is exceeded.

Valid Values

ReturnResults | ErrorAlways

Behavior

If set to ReturnResults, the driver returns any partial results it received prior to the call limit being exceeded. The driver generates a warning that not all of the results were fetched.

If set to ErrorAlways, the driver generates an exception if the maximum web service call limit is exceeded.

Data Source Methods

```
public String getStmtCallLimitBehavior()
```

```
public void setStmtCallLimitBehavior(String)
```

Default Value

ErrorAlways

Data Type

String

See also

StmtCallLimit on page 76

TransactionMode

Purpose

Specifies how the driver handles manual transactions.

Valid Values

`NoTransactions` | `Ignore`

Behavior

If set to `NoTransactions`, the data source and the driver do not support transactions. Metadata indicates that the driver does not support transactions.

If set to `Ignore`, the data source does not support transactions and the driver always operates in auto-commit mode. Calls to set the driver to manual commit mode and to commit transactions are ignored. Calls to rollback a transaction cause the driver to throw an exception indicating that no transaction is started. Metadata indicates that the driver supports transactions and the ReadUncommitted transaction isolation level.

Data Source Methods

```
public String getTransactionMode()
```

```
public void setTransactionMode(String)
```

Default Value

`NoTransactions`

Data Type

String

User

Purpose

Specifies the user name that is used to connect to the instance.

Valid Values

`String`

where:

String

is a valid user name. The user name is case-insensitive.

Data Source Methods

`public String getUser()`

`public void setUser(String)`

Default Value

No default value

Data Type

String

WSCompressData

Purpose

Specifies whether the driver compresses data it sends to or receives from the instance.

Valid Values

`Compress` | `None`

Behavior

If set to `Compress`, the driver sends and receives compressed data to and from the instance.

If set to `None`, the driver sends and receives uncompressed data to and from the instance.

Notes

Setting WSCompressData to `None` can significantly degrade performance.

Data Source Methods

`public String getWSCompressData()`

`public void setWSCompressData(String)`

Default Value

`Compress`

Data Type

String

WSFetchSize

Purpose

Specifies the number of rows of data the driver attempts to fetch for each JDBC call.

Valid Values

x

where:

x

is a positive integer from 100 to 1000000 that defines a number of rows.

Behavior

If set to *x*, the driver attempts to fetch up to a maximum of the specified number of rows. Setting the value lower than the maximum can reduce the response time for returning the initial data; therefore, consider specifying smaller values for interactive applications.

Notes

WSFetchSize and FetchSize can be used to adjust the trade-off between throughput and response time. Smaller fetch sizes can improve the initial response time of the query. Larger fetch sizes can improve overall response times at the cost of additional memory.

Data Source Methods

```
public Integer getWSFetchSize()
```

```
public void setWSFetchSize(Integer)
```

Default Value

```
100000
```

Data Type

Integer

See also

FetchSize on page 57
Performance considerations on page 46

5

Supported SQL statements and extensions

The driver provides support for the SQL statements and the SQL 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
- Update
- Subqueries
- SQL expressions

Alter Session (EXT)

Purpose

Changes various attributes of a local or remote session. A local session maintains the state of the overall connection. A remote session maintains the state that pertains to a particular remote data source connection.

Syntax

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

where:

attribute_name

specifies the name of the attribute to be changed. Attributes apply to either local or remote sessions.

value

specifies the value for that attribute.

The following table lists the local and remote session attributes, and provides descriptions of each.

Table 9: Alter Session Attributes

Attribute Name	Session Type	Description
Current_Schema	Local	Sets the current schema for the local 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 local session. For example: `ALTER SESSION SET CURRENT_SCHEMA=SAP S/4HANA`
Stmt_Call_Limit	Local	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 Statement Call Limit 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 Statement Call Limit for the connection. The value specified must be a positive integer or `0`. The value `0` means that no call limit exists. For example: `ALTER SESSION SET STMT_CALL_LIMIT=150`

Ws_Call_Count	Remote	Resets the Web service call count of a remote session to the value specified. The value must be `0` or a positive integer. WS_Call_Count represents the total number of Web service calls made to the remote data source instance for the current session. For example:
		`ALTER SESSION SET s4hana.WS_CALL_COUNT=0`
		The current value of WS_Call_Count can be obtained by referring to the System_Remote_Sessions system table (see SYSTEM_REMOTE_SESSIONS Catalog Table for details). For example:
		`SELECT * from information_schema.system_remote_sessions WHERE session_id = cursessionid()`

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 definition and associated files.

Syntax

```
REFRESH MAP
```

Notes

- REFRESH MAP is an expensive query since it involves the discovery of native data.

Select

Purpose

Use the Select statement to fetch results from one or more tables.

Syntax

```
SELECT select_clause from_clause
[where_clause]
[groupby_clause]
[having_clause]
[{UNION [ALL | DISTINCT] |
 {MINUS [DISTINCT] | EXCEPT [DISTINCT]} |
 INTERSECT [DISTINCT]} select_statement]
[limit_clause]
```

where:

select_clause

specifies the columns from which results are to be returned by the query. See "Select clause" for a complete explanation.

from_clause

specifies one or more tables on which the other clauses in the query operate. See "From clause" for a complete explanation.

where_clause

is optional and restricts the results that are returned by the query. See "Where clause" for a complete explanation.

groupby_clause

is optional and allows query results to be aggregated in terms of groups. See "Group By clause" for a complete explanation.

having_clause

is optional and specifies conditions for groups of rows (for example, display only the departments that have salaries totaling more than $200,000). See "Having clause" for a complete explanation.

UNION

is an optional operator that combines the results of the left and right Select statements into a single result. See "Union operator" for a complete explanation.

INTERSECT

is an optional operator that returns a single result by keeping any distinct values from the results of the left and right Select statements. See "Intersect operator" for a complete explanation.

EXCEPT | MINUS

are synonymous optional operators that returns a single result by taking the results of the left Select statement and removing the results of the right Select statement. See "Except and Minus operators" for a complete explanation.

orderby_clause

is optional and sorts the results that are returned by the query. See "Order By clause" for a complete explanation.

limit_clause

is optional and places an upper bound on the number of rows returned in the result. See "Limit clause" for a complete explanation.

See also

Select clause on page 85
From clause on page 87
Where clause on page 88
Group By clause on page 89
Having clause on page 90
Union operator on page 90
Intersect operator on page 91
Except and Minus operators on page 92
Order By clause on page 93
Limit clause on page 93

Select clause

Purpose

Use the Select clause to specify with a list of column expressions that identify columns of values that you want to retrieve or an asterisk (*) to retrieve the value of all columns.

Syntax

```
SELECT [{LIMIT offsetnumber | TOP number}] [ALL | DISTINCT] {* | column_expression
[[AS] column_alias] [,column_expression [[AS] column_alias], ...]}
```

where:

`LIMIT offset number`

creates the result set for the Select statement first and then discards the first number of rows specified by *offset* and returns the number of remaining rows specified by *number*. To not discard any of the rows, specify `0` for *offset*, for example, `LIMIT 0` *number*. To discard the first *offset* number of rows and return all the remaining rows, specify `0` for *number*, for example, `LIMIT` *offset*`0`.

`TOP` *number*

is equivalent to `LIMIT 0`*number*.

column_expression

can be simply a column name (for example, `last_name`). More complex expressions may include mathematical operations or string manipulation (for example, `salary * 1.05`). See "SQL expressions" for details. *column_expression* can also include aggregate functions. See "Aggregate functions" for details.

column_alias

can be used to give the column a descriptive name. For example, to assign the alias department to the column dep:

```
SELECT dep AS department FROM emp
```

`DISTINCT`

eliminates duplicate rows from the result of a query. This operator can precede the first column expression. For example:

```
SELECT DISTINCT dep FROM emp
```

Notes

- Separate multiple column expressions with commas (for example, `SELECT last_name, first_name, hire_date`).
- Column names can be prefixed with the table name or table alias. For example, `SELECT emp.last_name or e.last_name`, where `e` is the alias for the table `emp`.
- NULL values are not treated as distinct from each other. The default behavior is that all result rows be returned, which can be made explicit with the keyword `ALL`.

See also

SQL expressions on page 97
Aggregate functions on page 86

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 following table lists supported aggregate functions.

Note: Doubly nested aggregates, such as `SUM(COUNT(col1))`, are currently not permitted by the driver.

Table 10: Aggregate Functions

Aggregate	Returns
AVG	The average of the values in a numeric column expression. For example, `AVG(salary)` returns the average of all salary column values.
COUNT	The number of values in any field expression. For example, `COUNT(name)` returns the number of name values. When using `COUNT` with a field name, `COUNT` returns the number of non-NULL column values. A special example is `COUNT(*)`, which returns the number of rows in the set, including rows with NULL values. **Note:** The driver does not support `COUNT(DISTINCT *)`. For example, `SELECT COUNT(DISTINCT *) FROM mytable` results in a syntax error.
MAX	The maximum value in any column expression. For example, `MAX(salary)` returns the maximum salary column value.
MIN	The minimum value in any column expression. For example, `MIN(salary)` returns the minimum salary column value.
SUM	The total of the values in a numeric column expression. For example, `SUM(salary)` returns the sum of all salary column values.

Example

The following example uses the `COUNT`, `MAX`, and `AVG` aggregate functions:

```
SELECT
     COUNT(amount) AS numOpportunities,
     MAX(amount) AS maxAmount,
     AVG(amount) AS avgAmount
FROM opportunity o INNER JOIN user u
     ON o.ownerId = u.id
WHERE o.isClosed = 'false' AND
     u.name = 'MyName'
```

From clause

Purpose

The From clause indicates the tables to be used in the Select statement.

Syntax

```
FROM table_name [table_alias] [,...]
```

where:

table_name

is the name of a table or a subquery. Multiple tables define an implicit inner join among those tables. Multiple table names must be separated by a comma. For example:

```
SELECT * FROM emp, dep
```

Subqueries can be used instead of table names. Subqueries must be enclosed in parentheses. See "Subquery in a From clause" for an example.

table_alias

is a name used to refer to a table in the rest of the Select statement. When you specify an alias for a table, you can prefix all column names of that table with the table alias.

Example

The following example specifies two table aliases, `e` for `emp` and `d` for `dep`:

```
SELECT e.name, d.deptName
FROM emp e, dep d
WHERE e.deptId = d.id
```

table_alias is a name used to refer to a table in the rest of the Select statement. When you specify an alias for a table, you can prefix all column names of that table with the table alias. For example, given the table specification:

```
FROM emp E
```

you may refer to the last_name field as E.last_name. Table aliases must be used if the Select statement joins a table to itself. For example:

```
SELECT * FROM emp E, emp F WHERE E.mgr_id = F.emp_id
```

The equal sign (=) includes only matching rows in the results.

See also

Subquery in a From clause on page 88

Join in a From clause

Purpose

You can use a Join as a way to associate multiple tables within a Select statement. Joins may be either explicit or implicit. For example, the following is the example from the previous section restated as an explicit inner join:

```
SELECT * FROM emp INNER JOIN dep ON id=empId
SELECT e.name, d.deptName
FROM emp e INNER JOIN dep d ON e.deptId = d.id;
```

whereas the following is the same statement as an implicit inner join:

```
SELECT * FROM emp, dep WHERE emp.deptID=dep.id
```

Note: The `ON` clause in a join expression must evaluate to a true or false value.

Syntax

```
FROM table_name {RIGHT OUTER | INNER | LEFT OUTER | CROSS | FULL OUTER} JOIN table.key
 ON search-condition
```

Example

In this example, two tables are joined using `LEFT OUTER JOIN`. `T1`, the first table named includes nonmatching rows.

```
SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.key = T2.key
```

If you use a `CROSS JOIN`, no `ON` expression is allowed for the join.

Subquery in a From clause

Subqueries can be used in the From clause in place of table references (*`table_name`*).

Example

```
SELECT * FROM (SELECT * FROM emp WHERE sal > 10000) new_emp, dept WHERE
new_emp.deptno = dept.deptno
```

See also

Subqueries on page 95

Where clause

Purpose

Specifies the conditions that rows must meet to be retrieved.

Syntax

`WHERE` *`expr1 rel_operator expr2`*

where:

expr1

is either a column name, literal, or expression.

expr2

is either a column name, literal, expression, or subquery. Subqueries must be enclosed in parentheses.

rel_operator

is the relational operator that links the two expressions.

Example

The following Select statement retrieves the first and last names of employees that make at least $20,000.

```
SELECT last_name, first_name FROM emp WHERE salary >= 20000
```

See also

Subqueries on page 95
SQL expressions on page 97

Group By clause

Purpose

Specifies the names of one or more columns by which the returned values are grouped. This clause is used to return a set of aggregate values.

Syntax

```
GROUP BY column_expression [,...]
```

where:

column_expression

is either a column name or a SQL expression. Multiple values must be separated by a comma. If *column_expression* is a column name, it must match one of the column names specified in the Select clause. Also, the Group By clause must include all non-aggregate columns specified in the Select list.

Example

The following example totals the salaries in each department:

```
SELECT dept_id, sum(salary) FROM emp GROUP BY dept_id
```

This statement returns one row for each distinct department ID. Each row contains the department ID and the sum of the salaries of the employees in the department.

See also

Subqueries on page 95
SQL expressions on page 97

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" 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

Subqueries on page 95
SQL expressions on page 97

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

Subqueries on page 95
SQL expressions on page 97

Limit clause

Purpose

Places an upper bound on the number of rows returned in the result.

Syntax

```
LIMIT number_of_rows [OFFSET offset_number]
```

where:

number_of_rows

specifies a maximum number of rows in the result. A negative number indicates no upper bound.

OFFSET

specifies how many rows to skip at the beginning of the result set. *offset_number* is the number of rows to skip.

Notes

- In a compound query, the Limit clause can appear only on the final Select statement. The limit is applied to the entire query, not to the individual Select statement to which it is attached.

Example

The following example returns a maximum of 20 rows.

```
SELECT last_name, first_name FROM emp WHERE salary > 20000 ORDER BY dept_idc LIMIT
20
```

Update

Purpose

An Update statement changes the value of columns in the selected rows of a table.

Syntax

```
UPDATE table_name SET column_name = expression
[, column_name = expression] [WHERE conditions]
```

table_name

is the name of the table for which you want to update values.

column_name

is the name of a column, the value of which is to be changed. Multiple column values can be changed in a single statement.

expression

is the new value for the column. The expression can be a constant value or a subquery that returns a single value. Subqueries must be enclosed in parentheses.

Example A

The following example changes every record that meets the conditions in the Where clause. In this case, the salary and exempt status are changed for all employees having the employee ID E10001. Because employee IDs are unique in the emp table, only one record is updated.

```
UPDATE emp SET salary=32000, exempt=1
WHERE emp_id = 'E10001'
```

Example B

The following example uses a subquery. In this example, the salary is changed to the average salary in the company for the employee having employee ID E10001.

```
UPDATE emp SET salary = (SELECT avg(salary) FROM emp)
WHERE emp_id = 'E10001'
```

Notes

- To enable Insert, Update, and Delete, set the ReadOnly connection option to `false`.
- A Where clause can be used to restrict which rows are updated.

See also

Where clause on page 88

Subqueries

A query is an operation that retrieves data from one or more tables or views. In this reference, a top-level query is called a Select statement, and a query nested within a Select statement is called a subquery.

A subquery is a query expression that appears in the body of another expression such as a Select, an Update, or a Delete statement. In the following example, the second Select statement is a subquery:

```
SELECT * FROM emp WHERE deptno IN (SELECT deptno FROM dept)
```

IN predicate

Purpose

The In predicate specifies a set of values against which to compare a result set. If the values are being compared against a subquery, only a single column result set is returned.

Syntax

value [NOT] IN (*value1, value2,...*)

OR

value [NOT] IN (*subquery*)

Example

```
SELECT * FROM emp WHERE deptno IN
(SELECT deptno FROM dept WHERE dname <> 'Sales')
```

EXISTS predicate

Purpose

The Exists predicate is true only if the cardinality of the subquery is greater than 0; otherwise, it is false.

Syntax

EXISTS (*subquery*)

Example

```
SELECT empno, ename, deptno FROM emp e WHERE EXISTS
(SELECT deptno FROM dept WHERE e.deptno = dept.deptno)
```

UNIQUE predicate

Purpose

The Unique predicate is used to determine whether duplicate rows exist in a virtual table (one returned from a subquery).

Syntax

```
UNIQUE (subquery)
```

Example

```
SELECT * FROM dept d WHERE UNIQUE
(SELECT deptno FROM emp e WHERE e.deptno = d.deptno)
```

Correlated subqueries

Purpose

A correlated subquery is a subquery that references a column from a table referred to in the parent statement. A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a Select, Update, or Delete statement.

A correlated subquery answers a multiple-part question in which the answer depends on the value in each row processed by the parent statement. For example, you can use a correlated subquery to determine which employees earn more than the average salaries for their departments. In this case, the correlated subquery specifically computes the average salary for each department.

Syntax

```
SELECT select_list
   FROM table1 t_alias1
   WHERE expr rel_operator
   (SELECT column_list
       FROM table2 t_alias2
       WHERE t_alias1.columnrel_operatort_alias2.column)
UPDATE table1 t_alias1
   SET column =
   (SELECT expr
       FROM table2 t_alias2
        WHERE t_alias1.column = t_alias2.column)
DELETE FROM table1 t_alias1
   WHERE column rel_operator
   (SELECT expr
       FROM table2 t_alias2
       WHERE t_alias1.column = t_alias2.column)
```

Notes

- Correlated column names in correlated subqueries must be explicitly qualified with the table name of the parent.

Example A

The following statement returns data about employees whose salaries exceed their department average. This statement assigns an alias to emp, the table containing the salary information, and then uses the alias in a correlated subquery:

```
SELECT deptno, ename, sal FROM emp x WHERE sal >
  (SELECT AVG(sal) FROM emp WHERE x.deptno = deptno)
  ORDER BY deptno
```

Example B

This is an example of a correlated subquery that returns row values:

```
SELECT * FROM dept "outer" WHERE 'manager' IN
  (SELECT managername FROM emp
  WHERE "outer".deptno = emp.deptno)
```

Example C

This is an example of finding the department number (deptno) with multiple employees:

```
SELECT * FROM dept main WHERE 1 <
  (SELECT COUNT(*) FROM emp WHERE deptno = main.deptno)
```

Example D

This is an example of correlating a table with itself:

```
SELECT deptno, ename, sal FROM emp x WHERE sal >
  (SELECT AVG(sal) FROM emp WHERE x.deptno = deptno)
```

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

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 11: Literal Syntax Examples

SQL Type	Literal Syntax	Example
BIGINT	*n* where *n* 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'*date*'	`'2010-05-21'`
DATETIME	TIMESTAMP'*ts*'	`'2010-05-21 18:33:05.025'`

SQL Type	Literal Syntax	Example

DECIMAL	*n.f*
	where:
	n
	is the integral part
	f
	is the fractional part
		`0.25`
		`3.1415`
		`-7.48`

DOUBLE	*n.f*E*x* where: *n* is the integral part *f* is the fractional part *x* is the exponent	`1.2E0` or `2.5E40` or `-3.45E2` or `5.67E-4`
INTEGER	*n* where *n* is a valid integer value in the range of the INTEGER data type	`12` or `-34` or `0`
LONGVARBINARY	'*hex_value*'	`'000482ff'`
LONGVARCHAR	'*value*'	`'This is a string literal'`
TIME	TIME'*time*'	`'2010-05-21 18:33:05.025'`
VARCHAR	'*value*'	`'This is a string literal'`

Character string literals

Text specifies a character string literal. A character string literal must be enclosed in single quotation marks. To represent one single quotation mark within a literal, you must enter two single quotation marks. When the data in the fields is returned to the client, trailing blanks are stripped.

A character string literal can have a maximum length of 32 KB, that is, (32*1024) bytes.

Example

```
'Hello'
'Jim''s friend is Joe'
```

Numeric literals

Unquoted numeric values are treated as numeric literals. If the unquoted numeric value contains a decimal point or exponent, it is treated as a real literal; otherwise, it is treated as an integer literal.

Example
```
+1894.1204
```

Binary literals

Binary literals are represented with single quotation marks. The valid characters in a binary literal are 0-9, a-f, and A-F.

Example
```
'00af123d'
```

Date/Time literals

Date and time literal values are enclosed in single quotion 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.

Note: Numeric operators are restricted to numeric types. Numeric operators do not support non-numeric types.

Unary operator

A unary operator operates on only one operand.

operator operand

Binary operator

A binary operator operates on two operands.

operand1 operator operand2

If an operator is given a null operand, the result is always null. The only operator that does not follow this rule is concatenation (||).

Arithmetic operators

You can use an arithmetic operator in an expression to negate, add, subtract, multiply, and divide numeric values. The result of this operation is also a numeric value. The + and - operators are also supported in date/time fields to allow date arithmetic. The following table lists the supported arithmetic operators.

Table 12: 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 13: 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 14: Comparison Operators

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

Operator	Purpose	Example
><	"Greater than" and "less than" tests.	`SELECT * FROM emp WHERE sal > 1500 SELECT * FROM emp WHERE sal < 1500`
>=<=	"Greater than or equal to" and "less than or equal to" tests.	`SELECT * FROM emp WHERE sal >= 1500 SELECT * FROM emp WHERE sal <= 1500`
LIKE	% and _ wildcards can be used to search for a pattern in a column. The percent sign denotes zero, one, or multiple characters, while the underscore denotes a single character. The right-hand side of a LIKE expression must evaluate to a string or binary.	`SELECT * FROM emp WHERE ENAME LIKE 'J%'`

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`
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 15: 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 16: Operator Precedence

Precedence	Operator
1	+ (Positive), - (Negative)
2	*(Multiply), / (Division)
3	+ (Add), - (Subtract)
4	|| (Concatenate)
5	=, >, <, >=, <=, <>, != (Comparison operators)
6	NOT, IN, LIKE

Precedence	Operator
7	AND
8	OR

Example A

The query in the following example returns employee records for which the department number is 1 or 2 and the salary is greater than $1000:

```
SELECT * FROM emp WHERE (deptno = 1 OR deptno = 2) AND sal > 1000
```

Because parenthetical expressions are forced to be evaluated first, the OR operation takes precedence over AND.

Example B

In the following example, the query returns records for all the employees in department 1, but only employees whose salary is greater than $1000 in department 2.

```
SELECT * FROM emp WHERE deptno = 1 OR deptno = 2 AND sal > 1000
```

The AND operator takes precedence over OR, so that the search condition in the example is equivalent to the expression `deptno = 1 OR (deptno = 2 AND sal > 1000)`.

Functions

The driver supports a number of functions that you can use in expressions, including String, Numeric, Timedate, and System functions.

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

Conditions

A condition specifies a combination of one or more expressions and logical operators that evaluates to either TRUE, FALSE, or UNKNOWN. You can use a condition in the Where clause of the Delete, Select, and Update statements; and in the Having clauses of Select statements. The following describes supported conditions.

Table 17: Conditions

Condition	Description
Simple comparison	Specifies a comparison with expressions or subquery results. `= , !=, <>, < , >, <=, <=`
Group comparison	Specifies a comparison with any or all members in a list or subquery. `[= , !=, <>, < , >, <=, <=] [ANY, ALL, SOME]`

Condition	Description
Membership	Tests for membership in a list or subquery. `[NOT] IN`
Range	Tests for inclusion in a range. `[NOT] BETWEEN`
NULL	Tests for nulls. `IS NULL, IS NOT NULL`
EXISTS	Tests for existence of rows in a subquery. `[NOT] EXISTS`
LIKE	Specifies a test involving pattern matching. `[NOT] LIKE`
Compound	Specifies a combination of other conditions. `CONDITION [AND/OR] CONDITION`

6

Introduction to the SAP S/4HANA data model

The S/4HANA data model is defined using a collection of standard JSON documents that contain the data, identifiers, and object relationships for a given service. These documents are stored on URL endpoints that are accessible using sets of proprietary OData API calls. To expose S/4HANA resources to SQL applications, the driver maps S/4HANA endpoints to one schema and each schema has a set of relational parent and child tables. The following sections describe the relational tables exposed by the driver along with their corresponding S/4HANA OData API call.

For details, see the following topics:

- A_BUSINESSPARTNERCONTACT
- A_BUSINESSPARTNERROLE
- A_BUSINESSPARTNERTAXNUMBER
- A_CUSTOMER
- A_CUSTOMERCOMPANY
- A_CUSTOMERDUNNING
- A_CUSTOMERSALESAREA
- A_CUSTOMERSALESAREATAX
- A_CUSTOMERWITHHOLDINGTAX
- A_CUSTSALESPARTNERFUNC
- A_SUPPLIER
- A_SUPPLIERCOMPANY
- A_SUPPLIERDUNNING
- A_SUPPLIERPARTNERFUNC
- A_SUPPLIERPURCHASINGORG
- A_SUPPLIERWITHHOLDINGTAX

A_ADDRESSEMAILADDRESS

Columns

The A_ADDRESSEMAILADDRESS table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
ADDRESSID*	STRING(10)
PERSON*	STRING(10)
ORDINALNUMBER*	STRING(3)
EMAILADDRESS	STRING(241)
ISDEFAULTEMAILADDRESS	BOOLEAN
SEARCHEMAILADDRESS	STRING(20)

A_ADDRESSFAXNUMBER

Columns

The A_ADDRESSFAXNUMBER table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
ADDRESSID*	STRING(10)
PERSON*	STRING(10)
ORDINALNUMBER*	STRING(3)
FAXCOUNTRY	STRING(3)
FAXNUMBER	STRING(30)
FAXNUMBEREXTENSION	STRING(10)
INTERNATIONALFAXNUMBER	STRING(30)
ISDEFAULTFAXNUMBER	BOOLEAN

A_ADDRESSHOMEPAGEURL

Columns

The A_ADDRESSHOMEPAGEURL table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
ADDRESSID*	STRING(10)
PERSON*	STRING(10)
ORDINALNUMBER*	STRING(3)
VALIDITYSTARTDATE*	DATETIME(0)
ISDEFAULTURLADDRESS*	BOOLEAN
SEARCHURLADDRESS	STRING(50)

Column Name	Data Type
URLFIELDLENGTH	INT16
WEBSITEURL	STRING(2048)

A_ADDRESSPHONENUMBER

Columns

The A_ADDRESSPHONENUMBER table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
ADDRESSID*	STRING(10)
PERSON*	STRING(10)
ORDINALNUMBER*	STRING(3)
DESTINATIONLOCATIONCOUNTRY	STRING(3)
INTERNATIONALPHONENUMBER	STRING(30)
ISDEFAULTPHONENUMBER	BOOLEAN
PHONENUMBER	STRING(30)
PHONENUMBEREXTENSION	STRING(10)
PHONENUMBERTYPE	STRING(1)

A_BPCONTACTTOADDRESS

Columns

The A_BPCONTACTTOADDRESS table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
RELATIONSHIPNUMBER*	STRING(12)
BUSINESSPARTNERCOMPANY*	STRING(10)
BUSINESSPARTNERPERSON*	STRING(10)

Column Name	Data Type
VALIDITYENDDATE*	DATETIME(0)
ADDRESSID*	STRING(10)
ADDITIONALSTREETPREFIXNAME	STRING(40)
ADDITIONALSTREETSUFFIXNAME	STRING(40)
ADDRESSNUMBER	STRING(10)
ADDRESSTIMEZONE	STRING(6)
CAREOFNAME	STRING(40)
CITYCODE	STRING(12)
CITYNAME	STRING(40)
COMPANYPOSTALCODE	STRING(10)
COUNTRY	STRING(3)
COUNTY	STRING(40)
DELIVERYSERVICENUMBER	STRING(10)
DELIVERYSERVICETYPECODE	STRING(4)
DISTRICT	STRING(40)
FORMOFADDRESS	STRING(4)
FULLNAME	STRING(80)
HOMECITYNAME	STRING(40)
HOUSENUMBER	STRING(10)
HOUSENUMBERSUPPLEMENTTEXT	STRING(10)
LANGUAGE	STRING(2)
PERSON	STRING(10)
POBOX	STRING(10)
POBOXDEVIATINGCITYNAME	STRING(40)
POBOXDEVIATINGCOUNTRY	STRING(3)
POBOXDEVIATINGREGION	STRING(3)

Column Name	Data Type
POBOXISWITHOUTNUMBER	BOOLEAN
POBOXLOBBYNAME	STRING(40)
POBOXPOSTALCODE	STRING(10)
POSTALCODE	STRING(10)
PRFRDCOMMMEDIUMTYPE	STRING(3)
REGION	STRING(3)
STREETNAME	STRING(60)
STREETPREFIXNAME	STRING(40)
STREETSUFFIXNAME	STRING(40)
TAXJURISDICTION	STRING(15)
TRANSPORTZONE	STRING(10)

A_BPCONTACTTOFUNCANDDEPT

Columns

The A_BPCONTACTTOFUNCANDDEPT table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
RELATIONSHIPNUMBER*	STRING(12)
BUSINESSPARTNERCOMPANY*	STRING(10)
BUSINESSPARTNERPERSON*	STRING(10)
VALIDITYENDDATE*	DATETIME(0)
CONTACTPERSONDEPARTMENT	STRING(4)
CONTACTPERSONFUNCTION	STRING(4)
EMAILADDRESS	STRING(241)
FAXNUMBER	STRING(30)
FAXNUMBEREXTENSION	STRING(10)

Column Name	Data Type
PHONENUMBER	STRING(30)
PHONENUMBEREXTENSION	STRING(10)
RELATIONSHIPCATEGORY	STRING(6)

A_BUPAADDRESSUSAGE

Columns

The A_BUPAADDRESSUSAGE table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
BUSINESSPARTNER*	STRING(10)
VALIDITYENDDATE*	DATETIME(0)
ADDRESSUSAGE*	STRING(10)
ADDRESSID*	STRING(10)
AUTHORIZATIONGROUP	STRING(4)
STANDARDUSAGE	BOOLEAN
VALIDITYSTARTDATE	DATETIME(0)

A_BUPAIDENTIFICATION

Columns

The A_BUPAIDENTIFICATION table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
BUSINESSPARTNER*	STRING(10)
BPIDENTIFICATIONTYPE*	STRING(6)
BPIDENTIFICATIONNUMBER*	STRING(60)
AUTHORIZATIONGROUP	STRING(4)

Column Name	Data Type
BPIDENTIFICATIONENTRYDATE	DATETIME(0)
BPIDNNMBRISSUINGINSTITUTE	STRING(40)
COUNTRY	STRING(3)
REGION	STRING(3)
VALIDITYENDDATE	DATETIME(0)
VALIDITYSTARTDATE	DATETIME(0)

A_BUPAINDUSTRY

Columns

The A_BUPAINDUSTRY table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
INDUSTRYSECTOR*	STRING(10)
INDUSTRYSYSTEMTYPE*	STRING(4)
BUSINESSPARTNER*	STRING(10)
INDUSTRYKEYDESCRIPTION	STRING(100)
ISSTANDARDINDUSTRY	STRING(1)

A_BUSINESSPARTNER

Columns

The A_BUSINESSPARTNER table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
BUSINESSPARTNER*	STRING(10)
ACADEMICTITLE	STRING(4)
ADDITIONALLASTNAME	STRING(40)

Column Name	Data Type
AUTHORIZATIONGROUP	STRING(4)
BIRTHDATE	DATETIME(0)
BUSINESSPARTNERCATEGORY	STRING(1)
BUSINESSPARTNERFULLNAME	STRING(81)
BUSINESSPARTNERGROUPING	STRING(4)
BUSINESSPARTNERIDBYEXTSYSTEM	STRING(20)
BUSINESSPARTNERISBLOCKED	BOOLEAN
BUSINESSPARTNERNAME	STRING(81)
BUSINESSPARTNERTYPE	STRING(4)
BUSINESSPARTNERUUID	GUID(36)
CORRESPONDENCELANGUAGE	STRING(2)
CREATEDBYUSER	STRING(12)
CREATIONDATE	DATETIME(0)
CREATIONTIME	TIMEOFDAY(0)
CUSTOMER	STRING(10)
ETAG	STRING(26)
FIRSTNAME	STRING(40)
FORMOFADDRESS	STRING(4)
GROUPBUSINESSPARTNERNAME1	STRING(40)
GROUPBUSINESSPARTNERNAME2	STRING(40)
INDEPENDENTADDRESSID	STRING(10)
INDUSTRY	STRING(10)
INTERNATIONALLOCATIONNUMBER1	STRING(7)
INTERNATIONALLOCATIONNUMBER2	STRING(5)
INTERNATIONALLOCATIONNUMBER3	STRING(1)
ISFEMALE	BOOLEAN

Column Name	Data Type
ISMALE	BOOLEAN
ISMARKEDFORARCHIVING	BOOLEAN
ISNATURALPERSON	STRING(1)
ISSEXUNKNOWN	BOOLEAN
LANGUAGE	STRING(2)
LASTCHANGEDATE	DATETIME(0)
LASTCHANGEDBYUSER	STRING(12)
LASTCHANGETIME	TIMEOFDAY(0)
LASTNAME	STRING(40)
LEGALFORM	STRING(2)
MIDDLENAME	STRING(40)
NAMECOUNTRY	STRING(3)
NAMEFORMAT	STRING(2)
ORGANIZATIONBPNAME1	STRING(40)
ORGANIZATIONBPNAME2	STRING(40)
ORGANIZATIONBPNAME3	STRING(40)
ORGANIZATIONBPNAME4	STRING(40)
ORGANIZATIONFOUNDATIONDATE	DATETIME(0)
ORGANIZATIONLIQUIDATIONDATE	DATETIME(0)
PERSONFULLNAME	STRING(80)
PERSONNUMBER	STRING(10)
SEARCHTERM1	STRING(20)
SUPPLIER	STRING(10)
TRADINGPARTNER	STRING(6)

A_BUSINESSPARTNERADDRESS

Columns

The A_BUSINESSPARTNERADDRESS table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
BUSINESSPARTNER*	STRING(10)
ADDRESSID*	STRING(10)
ADDITIONALSTREETPREFIXNAME	STRING(40)
ADDITIONALSTREETSUFFIXNAME	STRING(40)
ADDRESSIDBYEXTERNALSYSTEM	STRING(20)
ADDRESSTIMEZONE	STRING(6)
ADDRESSUUID	GUID(36)
AUTHORIZATIONGROUP	STRING(4)
CAREOFNAME	STRING(40)
CITYCODE	STRING(12)
CITYNAME	STRING(40)
COMPANYPOSTALCODE	STRING(10)
COUNTRY	STRING(3)
COUNTY	STRING(40)
DELIVERYSERVICENUMBER	STRING(10)
DELIVERYSERVICETYPECODE	STRING(4)
DISTRICT	STRING(40)
FORMOFADDRESS	STRING(4)
FULLNAME	STRING(80)
HOMECITYNAME	STRING(40)
HOUSENUMBER	STRING(10)

Column Name	Data Type
HOUSENUMBERSUPPLEMENTTEXT	STRING(10)
LANGUAGE	STRING(2)
PERSON	STRING(10)
POBOX	STRING(10)
POBOXDEVIATINGCITYNAME	STRING(40)
POBOXDEVIATINGCOUNTRY	STRING(3)
POBOXDEVIATINGREGION	STRING(3)
POBOXISWITHOUTNUMBER	BOOLEAN
POBOXLOBBYNAME	STRING(40)
POBOXPOSTALCODE	STRING(10)
POSTALCODE	STRING(10)
PRFRDCOMMMEDIUMTYPE	STRING(3)
REGION	STRING(3)
STREETNAME	STRING(60)
STREETPREFIXNAME	STRING(40)
STREETSUFFIXNAME	STRING(40)
TAXJURISDICTION	STRING(15)
TRANSPORTZONE	STRING(10)
VALIDITYENDDATE	DATETIME(0)
VALIDITYSTARTDATE	DATETIME(0)

A_BUSINESSPARTNERBANK

Columns

The A_BUSINESSPARTNERBANK table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
BUSINESSPARTNER*	STRING(10)
BANKIDENTIFICATION*	STRING(4)
AUTHORIZATIONGROUP	STRING(4)
BANKACCOUNT	STRING(18)
BANKACCOUNTHOLDERNAME	STRING(60)
BANKACCOUNTNAME	STRING(40)
BANKACCOUNTREFERENCETEXT	STRING(20)
BANKCONTROLKEY	STRING(2)
BANKCOUNTRYKEY	STRING(3)
BANKNAME	STRING(60)
BANKNUMBER	STRING(15)
CITYNAME	STRING(35)
COLLECTIONAUTHIND	BOOLEAN
IBAN	STRING(34)
IBANVALIDITYSTARTDATE	DATETIME(0)
SWIFTCODE	STRING(11)
VALIDITYENDDATE	DATETIME(0)
VALIDITYSTARTDATE	DATETIME(0)

A_BUSINESSPARTNERCONTACT

Columns

The A_BUSINESSPARTNERCONTACT table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
RELATIONSHIPNUMBER*	STRING(12)
BUSINESSPARTNERCOMPANY*	STRING(10)

Column Name	Data Type
BUSINESSPARTNERPERSON*	STRING(10)
VALIDITYENDDATE*	DATETIME(0)
ISSTANDARDRELATIONSHIP	BOOLEAN
RELATIONSHIPCATEGORY	STRING(6)
VALIDITYSTARTDATE	DATETIME(0)

A_BUSINESSPARTNERROLE

Columns

The A_BUSINESSPARTNERROLE table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
BUSINESSPARTNER*	STRING(10)
BUSINESSPARTNERROLE*	STRING(6)
AUTHORIZATIONGROUP	STRING(4)
VALIDFROM	DATETIME(0)
VALIDTO	DATETIME(0)

A_BUSINESSPARTNERTAXNUMBER

Columns

The A_BUSINESSPARTNERTAXNUMBER table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
BUSINESSPARTNER*	STRING(10)
BPTAXTYPE*	STRING(4)
AUTHORIZATIONGROUP	STRING(4)

Column Name	Data Type
BPTAXLONGNUMBER	STRING(60)
BPTAXNUMBER	STRING(20)

A_CUSTOMER

Columns

The A_CUSTOMER table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
CUSTOMER*	STRING(10)
AUTHORIZATIONGROUP	STRING(4)
BILLINGISBLOCKEDFORCUSTOMER	STRING(2)
CREATEDBYUSER	STRING(12)
CREATIONDATE	DATETIME(0)
CUSTOMERACCOUNTGROUP	STRING(4)
CUSTOMERCLASSIFICATION	STRING(2)
CUSTOMERCORPORATEGROUP	STRING(10)
CUSTOMERFULLNAME	STRING(220)
CUSTOMERNAME	STRING(80)
DELETIONINDICATOR	BOOLEAN
DELIVERYISBLOCKED	STRING(2)
FISCALADDRESS	STRING(10)
INDUSTRY	STRING(4)
INDUSTRYCODE1	STRING(10)
INDUSTRYCODE2	STRING(10)
INDUSTRYCODE3	STRING(10)
INDUSTRYCODE4	STRING(10)

Column Name	Data Type
INDUSTRYCODE5	STRING(10)
INTERNATIONALLOCATIONNUMBER1	STRING(7)
NFPARTNERISNATURALPERSON	STRING(1)
NIELSENREGION	STRING(2)
ORDERISBLOCKEDFORCUSTOMER	STRING(2)
POSTINGISBLOCKED	BOOLEAN
RESPONSIBLETYPE	STRING(2)
SUPPLIER	STRING(10)
TAXNUMBER1	STRING(16)
TAXNUMBER2	STRING(11)
TAXNUMBER3	STRING(18)
TAXNUMBER4	STRING(18)
TAXNUMBER5	STRING(60)
TAXNUMBERTYPE	STRING(2)
VATREGISTRATION	STRING(20)

A_CUSTOMERCOMPANY

Columns

The A_CUSTOMERCOMPANY table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
CUSTOMER*	STRING(10)
COMPANYCODE*	STRING(4)
ACCOUNTBYCUSTOMER	STRING(12)
ACCOUNTINGCLERK	STRING(2)
ACCOUNTINGCLERKFAXNUMBER	STRING(31)

Column Name	Data Type
ACCOUNTINGCLERKINTERNETADDRESS	STRING(130)
ACCOUNTINGCLERKPHONENUMBER	STRING(30)
ALTERNATIVEPAYERACCOUNT	STRING(10)
APARTOLERANCEGROUP	STRING(4)
AUTHORIZATIONGROUP	STRING(4)
COLLECTIVEINVOICEVARIANT	STRING(1)
CUSTOMERACCOUNTGROUP	STRING(4)
CUSTOMERACCOUNTNOTE	STRING(30)
CUSTOMERHEADOFFICE	STRING(10)
CUSTOMERSUPPLIERCLEARINGISUSED	BOOLEAN
DELETIONINDICATOR	BOOLEAN
HOUSEBANK	STRING(5)
INTERESTCALCULATIONCODE	STRING(2)
INTERESTCALCULATIONDATE	DATETIME(0)
INTRSTCALCFREQUENCYINMONTHS	STRING(2)
ISTOBELOCALLYPROCESSED	BOOLEAN
ITEMISTOBEPAIDSEPARATELY	BOOLEAN
LAYOUTSORTINGRULE	STRING(3)
PAYMENTBLOCKINGREASON	STRING(1)
PAYMENTMETHODSLIST	STRING(10)
PAYMENTTERMS	STRING(4)
PAYTADVICEISSENTBYEDI	BOOLEAN
PHYSICALINVENTORYBLOCKIND	BOOLEAN
RECONCILIATIONACCOUNT	STRING(10)
RECORDPAYMENTHISTORYINDICATOR	BOOLEAN
USERATCUSTOMER	STRING(15)

A_CUSTOMERDUNNING

Columns

The A_CUSTOMERDUNNING table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
CUSTOMER*	STRING(10)
COMPANYCODE*	STRING(4)
DUNNINGAREA*	STRING(2)
AUTHORIZATIONGROUP	STRING(4)
CUSTOMERACCOUNTGROUP	STRING(4)
DUNNINGBLOCK	STRING(1)
DUNNINGCLERK	STRING(2)
DUNNINGLEVEL	STRING(1)
DUNNINGPROCEDURE	STRING(4)
DUNNINGRECIPIENT	STRING(10)
LASTDUNNEDON	DATETIME(0)
LEGDUNNINGPROCEDUREON	DATETIME(0)

A_CUSTOMERSALESAREA

Columns

The A_CUSTOMERSALESAREA table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
CUSTOMER*	STRING(10)
SALESORGANIZATION*	STRING(4)
DISTRIBUTIONCHANNEL*	STRING(2)

Column Name	Data Type
DIVISION*	STRING(2)
ACCOUNTBYCUSTOMER	STRING(12)
AUTHORIZATIONGROUP	STRING(4)
BILLINGISBLOCKEDFORCUSTOMER	STRING(2)
COMPLETEDELIVERYISDEFINED	BOOLEAN
CURRENCY	STRING(5)
CUSTOMERABCCLASSIFICATION	STRING(2)
CUSTOMERACCOUNTASSIGNMENTGROUP	STRING(2)
CUSTOMERACCOUNTGROUP	STRING(4)
CUSTOMERGROUP	STRING(2)
CUSTOMERPAYMENTTERMS	STRING(4)
CUSTOMERPRICEGROUP	STRING(2)
CUSTOMERPRICINGPROCEDURE	STRING(2)
DELETIONINDICATOR	BOOLEAN
DELIVERYISBLOCKEDFORCUSTOMER	STRING(2)
DELIVERYPRIORITY	STRING(2)
EXCHANGERATETYPE	STRING(4)
INCOTERMSCLASSIFICATION	STRING(3)
INCOTERMSLOCATION1	STRING(70)
INCOTERMSLOCATION2	STRING(70)
INCOTERMSTRANSFERLOCATION	STRING(28)
INCOTERMSVERSION	STRING(4)
INVOICEDATE	STRING(2)
INVOICELISTSCHEDULE	STRING(2)
ITEMORDERPROBABILITYINPERCENT	STRING(3)
ORDERCOMBINATIONISALLOWED	BOOLEAN

Column Name	Data Type
ORDERISBLOCKEDFORCUSTOMER	STRING(2)
PARTIALDELIVERYISALLOWED	STRING(1)
PRICELISTTYPE	STRING(2)
SALESDISTRICT	STRING(6)
SALESGROUP	STRING(3)
SALESOFFICE	STRING(4)
SHIPPINGCONDITION	STRING(2)
SUPPLYINGPLANT	STRING(4)

A_CUSTOMERSALESAREATAX

Columns

The A_CUSTOMERSALESAREATAX table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
CUSTOMER*	STRING(10)
SALESORGANIZATION*	STRING(4)
DISTRIBUTIONCHANNEL*	STRING(2)
DIVISION*	STRING(2)
DEPARTURECOUNTRY*	STRING(3)
CUSTOMERTAXCATEGORY*	STRING(4)
CUSTOMERTAXCLASSIFICATION	STRING(1)

A_CUSTOMERWITHHOLDINGTAX

Columns

The A_CUSTOMERWITHHOLDINGTAX table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
CUSTOMER*	STRING(10)
COMPANYCODE*	STRING(4)
WITHHOLDINGTAXTYPE*	STRING(2)
AUTHORIZATIONGROUP	STRING(4)
EXEMPTIONDATEBEGIN	DATETIME(0)
EXEMPTIONDATEEND	DATETIME(0)
EXEMPTIONREASON	STRING(2)
OBLIGATIONDATEBEGIN	DATETIME(0)
OBLIGATIONDATEEND	DATETIME(0)
WITHHOLDINGTAXAGENT	BOOLEAN
WITHHOLDINGTAXCERTIFICATE	STRING(25)
WITHHOLDINGTAXCODE	STRING(2)
WITHHOLDINGTAXEXMPTPERCENT	DECIMAL(5, 2)
WITHHOLDINGTAXNUMBER	STRING(16)

A_CUSTSALESPARTNERFUNC

Columns

The A_CUSTSALESPARTNERFUNC table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
CUSTOMER*	STRING(10)
SALESORGANIZATION*	STRING(4)
DISTRIBUTIONCHANNEL*	STRING(2)
DIVISION*	STRING(2)
PARTNERCOUNTER*	STRING(3)
PARTNERFUNCTION*	STRING(2)

Column Name	Data Type
AUTHORIZATIONGROUP	STRING(4)
BPCUSTOMERNUMBER	STRING(10)
CUSTOMERPARTNERDESCRIPTION	STRING(30)
DEFAULTPARTNER	BOOLEAN

A_SUPPLIER

Columns

The A_SUPPLIER table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
SUPPLIER*	STRING(10)
ALTERNATIVEPAYEEACCOUNTNUMBER	STRING(10)
AUTHORIZATIONGROUP	STRING(4)
BIRTHDATE	DATETIME(0)
CONCATENATEDINTERNATIONALLOCNO	STRING(20)
CREATEDBYUSER	STRING(12)
CREATIONDATE	DATETIME(0)
CUSTOMER	STRING(10)
DELETIONINDICATOR	BOOLEAN
FISCALADDRESS	STRING(10)
INDUSTRY	STRING(4)
INTERNATIONALLOCATIONNUMBER1	STRING(7)
INTERNATIONALLOCATIONNUMBER2	STRING(5)
INTERNATIONALLOCATIONNUMBER3	STRING(1)
ISNATURALPERSON	STRING(1)
PAYMENTISBLOCKEDFORSUPPLIER	BOOLEAN

Column Name	Data Type
POSTINGISBLOCKED	BOOLEAN
PURCHASINGISBLOCKED	BOOLEAN
RESPONSIBLETYPE	STRING(2)
SUPLRPROOFOFDELIVRLVTCODE	STRING(1)
SUPLRQLTYINPROCMTCERTFNVALIDTO	DATETIME(0)
SUPLRQUALITYMANAGEMENTSYSTEM	STRING(4)
SUPPLIERACCOUNTGROUP	STRING(4)
SUPPLIERCORPORATEGROUP	STRING(10)
SUPPLIERFULLNAME	STRING(220)
SUPPLIERNAME	STRING(80)
SUPPLIERPROCUREMENTBLOCK	STRING(2)
TAXNUMBER1	STRING(16)
TAXNUMBER2	STRING(11)
TAXNUMBER3	STRING(18)
TAXNUMBER4	STRING(18)
TAXNUMBER5	STRING(60)
TAXNUMBERRESPONSIBLE	STRING(18)
TAXNUMBERTYPE	STRING(2)
VATREGISTRATION	STRING(20)

A_SUPPLIERCOMPANY

Columns

The A_SUPPLIERCOMPANY table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
SUPPLIER*	STRING(10)

Column Name	Data Type
COMPANYCODE*	STRING(4)
ACCOUNTINGCLERK	STRING(2)
ACCOUNTINGCLERKFAXNUMBER	STRING(31)
ACCOUNTINGCLERKPHONENUMBER	STRING(30)
ALTERNATIVEPAYEE	STRING(10)
APARTOLERANCEGROUP	STRING(4)
AUTHORIZATIONGROUP	STRING(4)
BILLOFEXCHLMTAMTINCOCODECRCY	DECIMAL(14, 3)
CASHPLANNINGGROUP	STRING(10)
CHECKPAIDDURATIONINDAYS	DECIMAL(3, 0)
CLEARCUSTOMERSUPPLIER	BOOLEAN
COMPANYCODENAME	STRING(25)
CURRENCY	STRING(5)
DELETIONINDICATOR	BOOLEAN
HOUSEBANK	STRING(5)
INTERESTCALCULATIONCODE	STRING(2)
INTERESTCALCULATIONDATE	DATETIME(0)
INTRSTCALCFREQUENCYINMONTHS	STRING(2)
ISTOBECHECKEDFORDUPLICATES	BOOLEAN
ISTOBELOCALLYPROCESSED	BOOLEAN
ITEMISTOBEPAIDSEPARATELY	BOOLEAN
LAYOUTSORTINGRULE	STRING(3)
PAYMENTBLOCKINGREASON	STRING(1)
PAYMENTISTOBESENTBYEDI	BOOLEAN
PAYMENTMETHODSLIST	STRING(10)
PAYMENTTERMS	STRING(4)

Column Name	Data Type
RECONCILIATIONACCOUNT	STRING(10)
SUPPLIERACCOUNTGROUP	STRING(4)
SUPPLIERACCOUNTNOTE	STRING(30)
SUPPLIERCERTIFICATIONDATE	DATETIME(0)
SUPPLIERCLERK	STRING(15)
SUPPLIERCLERKIDBYSUPPLIER	STRING(12)
SUPPLIERCLERKURL	STRING(130)
SUPPLIERHEADOFFICE	STRING(10)
SUPPLIERISBLOCKEDFORPOSTING	BOOLEAN
WITHHOLDINGTAXCOUNTRY	STRING(3)

A_SUPPLIERDUNNING

Columns

The A_SUPPLIERDUNNING table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
SUPPLIER*	STRING(10)
COMPANYCODE*	STRING(4)
DUNNINGAREA*	STRING(2)
AUTHORIZATIONGROUP	STRING(4)
DUNNINGBLOCK	STRING(1)
DUNNINGCLERK	STRING(2)
DUNNINGLEVEL	STRING(1)
DUNNINGPROCEDURE	STRING(4)
DUNNINGRECIPIENT	STRING(10)
LASTDUNNEDON	DATETIME(0)

Column Name	Data Type
LEGDUNNINGPROCEDUREON	DATETIME(0)
SUPPLIERACCOUNTGROUP	STRING(4)

A_SUPPLIERPARTNERFUNC

Columns

The A_SUPPLIERPARTNERFUNC table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
SUPPLIER*	STRING(10)
PURCHASINGORGANIZATION*	STRING(4)
SUPPLIERSUBRANGE*	STRING(6)
PLANT*	STRING(4)
PARTNERFUNCTION*	STRING(2)
PARTNERCOUNTER*	STRING(3)
AUTHORIZATIONGROUP	STRING(4)
CREATEDBYUSER	STRING(12)
CREATIONDATE	DATETIME(0)
DEFAULTPARTNER	BOOLEAN
REFERENCESUPPLIER	STRING(10)

A_SUPPLIERPURCHASINGORG

Columns

The A_SUPPLIERPURCHASINGORG table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
SUPPLIER*	STRING(10)

Column Name	Data Type
PURCHASINGORGANIZATION*	STRING(4)
AUTHORIZATIONGROUP	STRING(4)
CALCULATIONSCHEMAGROUPCODE	STRING(2)
DELETIONINDICATOR	BOOLEAN
INCOTERMSCLASSIFICATION	STRING(3)
INCOTERMSLOCATION1	STRING(70)
INCOTERMSLOCATION2	STRING(70)
INCOTERMSTRANSFERLOCATION	STRING(28)
INCOTERMSVERSION	STRING(4)
INVOICEISGOODSRECEIPTBASED	BOOLEAN
MATERIALPLANNEDDELIVERYDURN	DECIMAL(3, 0)
MINIMUMORDERAMOUNT	DECIMAL(14, 3)
PAYMENTTERMS	STRING(4)
PRICINGDATECONTROL	STRING(1)
PURCHASEORDERCURRENCY	STRING(5)
PURCHASINGGROUP	STRING(3)
PURCHASINGISBLOCKEDFORSUPPLIER	BOOLEAN
PURORDAUTOGENERATIONISALLOWED	BOOLEAN
SHIPPINGCONDITION	STRING(2)
SUPPLIERABCCLASSIFICATIONCODE	STRING(1)
SUPPLIERACCOUNTGROUP	STRING(4)
SUPPLIERPHONENUMBER	STRING(16)
SUPPLIERRESPSALESPERSONNAME	STRING(30)

A_SUPPLIERWITHHOLDINGTAX

Columns

The A_SUPPLIERWITHHOLDINGTAX table contains the following columns. Columns marked with an asterisk comprise the primary key.

Column Name	Data Type
SUPPLIER*	STRING(10)
COMPANYCODE*	STRING(4)
WITHHOLDINGTAXTYPE*	STRING(2)
AUTHORIZATIONGROUP	STRING(4)
EXEMPTIONDATEBEGIN	DATETIME(0)
EXEMPTIONDATEEND	DATETIME(0)
EXEMPTIONREASON	STRING(2)
ISWITHHOLDINGTAXSUBJECT	BOOLEAN
RECIPIENTTYPE	STRING(2)
WITHHOLDINGTAXCERTIFICATE	STRING(25)
WITHHOLDINGTAXCODE	STRING(2)
WITHHOLDINGTAXEXMPTPERCENT	DECIMAL(5, 2)
WITHHOLDINGTAXNUMBER	STRING(16)