Progress DataDirect for JDBC for Aha!

User's Guide

Release 6.0.0

Copyright

Table of Contents

1

Welcome to the Progress DataDirect for JDBC for Aha! Driver

The Progress® DataDirect® for JDBC™ for Aha!™ driver (Aha! driver) supports SQL read-only access for JDBC applications to Aha!. To support SQL access to Aha! services, the driver creates a relational map of the Aha! data model and translates SQL statements to Aha! REST API requests. In addition, the driver employs a SQL engine component that provides support to SQL constructs unavailable to Aha! 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.

For an overview of the relational tables exposed by the driver and their corresponding API calls, see Introduction to the Aha! data model.

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-aha.

For details, see the following topics:

- What's new in this release?
- Requirements
- Installing and setting up the driver
- Driver and DataSource classes
- Connection URL examples
- Data types

- SQL escape sequences
- 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

Highlights of 6.0.0 Release

- The driver supports SQL read-only access to Aha!. See Supported SQL statements and extensions on page 91 for details.
- The driver supports JDBC core functions. For details, refer to "JDBC support" in the *Progress DataDirect for JDBC Drivers Reference*.
- The driver supports Aha! data types through data type inference. See Data types on page 18 and getTypeInfo() on page 19 for details.
- The driver supports Aha! custom fields, including many fields added through third-party plug-ins
- The driver supports OAuth 2.0 authentication. See OAuth 2.0 authentication on page 47 for supported grant types and further details.
- The driver supports basic authentication.
- The driver supports the handling of large result sets with paging, and the FetchSize on page 65 and WSFetchSize on page 86 connection properties.
- The driver includes the Aha! 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 39 and Testing connections and queries on page 40.

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\aha.jar
     ```
   - **UNIX/LINUX Example**

     ```
     CLASSPATH=.:/opt/Progress/DataDirect/JDBC/lib/60/aha.jar
     ```
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 either API key authentication or OAuth 2.0 authorization code grant authentication.

     URL parameter (API key) authentication

     ```
     jdbc:datadirect:aha://company.aha.io;authenticationMethod=UrlParameter;
     securitytoken=api_key;
     ```

OAuth 2.0 authorization token grant

```
jdbc:datadirect:aha://company.aha.io;authenticationMethod=OAuth2;
oauthCode=auth_code;clientId=client_id;clientSecret=client_secret;
redirectUri=https://lvh.me/app_callback.html
```

You can also generate a connection string using the Progress DataDirect Aha! 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 Aha! Configuration Manager: The Aha! 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.
   - 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. This tool provides a method of quickly testing your driver in an environment that does not support GUIs.
   - 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.aha.AhaDriver

DataSource class:

com.ddtek.jdbcx.aha.AhaDataSource

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 39

Basic authentication

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

```
jdbc:datadirect:aha://servername;User=username;Password=password;[property=value[;...]];
```

where:

servername

specifies the URL of the Aha! service to which you want to issue requests. For example, `company.aha.io`.

user

specifies the user name that is used to connect to the Aha! service. For example, `jsmith@company.com`.

password

specifies the password used to connect to your Aha! 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 string includes the properties required for connecting with basic authentication.

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:aha://company.aha.io;user=jsmith@company.com;password=secret;");
```

See also

Basic authentication on page 46
Connection property descriptions on page 51

URL parameter (API key) authentication

This string includes the properties used to connect with URL parameter (API key) authentication.

```
jdbc:datadirect:aha:servername;AuthenticationMethod=UrlParameter;SecurityToken=security_token;
[property=value[;...]];
```

where:

servername

specifies the base URL of the Aha! service to which you want to issue requests. For example, `company.aha.io`.

security_token

specifies the API key required to make a connection to your endpoint. API keys can be generated through the Aha! user interface. Refer to the Aha! documentation for more information.

property=value

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

The following example connection string demonstrates the properties required for connecting with URL parameter authentication.

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:aha://company.aha.io;authenticationMethod=UrlParameter;
    securitytoken=1234abc5d6789efg;");
```

See also

Connection property descriptions on page 51
URL parameter (API key) authentication on page 47

OAuth 2.0 access token flow

Note: To use OAuth2 authentication, you must first register your application with Aha!. Refer to the Aha! documentation for more information.

Typically, an OAuth 2.0 grant type or access flow is handled by the application. However, in some scenarios, you may need to secure an access token using external processes. In those instances, you can use the access token flow to access the service manually.

Note: Access tokens are temporary and must be replaced to maintain the session without interruption. The life of an access token is typically one hour.

The following string includes the properties used to connect with the OAuth 2.0 access token flow.

```
jdbc:datadirect:aha:servername;AuthenticationMethod=OAuth2;
AccessToken=access_token;[property=value[;...]];
```

where:

serviceurl

specifies the base URL of the Aha! service to which you want to issue requests. For example, `company.aha.io`.

access_token

specifies the access token required to authenticate to Aha!. This property allows you to set the access token manually.

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 the OAuth 2.0 access token flow.

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:aha://company.aha.io;authenticationMethod=OAuth2;
    accesstoken=abc12cd34efg5678h9ij87klm6543no32pqr10;");
```

See also

Connection property descriptions on page 51
Access token flow on page 47

OAuth 2.0 authorization code grant

Note: To use OAuth2 authentication, you must first register your application with Aha!. Refer to the Aha! documentation for more information.

The authentication flow for the authorization code grant exchanges the authorization code and client credentials for the access token at the location specified by the RedirectURI.

```
jdbc:datadirect:aha:servername;AuthenticationMethod=OAuth2;oauthCode=auth_code;
ClientID=client_id;ClientSecret=client_secret;RedirectURI=redirect_uri;
[property=value[;...]];
```

where:

servername

specifies the base URL of the Aha! service to which you want to issue requests. For example, `company.aha.io`.

auth_code

specifies the authorization code that is exchanged for the access token.

client_id

specifies the client ID key for your application when authenticating with OAuth 2.0.

client_secret

specifies the client secret for your application when authenticating with OAuth 2.0.

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

redirect_uri

specifies the endpoint that the client is returned to after authenticating with the service. This value is the same as the one specified in the OAuth application.

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 the OAuth 2.0 client credentials grant.

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:aha://company.aha.io;authenticationMethod=OAuth2;
    oauthCode=abc12cd34efg5678h9ij87klm6543no32pqr10;
    clientId=cd34efg5678h9ij87klm6543no32pqr10st987;
    clientSecret=098zyx765wvu432tsr123qpo456;
    redirectUri=https://lvh.me/app_callback.html;");
```

See also

Connection property descriptions on page 51
Authorization code grant on page 48

Proxy server

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

```
jdbc:datadirect:aha:servername;ProxyHost=proxy_host;ProxyPassword=proxy_password;
ProxyPort=proxy_port;ProxyUser=proxy_user;AuthenticationMethod=method;
user=username;password=password;[property=value[;...]];
```

where:

servername

specifies the base URL of the Aha! service to which you want to issue requests. For example, `company.aha.io`.

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`, which means the port number is determined by the setting of the ProxyHost property. If an http URL is specified, the default port number is `80`. If an https URL is specified, the default is `443`.

proxy_user

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

user

specifies the user name that is used to connect to the Aha! service. For example, `jsmith@company.com`.

password

specifies the password used to connect to your Aha! 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 string includes the properties required for using a proxy server with OAuth 2.0 refresh token grant authentication.

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:aha:company.aha.io;ProxyHost=pserver;ProxyPassword=secret;
    ProxyPort=808;ProxyUser=jsmith;user=jsmith@company.com;password=secret;");
```

See also

Connection property descriptions on page 51

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.

Table 1: Aha Data Types

Aha Data Type	JDBC Data Type
BigInt	BIGINT
Binary	BINARY
Bit	BIT
Boolean	BOOLEAN
Char	CHAR
Date	DATE
Decimal	DECIMAL
Double	DOUBLE
Float	FLOAT
GUID	CHAR
Integer	INTEGER
JSON	VARCHAR
LongVarBinary	LONGVARBINARY
LongVarChar	LONGVARCHAR
NVarChar	NVARCHAR
SmallInt	SMALLINT
Time	TIME
Timestamp	TIMESTAMP
TinyInt	TINYINT

Aha Data Type	JDBC Data Type
VarBinary	VARBINARY
VarChar	VARCHAR

See also

getTypeInfo() on page 19

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 = BigInt 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 = BigInt 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 = Binary AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = -2 (BINARY) 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 = 32767 SEARCHABLE = 3 SQL_DATA_TYPE = 60 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = Bit AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = -7 (BIT) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = X' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = Bit MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 1 SEARCHABLE = 3 SQL_DATA_TYPE = 14 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 = Char AUTO_INCREMENT = FALSE CASE_SENSITIVE = TRUE CREATE_PARAMS = NULL DATA_TYPE = 1 (CHAR) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = Char MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 255 SEARCHABLE = 3 SQL_DATA_TYPE = 1 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = Date AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 91 (DATE) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = DATE ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = Date MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 10 SEARCHABLE = 3 SQL_DATA_TYPE = 91 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 = 1000	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 1000 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 = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 2 PRECISION = 53 SEARCHABLE = 3 SQL_DATA_TYPE = 8 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = FALSE

TYPE_NAME = Float AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 6 (FLOAT) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = Float MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 2 PRECISION = 24 SEARCHABLE = 3 SQL_DATA_TYPE = 6 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = FALSE
TYPE_NAME = GUID AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE 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 = Integer AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 4 (INTEGER) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = Integer MAXIMUM_SCALE = 0	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 = JSON AUTO_INCREMENT = FALSE CASE_SENSITIVE = TRUE CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = JSON MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 16777215 SEARCHABLE = 3 SQL_DATA_TYPE = 12 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = LongVarBinary AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = -4 (LONGVARBINARY) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = X' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = LongVarBinary MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 16777215 SEARCHABLE = 0 SQL_DATA_TYPE = -4 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = LongVarChar AUTO_INCREMENT = FALSE CASE_SENSITIVE = TRUE CREATE_PARAMS = NULL DATA_TYPE = -1 (LONGVARCHAR) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = LongVarChar MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 16777215 SEARCHABLE = 0 SQL_DATA_TYPE = -1 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = NVarChar AUTO_INCREMENT = FALSE CASE_SENSITIVE = TRUE CREATE_PARAMS = NULL DATA_TYPE = -9 (NVARCHAR) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = NVarChar MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 32767 SEARCHABLE = 3 SQL_DATA_TYPE = -9 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = SmallInt 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 = SmallInt 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 = Time 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 = Time MAXIMUM_SCALE = 9	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 12 SEARCHABLE = 3 SQL_DATA_TYPE = 92 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = Timestamp 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 = Timestamp MAXIMUM_SCALE = 9	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 23 SEARCHABLE = 3 SQL_DATA_TYPE = 93 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = TinyInt 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 = TinyInt 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 = VarBinary 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 = VarBinary MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 16777215 SEARCHABLE = 3 SQL_DATA_TYPE = 61 SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = VarChar AUTO_INCREMENT = FALSE CASE_SENSITIVE = TRUE CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = VarChar MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 32767 SEARCHABLE = 3 SQL_DATA_TYPE = 12 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		

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:

- Interactive SQL for JDBC (JDBCISQL)
- Tableau
- 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.aha.AhaDriver
```

3. Type connect followed by the connection URL for the driver; then, press **Enter**. For example:

```
connect jdbc:datadirect:aha://mycompany.aha.io;user=jsmith@progress.com;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 FEATURES;
```

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.

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:

```
aha.jar
```

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 either NTLM authentication or OAuth 2.0 refresh token grant authentication.

     URL parameter (API key) authentication

     ```
     jdbc:datadirect:aha:servername;AuthenticationMethod=UrlParameter;SecurityToken=security_token;
     ```

     Note: See URL parameter (API key) authentication on page 47 for details.

     OAuth 2.0 authorization code grant

     ```
     jdbc:datadirect:aha://example.org;authenticationMethod=OAuth2;
          oauthCode=abc12cd34efg5678h9ij87klm6543no32pqr10;
          clientId=cd34efg5678h9ij87klm6543no32pqr10st987;
          clientSecret=098zyx765wvu432tsr123qpo456;
          redirectUri=https://lvh.me/app_callback.html;
     ```

     Note: See OAuth 2.0 authentication on page 47 for details on this and other supported grant types.

   - **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\aha.jar
   ```

   Linux

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

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

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

     ```
     Aha
     ```

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

     ```
     jdbc:datadirect:aha:ServiceURL=<server>
     ```

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

     ```
     com.ddtek.jdbcx.aha.AhaDataSource
     ```

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 either URL parameter authentication or OAuth 2.0 authorization grant authentication.

     Note: You can also generate connection strings using Aha! Configuration Manager. For more information, see Generating connection URLs with the Configuration Manager on page 39.

     URL parameter (API key) authentication

     ```
     jdbc:datadirect:aha://example.org;authenticationMethod=UrlParameter;
     securitytoken=api_key;
     ```

Note: See URL parameter (API key) authentication on page 47 for details.

OAuth 2.0 authorization code grant

```
jdbc:datadirect:aha://example.org;authenticationMethod=OAuth2;
oauthCode=auth_code;clientId=client_id;clientSecret=client_secret;
redirectUri=https://lvh.me/app_callback.html
```

Note: See Authorization code grant on page 48 for details on this and other supported grant types.

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 FEATURES table:

   ```
   SELECT * FROM FEATURES
   ```

   To select the comments for a specified user :

   ```
   SELECT BODY FROM ISSUES WHERE USER_ID = <user_id>
   ```

   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.

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/aha.jar`

Windows Example

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

UNIX Example

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

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.

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:aha://example.org;user=jsmith@example.org;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 OAuth 2.0 authorization code grant authentication.

Connection URL Syntax

The connection URL takes the following form:

```
jdbc:datadirect:aha:servername;AuthenticationMethod=OAuth2;
ClientID=client_id;ClientSecret=client_secret;RedirectURI=redirect_uri;
[property=value[;...]];
```

where:

`serviceurl`

specifies the base URL of the Aha! service to which you want to issue requests. For example, `example.org`.

client_id

specifies the client ID key for your application when authenticating with OAuth 2.0.

client_secret

specifies the client secret for your application when authenticating with OAuth 2.0.

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

redirect_uri

specifies the endpoint that the client is returned to after authenticating with the service.

property=value

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

Connection URL Example

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:aha://example.org;authenticationMethod=OAuth2;
    oauthCode=abc12cd34efg5678h9ij87klm6543no32pqr10;
    clientId=cd34efg5678h9ij87klm6543no32pqr10st987;
    clientSecret=098zyx765wvu432tsr123qpo456;
    redirectUri=https://lvh.me/app_callback.html;");
```

See also

Connection property descriptions on page 51
Connection URL examples on page 13
Authorization code grant on page 48

Generating connection URLs with the Configuration Manager

The driver includes a browser-based tool, Progress DataDirect Aha! 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.

To generate a connection URL:

1. Open the Aha! 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 aha.jar
   ```

   The Aha! Configuration Manager opens in your default web browser.

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 Aha! Configuration Manager.

To test your connection and query:

1. Open the Aha! 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 aha.jar
   ```

   The Aha! Configuration Manager opens in your default web browser.

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 Aha! 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 42

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.aha.AhaDataSource;
```

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 OAuth 2.0 authorization code grant.

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")`.

```
AutoRESTDataSource mds = new AhaDataSource();
mds.setDescription("My Aha Data Source");
mds.setServerName("example.org");
mds.setAuthenticationMethod("OAuth2");
mds.setOAuthCode("abc12cd34efg5678h9ij87klm6543no32pqr10");
mds.setClientID("cd34efg5678h9ij87klm6543no32pqr10st987");
mds.setClientSecret("098zyx765wvu432tsr123qpo456");
mds.setRedirectURI("https://lvh.me/app_callback.html");
```

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 AhaDataSource)
{
AhaDataSource jmds = (AhaDataSource) ds;
System.out.println("description=" + jmds.getDescription());
System.out.println("servername=" + jmds.getServiceURL());
System.out.println("authenticationmethod=" + jmds.getServiceURL());
System.out.println("oauthcode=" + jmds.getServiceURL());
System.out.println("clientid=" + jmds.getServiceURL());
System.out.println("clientsecret=" + jmds.getServiceURL());
System.out.println("redirecturi=" + jmds.getServiceURL());
...
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.

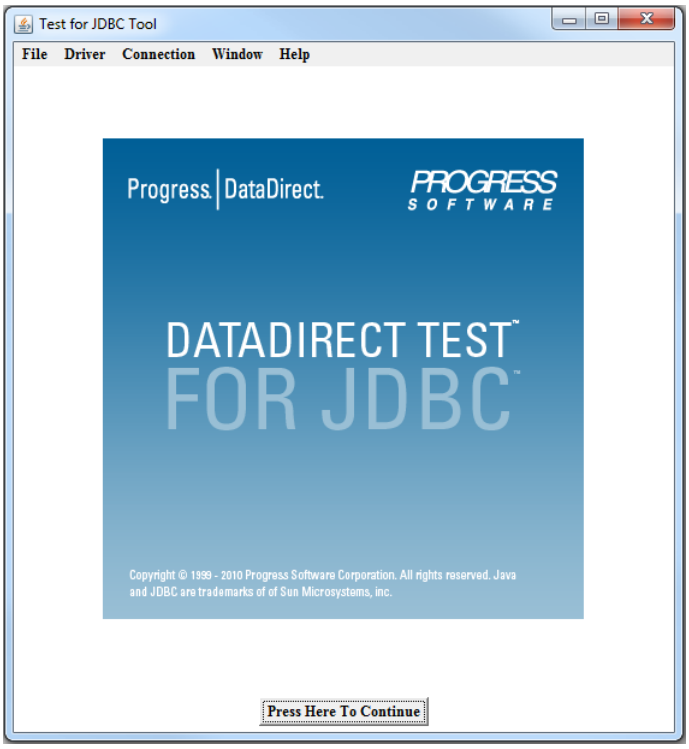
Take the following steps to establish a connection.

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.

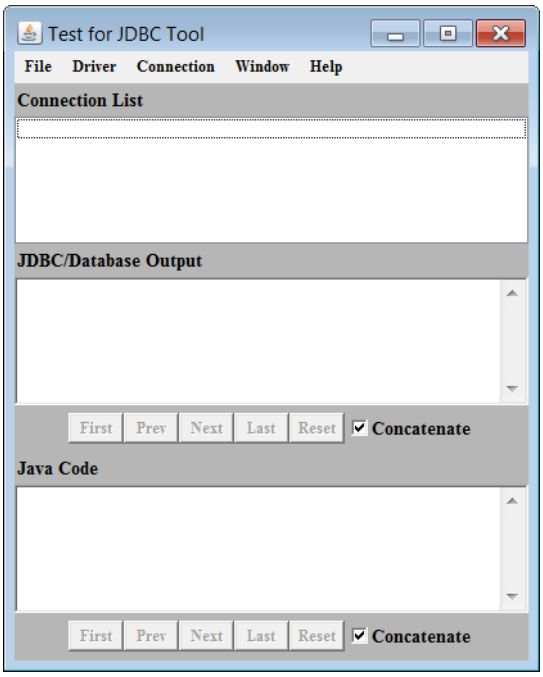
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

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



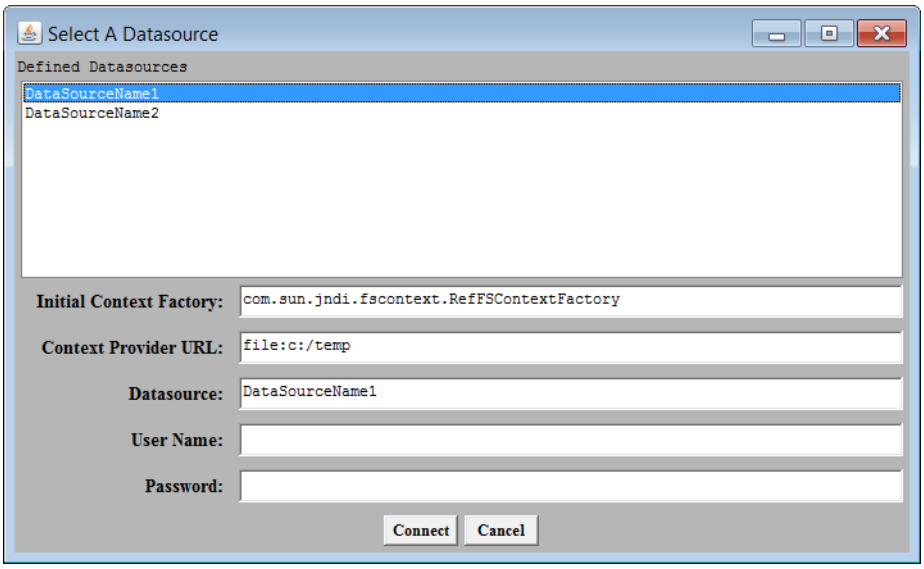
3. Click **Press Here to Continue**.

   The main dialog appears:



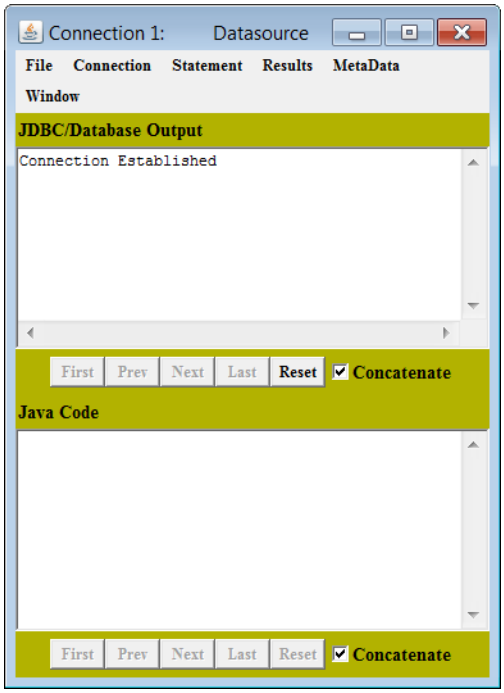
4. From the menu bar, select **Connection > Connect to DB via Data Source**.

   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 Authentication* authenticates using the specified user IDs, passwords, and HTTP headers.
- *URL Parameter Authentication* authenticates by passing security tokens using URLs. In some scenarios, the REST services may also authenticate the user ID.
- *OAuth 2.0 Authentication* authenticates using OAuth 2.0 authentication flows.

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

See also

AuthenticationMethod on page 58

Basic authentication

To configure the driver to use basic authentication:

- Set the ServerName property to specify the base URL of the Aha! service to which you want to issue requests. For example, `company.aha.io`.
- Set the AuthenticationMethod property to `Basic` (the default).
- 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 using a REST file with basic authentication enabled and AuthHeader set to the default.

For a connection URL:

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:aha://company.aha.io;user=jsmith@company.com;password=secret;");
```

For a data source:

```
AutoRESTDataSource mds = new AhaDataSource();
mds.setDescription("My Aha Data Source");
mds.setServerName("company.aha.io");
```

See also

Connection property descriptions on page 51

URL parameter (API key) authentication

To configure the driver to use URL parameter (API key) authentication:

- Set the ServerName property to specify the base URL of the Aha! service to which you want to issue requests. For example, `company.aha.io`.
- Set the AuthenticationMethod property to `UrlParameter`.
- Set the SecurityToken to specify the API key required to make a connection to your endpoint. API keys can be generated through the Aha! user interface. Refer to the Aha! documentation for more information.
- Optionally, specify values for any additional properties you want to configure.

The following examples demonstrate a string with URL parameter authentication enabled.

For a connection URL:

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:aha://company.aha.io;authenticationMethod=UrlParameter;
  securitytoken=1234abc5d6789efg;");
```

For a data source:

```
AutoRESTDataSource mds = new AhaDataSource();
mds.setDescription("My Aha Data Source");
mds.setServerName("company.aha.io");
mds.setAuthenticationMethod("UrlParameter");
mds.setSecurityToken("1234abc5d6789efg");
```

See also

Connection property descriptions on page 51

OAuth 2.0 authentication

The driver supports the OAuth 2.0 grant types described in this section. The OAuth 2.0 properties you must specify depend on the grant type used in your environment. If you are unsure of the grant type or its requirements, contact your system administrator.

- Access token flow on page 47
- Authorization code grant on page 48

See also

Connection property descriptions on page 51

Access token flow

Note: To use OAuth2 authentication, you must first register your application with Aha!. Refer to the Aha! documentation for more information.

The access token authentication flow passes the access token directly from the client to the Aha! service for authentication. Unlike other grant types, authentication credentials, such as authorization codes, are not exchanged in return for the access code. Instead, the access token has been obtained from sources external to the flow and specified using the AccessToken property.

Note: Access tokens are temporary and must be replaced to maintain the session without interruption. The life of an access token is typically one hour.

To configure the driver to use an access token flow:

- Set the ServerName property to the base URL of the Aha! instance to which you want to issue requests. For example, `company.aha.io`.
- Set the AuthenticationMethod property to `OAuth2`.
- Set the AccessToken property to the value of the access token obtained from external sources.

The following examples show the connection information required to establish a session using the access token flow.

Connection URL

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:aha://company.aha.io;authenticationMethod=OAuth2;
    accesstoken=abc12cd34efg5678h9ij87klm6543no32pqr10;");
```

Data Source

```
AhaDataSource mds = new AhaDataSource();
mds.setDescription("My Aha Data Source");
mds.setServerName("company.aha.io");
mds.setAuthenticationMethod("OAuth2");
mds.setAccessToken("abc12cd34efg5678h9ij87klm6543no32pqr10");
```

See also

OAuth 2.0 properties on page 52

Authorization code grant

Note: To use OAuth2 authentication, you must first register your application with Aha!. Refer to the Aha! documentation for more information.

The authorization code grant is a commonly used authentication flow for web and native applications. It provides secure connections by requiring multiple points of authentication before permitting access to data. When using the authorization code flow, the application is first redirected to the location hosting the temporary authorization code and retrieves it. Next, after being redirected to the location specified by the RedirectURI property, the application exchanges the authorization code, client ID, and client secret for the access token.

To use an authorization code grant:

- The application should be configured to set the OAuthCode property to specify the authorization code that is exchanged for the access token.
- Set the ServerName property to specify the base URL of the Aha! service to which you want to issue requests. For example, `company.aha.io`.
- Set the AuthenticationMethod property to `OAuth2`.
- Set the ClientID property to specify the client ID key for your application.
- Set the ClientSecret property to specify the client ID key for your application.
- Set the RedirectURI to specify the endpoint to which the client is returned after third-party authorization. This value is the same as the one specified in the OAuth application.

The following example demonstrates using an authorization code grant:

Using a connection URL:

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:aha://company.aha.io;authenticationMethod=OAuth2;
    oauthCode=abc12cd34efg5678h9ij87klm6543no32pqr10;
    clientId=cd34efg5678h9ij87klm6543no32pqr10st987;
    clientSecret=098zyx765wvu432tsr123qpo456;
    redirectUri=https://lvh.me/app_callback.html;");
```

Using a data source:

```
AutoRESTDataSource mds = new AhaDataSource();
mds.setDescription("My Aha Data Source");
mds.setServerName("company.aha.io");
mds.setAuthenticationMethod("OAuth2");
mds.setOAuthCode("abc12cd34efg5678h9ij87klm6543no32pqr10");
mds.setClientID("cd34efg5678h9ij87klm6543no32pqr10st987");
mds.setClientSecret("098zyx765wvu432tsr123qpo456");
mds.setredirectURI("https://lvh.me/app_callback.html");
```

See also

Connection property descriptions on page 51

Performance considerations

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 `1000` (the maximum), the driver only needs to make 100 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.

ReadAhead: The ReadAhead property allows you to issue multiple fetch requests in parallel. By increasing this number, you can improve throughput and performance, but it does so with the following restrictions:

- Larger values can increase the load on the server, which may adversely affect performance of other users. If you encounter issues, decrease the value specified for this option.
- Larger values may result in unnecessary requests if your application only requires the first few rows of results. This may be an issue if your service places limits on the number of web requests.

Caution: Due to potential impacts to other users, we strongly recommend specifying only smaller values for the ReadAhead property. For example, in fully optimized environments, which include exceptionally fast connections and low latency, we recommend a setting of no higher than 5. For typical environments, this value is sometimes lower.

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

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

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: Connection property names are case-insensitive. For example, `Password` is the same as `password`.

Note: In a JDBC data source, string values must be enclosed in double quotation marks, for example, `setUser("abc@defcorp.com")`.

The following tables describe the connection properties by functionality:

- Basic connection properties
- URL parameters authentication properties
- OAuth 2.0 properties
- Mapping properties
- Proxy server properties
- Web service properties
- Data type properties
- Timeout properties
- Statement pooling properties

- Additional properties

Basic connection properties

The following table summarizes connection properties which are required to connect to a Aha! data source.

Table 4: Required Properties

Property	Data Source Method	Default
ServerName on page 81	`getServerName()` `setServerName(String)`	No default value
Password on page 72	`getProxyPassword()` `setProxyPassword(String)`	No default value
User on page 86	`getUser()` `setUser(String)`	No default value

URL parameter authentication properties

The following table summarizes connection properties used for URL parameter authentication.

Table 5: URL parameter authentication properties

Property	Data Source Method	Default
AuthenticationMethod on page 58	`getAuthenticationMethod()` `setAuthenticationMethod(String)`	`Basic`
SecurityToken on page 81	`getSecurityToken()` `setSecurityToken(String)`	No default value

OAuth 2.0 properties

The following table summarizes properties used for OAuth 2.0 authentication. The OAuth 2.0 properties you must specify depend on the grant type used in your environment. If you are unsure of the grant type or its requirements, contact your system administrator. For details on supported grant types, see OAuth 2.0 authentication on page 47.

Table 6: OAuth 2.0 properties

Property	Data Source Method	Default
AccessToken on page 58	`getAccessToken()` `setAccessToken(String)`	No default value
AuthenticationMethod on page 58	`getAuthenticationMethod()` `setAuthenticationMethod(String)`	`Basic`

Property	Data Source Method	Default
ClientID on page 59	`getClientId()` `setClientId(String)`	No default value
ClientSecret on page 60	`getClientSecret()` `setClientSecret(String)`	No default value
OAuthCode on page 71	`getOauthCode()` `setOauthCode(String)`	No default value
RedirectURI on page 77	`getOauthRedirUri()` `setOauthRedirUri(String)`	No default value

Mapping properties

The following table summarizes connection properties involved in mapping the Aha! data model to a SQL model.

Property	Data Source Method	Default
CreateMap on page 63	`getCreateMap()` `setCreateMap(String)`	`Session`
RefreshSchema on page 78	`getRefreshSchema()` `setRefreshSchema(Boolean)`	`false`
SchemaMap on page 79	`getSchemaMap()` `setSchemaMap(String)`	Default depends on the environment

Proxy server properties

The following table summarizes proxy server connection properties.

Table 7: Proxy Server Properties

Property	Data Source Method	Default
ProxyHost on page 72	`getProxyHost()` `setProxyHost(String)`	No default value
ProxyPassword on page 73	`getProxyPassword()` `setProxyPassword(String)`	No default value

Property	Data Source Method	Default
ProxyPort on page 74	`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 74	`getProxyUser()` `setProxyUser(String)`	No default value

Web service properties

The following table summarizes Web service connection properties, including those related to timeouts.

Table 8: Web Service Properties

Property	Data Source Method	Default
LoginTimeout on page 69	`getLoginTimeout()` `setLoginTimeout(Integer)`	`0` (no timeout)
StmtCallLimit on page 84	`getStmtCallLimit()` `setStmtCallLimit(Integer)`	`0` (no limit)
StmtCallLimitBehavior on page 85	`getStmtCallLimitBehavior()` `setStmtCallLimitBehavior(String)`	`errorAlways`
WSFetchSize on page 86	`getWsFetchSize()` `setWsFetchSize(Integer)`	`2000`
WSPoolSize on page 87	`getWsPoolSize()` `setWsPoolSize(Integer)`	`1`
WSRetryCount on page 88	`getWsRetryCount()` `setWsRetryCount(Integer)`	`5`
WSTimeout on page 89	`getWsTimeout()` `setWsTimeout(Integer)`	`120`

Data type properties

The following table summarizes connection properties which can be used to handle data types.

Table 9: Data Type Properties

Property	Data Source Method	Default
ConvertNull on page 62	`getConvertNull()` `setConvertNull(Boolean)`	`true`
JDBCBehavior on page 68	`getJdbcBehavior()` `setJdbcBehavior(Integer)`	`1` (data types described using JDBC 3.0-equivalent data types)

Timeout properties

The following table summarizes timeout connection properties.

Table 10: Timeout Properties

Property	Data Source Method	Default
LoginTimeout on page 69	`getLoginTimeout()` `setLoginTimeout(Integer)`	`0` (no timeout)
QueryTimeout on page 75	`getQueryTimeout()` `setQueryTimeout(Integer)`	`0` (no timeout)
WSRetryCount on page 88	`getWsRetryCount()` `setWsRetryCount(Integer)`	`5`
WSTimeout on page 89	`getWsTimeout()` `setWsTimeout(Integer)`	`120` (seconds)

Statement pooling properties

The following table summarizes statement pooling connection properties.

Table 11: Statement Pooling Properties

Property	Data Source Method	Default
ImportStatementPool on page 66	`getImportStatementPool()` `setImportStatementPool(String)`	No default value
MaxPooledStatements on page 70	`getMaxPooledStatements()` `setMaxPooledStatements(Integer)`	`0`
RegisterStatementPoolMonitorMBean on page 78	`getRegisterStatementPoolMonitorMbean()` `setRegisterStatementPoolMonitorMbean(Boolean)`	`false`

Additional properties

The following table summarizes additional connection properties.

Table 12: Additional Properties

Property	Data Source Method	Default
ConnectionRetryCount on page 60	`getConnectionRetryCount()` `setConnectionRetryCount(Integer)`	`5`
ConnectionRetryDelay on page 61	`getConnectionRetryDelay()` `setConnectionRetryDelay(Integer)`	`1`
DebugRecord on page 64	`getDebugRecord()` `setDebugRecord(String)`	No default value
FetchSize on page 65	`getFetchSize()` `setFetchSize(Integer)`	`100` (rows)
InitializationString on page 67	`getInitializationString()` `setInitializationString(String)`	No default value
InsensitiveResultSetBufferSize on page 67	`getInsensitiveResultsetBufferSize()` `setInsensitiveResultsetBufferSize(Integer)`	`2048` (KB of memory)
LogConfigFile on page 69	`getLogConfigFile()` `setLogConfigFile(String)`	`ddlogging.properties`
ReadAhead on page 76	`getReadAheadThreads()` `setReadAheadThreads(Integer)`	`0`
SpyAttributes on page 82	`getSpyAttributes()` `setSpyAttributes(String)`	No default value

For details, see the following topics:

- AccessToken
- AuthenticationMethod
- ClientID
- ClientSecret
- ConnectionRetryCount
- ConnectionRetryDelay

- ConvertNull
- CreateMap
- DebugRecord
- FetchSize
- ImportStatementPool
- InitializationString
- InsensitiveResultSetBufferSize
- JDBCBehavior
- LogConfigFile
- LoginTimeout
- MaxPooledStatements
- OAuthCode
- Password
- ProxyHost
- ProxyPassword
- ProxyPort
- ProxyUser
- QueryTimeout
- ReadAhead
- RedirectURI
- RefreshSchema
- RegisterStatementPoolMonitorMBean
- SchemaMap
- SecurityToken
- ServerName
- SpyAttributes
- StmtCallLimit
- StmtCallLimitBehavior
- User
- WSFetchSize
- WSPoolSize
- WSRetryCount
- WSTimeout

AccessToken

Purpose

Specifies the access token used to authenticate to Aha! with OAuth 2.0 enabled. Typically, this property is configured by the application; however, in some scenarios, you may need to secure a token using external processes. In those instances, you can also use this property to set the access token manually.

Valid Values

String

where:

String

is an access token you have obtained from the authentication service.

Notes

- Access tokens expire ten minutes after generation. Once connected, the access token remains valid till the session is disconnected.
- See "OAuth 2.0 authentication" for examples and more information.

Data Source Methods

```
public String getAccessToken()
```

```
public void setAccessToken(String)
```

Default Value

No default value

Data Type

String

See also

OAuth 2.0 authentication on page 47

AuthenticationMethod

Purpose

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

Valid Values

`Basic` | `URLParameter` | `OAuth2`

Behavior

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

If set to `UrlParameter`, the driver passes security tokens (API keys) via the URL for authentication. You must also specify the API key using the SecurityToken property.

If set to `OAuth2`, the driver uses OAuth 2.0 to authenticate to REST endpoints. See "OAuth 2.0 authentication" for details.

Data Source Methods

```
public String getAuthenticationMethod()
```

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

Default Value

`Basic`

Data Type

String

See also

OAuth 2.0 authentication on page 47
Authentication on page 46

ClientID

Purpose

Specifies the client ID key for your application when authenticating to Aha! with OAuth 2.0 enabled.

Valid Values

`String`

where:

`String`

is the client ID key for your application.

Notes

See "OAuth 2.0 authentication" for more information.

Data Source Methods

```
public String getClientId()
```

```
public void setClientId(String)
```

Default Value

No default value

Data Type

String

See also

OAuth 2.0 authentication on page 47

ClientSecret

Purpose

Specifies the client secret for your application when authenticating to Aha! with OAuth 2.0 enabled.

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

Valid Values

String

where:

String

is the client secret for your application.

Notes

See "OAuth 2.0 authentication" for more information.

Data Source Methods

```
public String getClientSecret()
```

```
public void setClientSecret(String)
```

Default Value

No default value

Data Type

String

See also

OAuth 2.0 authentication on page 47

ConnectionRetryCount

Purpose

The number of times the driver retries connection attempts to Aha! until a successful connection is established.

Valid Values

`0` | *x*

where:

x

is a positive integer that represents the number of retries.

Behavior

If set to `0`, the driver does not try to reconnect after the initial unsuccessful attempt.

If set to *x*, the driver retries connection attempts the specified number of times. If a connection is not established during the retry attempts, the driver returns an exception that is generated by the last server to which it tried to connect.

Example

If this property is set to 2, the driver retries the server twice after the initial retry attempt.

Notes

- If an application sets a login timeout value (for example, using DataSource.loginTimeout or DriverManager.loginTimeout), and the login timeout expires, the driver ceases connection attempts.
- The ConnectionRetryDelay property specifies the wait interval, in seconds, to occur between retry attempts.

Data Source Methods

```
public Integer getConnectionRetryCount()
```

```
public void setConnectionRetryCount(Integer)
```

Default Value

5

Data Type

Integer

See also

ConnectionRetryDelay on page 61

ConnectionRetryDelay

Purpose

The number of seconds the driver waits between connection retry attempts when ConnectionRetryCount is set to a positive integer.

Valid Values

`0` | *x*

where:

x

is a number of seconds.

Behavior

If set to `0`, the driver does not delay between retries.

If set to *x*, the driver waits between connection retry attempts the specified number of seconds.

Example

If ConnectionRetryCount is set to 2 and this property is set to 3, the driver retries the server twice after the initial retry attempt. The driver waits 3 seconds between retry attempts.

Data Source Methods

```
public Integer getConnectionRetryDelay()
```

```
public void setConnectionRetryDelay(Integer)
```

Default Value

`1`

Data Type

Integer

See also

ConnectionRetryCount on page 60

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 79
Refresh Map (EXT) on page 93

DebugRecord

Purpose

Specifies the directory where the driver generates debug record files. When a value is specified, the driver records server requests and responses to a set of files stored in this location. These files assist in troubleshooting by providing a method for Technical Support to reproduce and debug issues for REST services that are not publicly accessible.

Important: Debug record files may capture security-related headers, such as auth or token headers. Before sending Technical Support debug files, review the content to remove any confidential information that may have been recorded.

Valid Values

debug_record_folder

where:

debug_record_folder

is the location of the folder where the debug record files are to be generated. For example, `C:\Temp\MyDebug Folder`.

Notes

- The specified directory must exist.
- You must have write access to the specified directory.
- The contents of the specified directory are deleted every time a connection is established.
- For more information, refer to "Enabling Debug Record Mode" in the *Progress DataDirect for JDBC Drivers Reference*.
- For assistance, contact Technical Support.

Data Source Methods

```
public String getDebugRecord()
public void setDebugRecord(String)
```

Default Value

No default value

Data Type

String

See also

Contacting Technical Support on page 29

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 49

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 49

JDBCBehavior

Purpose

Determines how the driver describes database data types that map to the following JDBC 4.0 data types: NCHAR, NVARCHAR, NLONGVARCHAR, NCLOB, and SQLXML.

Valid Values

`0 | 1`

Behavior

If set to `0`, the driver describes the data types as JDBC 4.0 data types.

If set to `1`, the driver describes the data types using JDBC 3.0-equivalent data types regardless of the JVM. This allows your application to continue using JDBC 3.0 types.

Data Source Methods

```
public Integer getJdbcBehavior()
```

```
public void setJdbcBehavior(Integer)
```

Default Value

1

Data Type

Integer

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

LoginTimeout

Purpose

The amount of time, in seconds, that the driver waits for a connection to be established before timing out the connection request.

Valid Values

-1 | 0 | *x*

where:

x

is a positive integer that specifies a number of seconds.

Behavior

If set to `-1` | `0`, the driver does not time out a connection request.

If set to *x*, the driver waits for the specified number of seconds before returning control to the application and throwing a timeout exception.

Data Source Methods

```
public Integer getLoginTimeout()
```

```
public void setLoginTimeout(Integer)
```

Default Value

`0`

Data Type

Integer

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

See also

RegisterStatementPoolMonitorMBean on page 78

OAuthCode

Purpose

Specifies the temporary authorization code that is exchanged for access tokens when OAuth 2.0 authentication is enabled.

Valid Values

`String`

where:

`String`

is an OAuth 2.0 authorization code.

Notes

- This property is configured by the application.
- See "OAuth 2.0 authentication" for more information.

Data Source Methods

```
public String getCode()
```

```
public void setCode(String)
```

Default Value

No default value

Data Type

String

See also

OAuth 2.0 authentication on page 47

Password

Purpose

A password that is used to connect to the service.

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 on page 46

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 on page 17

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 on page 17

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 on page 17

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 on page 17

QueryTimeout

Purpose

Sets the default query timeout (in seconds) for all statements created by a connection.

Valid Values

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

where:

x

is a number of seconds.

Behavior

If set to `-1`, the query timeout functionality is disabled. The driver silently ignores calls to the Statement.setQueryTimeout() method.

If set to `0`, the default query timeout is infinite (the query does not time out).

If set to *x*, the driver uses the value as the default timeout for any statement that is created by the connection. To override the default timeout value that is set by this property, call the Statement.setQueryTimeout() method to set a timeout value for a particular statement.

Data Source Methods

```
public Integer getQueryTimeout()
```

```
public void setQueryTimeout(Integer)
```

Default Value

`0`

Data Type

Integer

ReadAhead

Purpose

Specifies the maximum number of fetch requests the driver issues in parallel. By default, the driver queues the next page when processing the current page. This property allows you to fetch multiple requests simultaneously, thereby improving throughput and performance.

Caution: Due to potential impacts to other users on the network, we strongly recommend specifying only smaller values for this property. For example, in fully optimized environments, which include exceptionally fast connections and low latency, we recommend a setting of no higher than `10`. For typical environments, this value should be considerably lower.

Valid Values

`0` | *x*

where:

x

is the maximum number of fetch requests the driver issues in parallel up to 100.

Behavior

If set to `0`, the driver queues the next page while processing the current page.

If set to *x*, the driver executes fetch requests as they are issued until the number of active parallel-requests equals the specified value. When that threshold is met, the driver waits until the results of a request are processed before requesting the next page of data.

Notes

- Specifying larger values for this property generally improves performance; however, with the following warnings:
  - Larger values can increase the load on the server, which may adversely affect performance of other users. If you encounter issues, decrease the value specified for this property.
  - Larger values may result in unnecessary requests if your application only requires the first few rows of results. This may be an issue if your service places limits on the number of web requests.

Data Source Methods

```
public Integer getReadAheadThreads()
```

```
public void setReadAheadThreads(Integer)
```

Default Value

`0`

Data Type

Integer

See also

Performance considerations on page 49

RedirectURI

Purpose

Specifies the endpoint to which the client is returned after third-party authorization for OAuth 2.0 implementations.

Valid Values

String

where:

String

is the endpoint to which the client is returned after third-party authorization. For example, `http://localhost`.

Notes

- The redirect URI is often registered with the authentication service to provide improved security. Registering the endpoint prevents your valid authentication credentials being redirected to a malicious site; therefore, reducing the risk of sharing your access token and other sensitive information with unauthorized parties.
- See "OAuth 2.0 authentication" for examples and more information.

Data Source Methods

```
public String getRedirUri()
```

```
public void setRedirUri(String)
```

Default Value

No default value

Data Type

String

See also

OAuth 2.0 authentication on page 47

RefreshSchema

Purpose

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

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver automatically refreshes the map when a user first connects to the service. 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 70

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 93

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=UrlParameter`); 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 API key using a data source, be aware that the value of 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.

Notes

- The value of this property correlates to the API Key value for the service. API keys can be generated through the Aha! user interface. Refer to the Aha! documentation for more information.

Data Source Methods

```
public String getSecurityToken()
```

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

Default Value

No default value

Data Type

String

See also

URL parameter (API key) authentication on page 47

ServerName

Purpose

Specifies the host name portion of the HTTP endpoint to which you send requests.

Valid Values

`string`

where:

string

is the host name portion of the HTTP endpoint to which you send requests.

Data Source Methods

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

Default Value

No default value

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).
load=*classname*	Loads the driver specified by *classname*.

Attribute	Description
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.
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.

Attribute	Description
`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`.
- 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 85

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 84

User

Purpose

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

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

WSFetchSize

Purpose

Specifies the number of rows of data the driver attempts to fetch for each JDBC call when paging is enabled for an endpoint.

Valid Values

`0` | *`x`*

where:

x

is a positive integer from 1 to 10000 that defines a number of rows.

Behavior

If set to `0`, the driver attempts to fetch up to the maximum number of 10000 rows. This value typically provides the maximum throughput.

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

Notes

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

Data Source Methods

```
public Integer getWsFetchSize()
```

```
public void setWsFetchSize(Integer)
```

Default Value

`2000`

Data Type

Integer

See also

FetchSize on page 65
Performance considerations on page 49

WSPoolSize

Purpose

Specifies the maximum number of sessions the driver uses. This allows the driver to have multiple web service requests active when multiple JDBC connections are open, thereby improving throughput and performance.

Valid Values

x

where:

x

is an integer that determines the number of sessions the driver uses to distribute calls. This value should not exceed the number of sessions permitted by your account.

Notes

- You can improve performance by increasing the number of sessions specified by this property. By increasing the number of sessions the driver uses, you can improve throughput by distributing calls across multiple sessions when multiple connections are active.
- The maximum number of sessions is determined by the setting of WSPoolSize for the connection that initiates the session. For subsequent connections to an active session, the setting is ignored and a warning is returned. To change the maximum number of sessions, close all connections using the driver; then, open a new connection with desired limit specified for this property.

Data Source Methods

```
public Integer getWsPoolSize()
```

```
public void setWsPoolSize(Integer)
```

Default Value

`1`

Data Type

Integer

See also

Performance considerations on page 49

WSRetryCount

Purpose

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

Valid Values

`0` | *x*

where:

x

is a positive integer

Behavior

If set to `0`, the driver does not retry timed-out requests after the initial unsuccessful attempt.

If set to *x*, the driver retries the timed-out request the specified number of times.

Data Source Methods

```
public Integer getWSRetryCount()
```

```
public void setWSRetryCount(Integer)
```

Default Value

5

Data Type

Integer

See also

WSTimeout on page 89

WSTimeout

Purpose

Specifies the time, in seconds, that the driver waits for a response to a web service request.

Valid Values

`0` | *x*

where:

x

is a positive integer that defines the number of seconds the driver waits for a response to a web service request.

Behavior

If set to `0`, the driver waits indefinitely for a response; there is no timeout.

If set to *x*, the driver uses the value as the default timeout, measured in seconds, for any statement created by the connection.

If a Select request times out and WSRetryCount is set to retry timed-out requests, the driver retries the request the specified number of times.

Data Source Methods

```
public Integer getWSTimeout()
```

```
public void setWSTimeout(Integer)
```

Default Value

`120`

Data Type

Integer

See also

WSRetryCount on page 88

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
- 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 13: 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=AHA`
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 aha.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 95
From clause on page 97
Where clause on page 98
Group By clause on page 99
Having clause on page 100
Union operator on page 100
Intersect operator on page 101
Except and Minus operators on page 102
Order By clause on page 103
Limit clause on page 103

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 106
Aggregate functions on page 96

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 14: 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 98

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 104

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 104
SQL expressions on page 106

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 104
SQL expressions on page 106

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 104
SQL expressions on page 106

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 104
SQL expressions on page 106

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
```

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 15: 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 16: 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 17: 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 18: 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 19: 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 20: 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 21: 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 Aha! data model

The Aha! 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 REST API calls. To expose Aha! resources to SQL applications, the driver maps Aha! endpoints to a set of relational parent and child tables. The following sections describe the relational tables exposed by the driver along with their corresponding Aha! REST API call.

For details, see the following topics:

- IDEACUSTOMFIELDS
- IDEAENDORSEMENTS
- IDEAS
- IDEATAGS
- MASTERFEATURECHILDREN
- MASTERFEATURECUSTOMFIELDS
- MASTERFEATUREGOALS
- MASTERFEATUREINTEGRATIONFIELDS
- MASTERFEATURELINKS
- MASTERFEATURES
- MASTERFEATURETAGS
- PRODUCTCHILDREN
- PRODUCTCUSTOMFIELDS
- PRODUCTS
- PRODUCTSCREENCUSTOMFIELDOPTIONS
- PRODUCTSCREENCUSTOMFIELDS
- PRODUCTSCREENS
- RELEASECUSTOMFIELDS
- RELEASEGOALS
- RELEASEINITIATIVES
- RELEASEINTEGRATIONFIELDS
- RELEASES
- REQUIREMENTATTACHMENTS
- REQUIREMENTCUSTOMFIELDS
- REQUIREMENTINTEGRATIONFIELDS
- REQUIREMENTS

CAPACITYINVESTMENTESTIMATES

Endpoint

<*hostname*>/api/v1/features/{feature_key}/capacity_investments

Parent Table

CAPACITYINVESTMENTS

Columns

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

Column Name	Data Type	Notes
CAPACITYINVESTMENTS_ID*	BigInt	References: CAPACITYINVESTMENTS.ID
POSITION*	Integer	
COMPUTED	Boolean	
ID	BigInt	
IGNORED	Boolean	
PERIOD_START	Date	
TEAM_ID	Integer	
TOTAL	Double	

CAPACITYINVESTMENTS

Endpoint

`<hostname>/api/v1/features/{feature_key}/capacity_investments`

Columns

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

Column Name	Data Type
ID*	BigInt
CAPACITY_SCENARIO_ID	Integer
DATE_SOURCE	VarChar(64)
END_DATE	Date
ESTIMATE_SOURCE	VarChar(64)
FEATURE_CREATED_AT	Timestamp(3)
FEATURE_ID	BigInt
FEATURE_ID_1	Integer

Column Name	Data Type
FEATURE_KEY	VarChar(64)
FEATURE_NAME	VarChar(64)
FEATURE_PRODUCT_ID	Integer
FEATURE_REFERENCE_NUM	VarChar(64)
FEATURE_RESOURCE	VarChar(66)
FEATURE_URL	VarChar(64)
INITIATIVE_ID	BigInt
INITIATIVE_KEY	VarChar(64)
MASTER_FEATURE_ID	BigInt
MASTER_FEATURE_KEY	VarChar(64)
PRODUCT_ID	BigInt
PRODUCT_KEY	VarChar(64)
START_DATE	Date
TOTAL	Double

COMMENTS

Endpoint

<*hostname*>/api/v1/goals/{goal_id}/comments

Columns

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

Column Name	Data Type
ID*	BigInt
BODY	VarChar(64)
COMMENTABLE_ID	Integer
COMMENTABLE_PRODUCT_ID	Integer

Column Name	Data Type
COMMENTABLE_RESOURCE	VarChar(75)
COMMENTABLE_TYPE	VarChar(64)
COMMENTABLE_URL	VarChar(82)
CREATED_AT	Timestamp(9)
FEATURE_ID	BigInt
GOAL_ID	BigInt
IDEA_ID	BigInt
INITIATIVE_ID	BigInt
PRODUCT_ID	BigInt
REQUIREMENT_ID	BigInt
RESOURCE	VarChar(70)
UPDATED_AT	Timestamp(9)
URL	VarChar(64)
USER_CREATED_AT	Timestamp(9)
USER_EMAIL	VarChar(64)
USER_ID	BigInt
USER_NAME	VarChar(64)
USER_UPDATED_AT	Timestamp(9)

FEATUREATTACHMENTS

Endpoint

<*hostname*>/api/v1/features

Parent Table

FEATURES

Columns

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

Column Name	Data Type	Notes
FEATURES_ID*	BigInt	References: FEATURES.ID
POSITION*	Integer	
CONTENT_TYPE	VarChar(64)	
CREATED_AT	Timestamp(9)	
DOWNLOAD_URL	VarChar(207)	
FILE_NAME	VarChar(64)	
FILE_SIZE	Integer	
ID	BigInt	
UPDATED_AT	Timestamp(9)	

FEATURECUSTOMFIELDS

Endpoint

<*hostname*>/api/v1/features

Parent Table

FEATURES

Columns

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

Column Name	Data Type	Notes
FEATURES_ID*	BigInt	References: FEATURES.ID
POSITION*	Integer	
KEY	VarChar(64)	
NAME	VarChar(64)	

Column Name	Data Type	Notes
TYPE	VarChar(64)	
VALUE	VarChar(64)	

FEATUREGOALS

Endpoint

<*hostname*>/api/v1/features

Parent Table

FEATURES

Columns

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

Column Name	Data Type	Notes
FEATURES_ID*	BigInt	References: FEATURES.ID
POSITION*	Integer	
CREATED_AT	Timestamp(3)	
DESCRIPTION_BODY	VarChar(1024)	
DESCRIPTION_CREATED_AT	Timestamp(3)	
DESCRIPTION_ID	Integer	
ID	Integer	
NAME	VarChar(64)	
RESOURCE	VarChar(69)	
URL	VarChar(82)	

FEATUREINTEGRATIONFIELDS

Endpoint

<*hostname*>/api/v1/features

Parent Table

FEATURES

Columns

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

Column Name	Data Type	Notes
FEATURES_ID*	BigInt	References: FEATURES.ID
POSITION*	Integer	
CREATED_AT	Timestamp(9)	
ID	BigInt	
INTEGRATION_ID	BigInt	
NAME	VarChar(64)	
SERVICE_NAME	VarChar(64)	
VALUE	VarChar(105)	

FEATURELINKS

Endpoint

`<hostname>/api/v1/features`

Parent Table

FEATURES

Columns

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

Column Name	Data Type	Notes
FEATURES_ID*	BigInt	References: FEATURES.ID
POSITION*	Integer	
CHILD_RECORD_CREATED_AT	Timestamp(3)	
CHILD_RECORD_ID	BigInt	

Column Name	Data Type	Notes
CHILD_RECORD_NAME	VarChar(64)	
CHILD_RECORD_PRODUCT_ID	BigInt	
CHILD_RECORD_REFERENCE_NUM	VarChar(64)	
CHILD_RECORD_RESOURCE	VarChar(84)	
CHILD_RECORD_URL	VarChar(73)	
CREATED_AT	Timestamp(3)	
LINK_TYPE	VarChar(64)	
LINK_TYPE_ID	Integer	
PARENT_RECORD_CREATED_AT	Timestamp(3)	
PARENT_RECORD_ID	BigInt	
PARENT_RECORD_NAME	VarChar(114)	
PARENT_RECORD_PRODUCT_ID	BigInt	
PARENT_RECORD_REFERENCE_NUM	VarChar(64)	
PARENT_RECORD_RESOURCE	VarChar(85)	
PARENT_RECORD_URL	VarChar(75)	

FEATUREREQUIREMENTS

Endpoint

`<hostname>/api/v1/features`

Parent Table

FEATURES

Columns

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

Column Name	Data Type	Notes
FEATURES_ID*	BigInt	References: FEATURES.ID

Column Name	Data Type	Notes
POSITION_1*	Integer	
CREATED_AT	Timestamp(3)	
DESCRIPTION_BODY	VarChar(1024)	
DESCRIPTION_CREATED_AT	Timestamp(3)	
DESCRIPTION_ID	Integer	
ID	BigInt	
KEY	VarChar(64)	
NAME	VarChar(64)	
POSITION	Integer	
RELEASE_ID	Integer	
RESOURCE	VarChar(75)	
UPDATED_AT	Timestamp(3)	
URL	VarChar(64)	
WORKFLOW_STATUS_COLOR	VarChar(64)	
WORKFLOW_STATUS_COMPLETE	Boolean	
WORKFLOW_STATUS_ID	Integer	
WORKFLOW_STATUS_NAME	VarChar(64)	
WORKFLOW_STATUS_POSITION	Integer	

FEATURES

Endpoint

`<hostname>/api/v1/features`

Columns

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

Column Name	Data Type
ID*	BigInt
ASSIGNED_TO_USER	VarChar(64)
COMMENTS_COUNT	Integer
CREATED_AT	Timestamp(3)
CREATED_BY_USER_AVATAR_URL	VarChar(174)
CREATED_BY_USER_CREATED_AT	Timestamp(3)
CREATED_BY_USER_EMAIL	VarChar(64)
CREATED_BY_USER_ID	BigInt
CREATED_BY_USER_NAME	VarChar(64)
CREATED_BY_USER_UPDATED_AT	Timestamp(3)
DESCRIPTION_BODY	VarChar(1024)
DESCRIPTION_CREATED_AT	Timestamp(3)
DESCRIPTION_ID	BigInt
DUE_DATE	VarChar(64)
FEATURE_ONLY_ORIGINAL_ESTIMATE	VarChar(64)
FEATURE_ONLY_REMAINING_ESTIMATE	VarChar(64)
FEATURE_ONLY_WORK_DONE	VarChar(64)
FEATURETIMETRACKING_ID	BigInt
FEATURETIMETRACKING_OCCURRED_ON	Date
FEATURETIMETRACKING_USER_CREATED_AT	Timestamp(3)
FEATURETIMETRACKING_USER_EMAIL	VarChar(64)
FEATURETIMETRACKING_USER_ID	BigInt
FEATURETIMETRACKING_USER_NAME	VarChar(64)
FEATURETIMETRACKING_USER_UPDATED_AT	Timestamp(3)
FEATURETIMETRACKING_WORK_DONE	Double
FEATURETIMETRACKING_WORK_UNITS	Integer

Column Name	Data Type
INITIATIVE_CREATED_AT	Timestamp(3)
INITIATIVE_DESCRIPTION_BODY	VarChar(1024)
INITIATIVE_DESCRIPTION_CREATED_AT	Timestamp(3)
INITIATIVE_DESCRIPTION_ID	BigInt
INITIATIVE_ID	BigInt
INITIATIVE_NAME	VarChar(64)
INITIATIVE_RESOURCE	VarChar(87)
INITIATIVE_URL	VarChar(76)
KEY	VarChar(64)
NAME	VarChar(127)
ORIGINAL_ESTIMATE	VarChar(64)
POSITION	Integer
PRODUCT_ID	BigInt
PROGRESS	VarChar(64)
PROGRESS_SOURCE	VarChar(64)
RELEASE_CREATED_AT	Timestamp(3)
RELEASE_ID	BigInt
RELEASE_KEY	VarChar(64)
RELEASE_NAME	VarChar(64)
RELEASE_OWNER_AVATAR_URL	VarChar(174)
RELEASE_OWNER_CREATED_AT	Timestamp(3)
RELEASE_OWNER_EMAIL	VarChar(64)
RELEASE_OWNER_ID	BigInt
RELEASE_OWNER_NAME	VarChar(64)
RELEASE_OWNER_UPDATED_AT	Timestamp(3)
RELEASE_PARKING_LOT	Boolean

Column Name	Data Type
RELEASE_PRODUCT_ID	BigInt
RELEASE_PROJECT_CREATED_AT	Timestamp(3)
RELEASE_PROJECT_ID	BigInt
RELEASE_PROJECT_NAME	VarChar(64)
RELEASE_PROJECT_PRODUCT_LINE	Boolean
RELEASE_PROJECT_REFERENCE_PREFIX	VarChar(64)
RELEASE_RESOURCE	VarChar(87)
RELEASE_URL	VarChar(76)
REMAINING_ESTIMATE	VarChar(64)
RESOURCE	VarChar(84)
SCORE	Integer
START_DATE	VarChar(64)
UPDATED_AT	Timestamp(3)
URL	VarChar(73)
USE_REQUIREMENTS_ESTIMATES	Boolean
WORK_DONE	VarChar(64)
WORK_UNITS	Integer
WORKFLOW_KIND_ID	BigInt
WORKFLOW_KIND_NAME	VarChar(64)
WORKFLOW_STATUS_COLOR	VarChar(64)
WORKFLOW_STATUS_COMPLETE	Boolean
WORKFLOW_STATUS_ID	BigInt
WORKFLOW_STATUS_NAME	VarChar(64)
WORKFLOW_STATUS_POSITION	Integer

FEATURESCOREFACTS

Endpoint

<*hostname*>/api/v1/features

Parent Table

FEATURES

Columns

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

Column Name	Data Type	Notes
FEATURES_ID*	BigInt	References: FEATURES.ID
POSITION*	Integer	
ID	BigInt	
NAME	VarChar(64)	
VALUE	Integer	

FEATURETAGS

Endpoint

<*hostname*>/api/v1/features

Parent Table

FEATURES

Columns

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

Column Name	Data Type	Notes
FEATURES_ID*	BigInt	References: FEATURES.ID
POSITION*	Integer	
FEATURETAG	VarChar(64)	

IDEACATEGORIES

Endpoint

<*hostname*>/api/v1/products/{product_key}/idea_categories

Columns

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

Column Name	Data Type
ID*	BigInt
CREATED_AT	Timestamp(9)
NAME	VarChar(64)
PARENT_ID	Integer
PRODUCT_ID	BigInt
PRODUCT_KEY	VarChar(64)

IDEACUSTOMFIELDS

Endpoint

<*hostname*>/api/v1/ideas

Parent Table

IDEAS

Columns

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

Column Name	Data Type	Notes
IDEAS_ID*	BigInt	References: IDEAS.ID
POSITION*	Integer	
KEY	VarChar(64)	
NAME	VarChar(64)	

Column Name	Data Type	Notes
TYPE	VarChar(64)	
VALUE	VarChar(64)	

IDEAENDORSEMENTS

Endpoint

<*hostname*>/api/v1/ideas/{idea_key}/endorsements

Columns

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

Column Name	Data Type
ID*	BigInt
CREATED_AT	Timestamp(3)
ENDORSED_BY_PORTAL_USER_CREATED_AT	Timestamp(3)
ENDORSED_BY_PORTAL_USER_EMAIL	VarChar(64)
ENDORSED_BY_PORTAL_USER_ID	Integer
ENDORSED_BY_PORTAL_USER_NAME	VarChar(64)
IDEA_ID	BigInt
IDEA_KEY	VarChar(64)
VALUE	VarChar(64)

IDEAS

Endpoint

<*hostname*>/api/v1/ideas

Columns

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

Column Name	Data Type
ID*	BigInt
ASSIGNED_TO_USER	VarChar(64)
CATEGORIES_CREATED_AT	Timestamp(3)
CATEGORIES_ID	BigInt
CATEGORIES_NAME	VarChar(64)
CATEGORIES_PARENT_ID	Integer
COMMENTS_COUNT	Integer
CREATED_AT	Timestamp(9)
CREATED_BY_PORTAL_USER_CREATED_AT	Timestamp(3)
CREATED_BY_PORTAL_USER_EMAIL	VarChar(64)
CREATED_BY_PORTAL_USER_ID	BigInt
CREATED_BY_PORTAL_USER_NAME	VarChar(64)
DESCRIPTION_BODY	LongVarChar(1024)
DESCRIPTION_CREATED_AT	Timestamp(9)
DESCRIPTION_ID	VarChar(32767)
ENDORSEMENTS_COUNT	Integer
KEY	VarChar(64)
NAME	VarChar(184)
PRODUCT_CREATED_AT	Timestamp(3)
PRODUCT_ID	BigInt
PRODUCT_NAME	VarChar(64)
PRODUCT_PRODUCT_LINE	Boolean
PRODUCT_REFERENCE_PREFIX	VarChar(64)
RESOURCE	VarChar(87)
SCORE_FACTS_ID	BigInt
SCORE_FACTS_NAME	VarChar(64)

Column Name	Data Type
SCORE_FACTS_VALUE	Integer
URL	VarChar(85)
WORKFLOW_STATUS_COLOR	VarChar(64)
WORKFLOW_STATUS_COMPLETE	Boolean
WORKFLOW_STATUS_ID	VarChar(32767)
WORKFLOW_STATUS_NAME	VarChar(64)
WORKFLOW_STATUS_POSITION	Integer

IDEATAGS

Endpoint

<*hostname*>/api/v1/ideas

Parent Table

IDEAS

Columns

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

Column Name	Data Type	Notes
IDEAS_ID*	BigInt	References: IDEAS.ID
POSITION*	Integer	
IDEATAG	VarChar(64)	

MASTERFEATURECHILDREN

Endpoint

<*hostname*>/api/v1/master_features

Parent Table

MASTERFEATURES

Columns

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

Column Name	Data Type	Notes
MASTERFEATURES_ID*	BigInt	References: MASTERFEATURES.ID
POSITION*	Integer	
CREATED_AT	Timestamp(3)	
ID	Integer	
NAME	VarChar(64)	
PRODUCT_ID	Integer	
REFERENCE_NUM	VarChar(64)	
RESOURCE	VarChar(66)	
URL	VarChar(64)	

MASTERFEATURECUSTOMFIELDS

Endpoint

`<hostname>/api/v1/master_features`

Parent Table

MASTERFEATURES

Columns

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

Column Name	Data Type	Notes
MASTERFEATURES_ID*	BigInt	References: MASTERFEATURES.ID
POSITION*	Integer	
KEY	VarChar(64)	
NAME	VarChar(64)	

Column Name	Data Type	Notes
TYPE	VarChar(64)	
VALUE	VarChar(64)	

MASTERFEATUREGOALS

Endpoint

<*hostname*>/api/v1/master_features

Parent Table

MASTERFEATURES

Columns

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

Column Name	Data Type	Notes
MASTERFEATURES_ID*	BigInt	References: MASTERFEATURES.ID
POSITION*	Integer	
CREATED_AT	Timestamp(3)	
DESCRIPTION_BODY	VarChar(1024)	
DESCRIPTION_CREATED_AT	Timestamp(3)	
DESCRIPTION_ID	BigInt	
ID	Integer	
NAME	VarChar(64)	
RESOURCE	VarChar(69)	
URL	VarChar(82)	

MASTERFEATUREINTEGRATIONFIELDS

Endpoint

<*hostname*>/api/v1/master_features

Parent Table

MASTERFEATURES

Columns

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

Column Name	Data Type	Notes
MASTERFEATURES_ID*	BigInt	References: MASTERFEATURES.ID
POSITION*	Integer	
CREATED_AT	Timestamp(6)	
ID	BigInt	
INTEGRATION_ID	Integer	
NAME	VarChar(64)	
SERVICE_NAME	VarChar(64)	
VALUE	VarChar(64)	

MASTERFEATURELINKS

Endpoint

`<hostname>/api/v1/master_features`

Parent Table

MASTERFEATURES

Columns

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

Column Name	Data Type	Notes
MASTERFEATURES_ID*	BigInt	References: MASTERFEATURES.ID
POSITION*	Integer	
CHILD_RECORD_CREATED_AT	Timestamp(3)	
CHILD_RECORD_ID	Integer	

Column Name	Data Type	Notes
CHILD_RECORD_NAME	VarChar(64)	
CHILD_RECORD_PRODUCT_ID	Integer	
CHILD_RECORD_REFERENCE_NUM	VarChar(64)	
CHILD_RECORD_RESOURCE	VarChar(64)	
CHILD_RECORD_URL	VarChar(64)	
CREATED_AT	Timestamp(3)	
LINK_TYPE	VarChar(64)	
LINK_TYPE_ID	Integer	
PARENT_RECORD_CREATED_AT	Timestamp(3)	
PARENT_RECORD_ID	Integer	
PARENT_RECORD_NAME	VarChar(64)	
PARENT_RECORD_PRODUCT_ID	Integer	
PARENT_RECORD_REFERENCE_NUM	VarChar(64)	
PARENT_RECORD_RESOURCE	VarChar(64)	
PARENT_RECORD_URL	VarChar(64)	

MASTERFEATURES

Endpoint

`<hostname>/api/v1/master_features`

Columns

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

Column Name	Data Type
ID*	BigInt
ASSIGNED_TO_USER_CREATED_AT	Timestamp(3)
ASSIGNED_TO_USER_DEFAULT_ASSIGNEE	Boolean

Column Name	Data Type
ASSIGNED_TO_USER_EMAIL	VarChar(64)
ASSIGNED_TO_USER_ID	Integer
ASSIGNED_TO_USER_NAME	VarChar(64)
ASSIGNED_TO_USER_UPDATED_AT	Timestamp(3)
ATTACHMENTS_CONTENT_TYPE	VarChar(64)
ATTACHMENTS_CREATED_AT	Timestamp(9)
ATTACHMENTS_DOWNLOAD_URL	VarChar(207)
ATTACHMENTS_FILE_NAME	VarChar(64)
ATTACHMENTS_FILE_SIZE	Integer
ATTACHMENTS_ID	BigInt
ATTACHMENTS_UPDATED_AT	Timestamp(9)
COMMENTS_COUNT	Integer
CREATED_AT	Timestamp(3)
CREATED_BY_USER_CREATED_AT	Timestamp(3)
CREATED_BY_USER_EMAIL	VarChar(64)
CREATED_BY_USER_ID	Integer
CREATED_BY_USER_NAME	VarChar(64)
CREATED_BY_USER_UPDATED_AT	Timestamp(3)
DESCRIPTION_BODY	VarChar(1024)
DESCRIPTION_CREATED_AT	Timestamp(3)
DESCRIPTION_ID	Integer
DUE_DATE	Date
INITIATIVE_CREATED_AT	Timestamp(3)
INITIATIVE_DESCRIPTION_BODY	VarChar(1024)
INITIATIVE_DESCRIPTION_CREATED_AT	Timestamp(3)
INITIATIVE_DESCRIPTION_ID	BigInt

Column Name	Data Type
INITIATIVE_ID	Integer
INITIATIVE_NAME	VarChar(64)
INITIATIVE_RESOURCE	VarChar(73)
INITIATIVE_URL	VarChar(64)
KEY	VarChar(64)
MASTER_FEATURE_ONLY_ORIGINAL_ESTIMATE	VarChar(64)
MASTER_FEATURE_ONLY_REMAINING_ESTIMATE	VarChar(64)
MASTER_FEATURE_ONLY_WORK_DONE	VarChar(64)
NAME	VarChar(64)
POSITION	Integer
PRODUCT_ID	Integer
PROGRESS	VarChar(64)
PROGRESS_SOURCE	VarChar(64)
RELEASE_CREATED_AT	Timestamp(3)
RELEASE_ID	BigInt
RELEASE_NAME	VarChar(64)
RELEASE_OWNER_CREATED_AT	Timestamp(3)
RELEASE_OWNER_EMAIL	VarChar(64)
RELEASE_OWNER_ID	Integer
RELEASE_OWNER_NAME	VarChar(64)
RELEASE_OWNER_UPDATED_AT	Timestamp(3)
RELEASE_PARKING_LOT	Boolean
RELEASE_PRODUCT_ID	Integer
RELEASE_PROJECT_CREATED_AT	Timestamp(3)
RELEASE_PROJECT_ID	Integer
RELEASE_PROJECT_NAME	VarChar(64)

Column Name	Data Type
RELEASE_PROJECT_PRODUCT_LINE	Boolean
RELEASE_PROJECT_REFERENCE_PREFIX	VarChar(64)
RELEASE_REFERENCE_NUM	VarChar(64)
RELEASE_RELEASE_DATE	Date
RELEASE_RESOURCE	VarChar(69)
RELEASE_START_DATE	Date
RELEASE_URL	VarChar(64)
RESOURCE	VarChar(79)
SCORE	VarChar(64)
SCORE_FACTS_ID	BigInt
SCORE_FACTS_NAME	VarChar(64)
SCORE_FACTS_VALUE	Integer
START_DATE	Date
UPDATED_AT	Timestamp(3)
URL	VarChar(64)
WORKFLOW_STATUS_COLOR	VarChar(64)
WORKFLOW_STATUS_COMPLETE	Boolean
WORKFLOW_STATUS_ID	Integer
WORKFLOW_STATUS_NAME	VarChar(64)
WORKFLOW_STATUS_POSITION	Integer

MASTERFEATURETAGS

Endpoint

`<hostname>/api/v1/master_features`

Parent Table

MASTERFEATURES

Columns

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

Column Name	Data Type	Notes
MASTERFEATURES_ID*	BigInt	References: MASTERFEATURES.ID
POSITION*	Integer	
MASTERFEATURETAG	VarChar(64)	

PRODUCTCHILDREN

Endpoint

<*hostname*>/api/v1/products

Parent Table

PRODUCTS

Columns

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

Column Name	Data Type	Notes
PRODUCTS_ID*	BigInt	References: PRODUCTS.ID
POSITION*	Integer	
CREATED_AT	Timestamp(9)	
ID	BigInt	
NAME	VarChar(54)	
PRODUCT_LINE	Boolean	
REFERENCE_PREFIX	VarChar(15)	

PRODUCTCUSTOMFIELDS

Endpoint

<*hostname*>/api/v1/products

Parent Table

PRODUCTS

Columns

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

Column Name	Data Type	Notes
PRODUCTS_ID*	BigInt	References: PRODUCTS.ID
POSITION*	Integer	
KEY	VarChar(64)	
NAME	VarChar(64)	
TYPE	VarChar(64)	
VALUE	VarChar(64)	

PRODUCTS

Endpoint

<*hostname*>/api/v1/products

Columns

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

Column Name	Data Type
ID*	BigInt
CAPACITY_PLANNING_ENABLED	Boolean
CREATED_AT	Timestamp(9)
DEFAULT_CAPACITY_UNITS	Integer

Column Name	Data Type
DESCRIPTION_BODY	VarChar(1024)
DESCRIPTION_CREATED_AT	Timestamp(9)
DESCRIPTION_ID	BigInt
EPIC_WORKFLOW_ID	BigInt
EPIC_WORKFLOW_NAME	VarChar(64)
EPIC_WORKFLOW_STATUSABLE_TYPE	VarChar(64)
EPIC_WORKFLOW_WORKFLOW_TYPE	VarChar(64)
FEATURE_WORKFLOW_ID	BigInt
FEATURE_WORKFLOW_NAME	VarChar(64)
FEATURE_WORKFLOW_STATUSABLE_TYPE	VarChar(64)
FEATURE_WORKFLOW_WORKFLOW_TYPE	VarChar(64)
HAS_IDEAS	Boolean
HAS_MASTER_FEATURES	Boolean
IDEA_WORKFLOW_ID	BigInt
IDEA_WORKFLOW_NAME	VarChar(64)
IDEA_WORKFLOW_STATUSABLE_TYPE	VarChar(64)
IDEA_WORKFLOW_WORKFLOW_TYPE	VarChar(64)
INITIATIVE_WORKFLOW_ID	BigInt
INITIATIVE_WORKFLOW_NAME	VarChar(64)
INITIATIVE_WORKFLOW_STATUSABLE_TYPE	VarChar(64)
INITIATIVE_WORKFLOW_WORKFLOW_TYPE	VarChar(64)
KEY	VarChar(64)
NAME	VarChar(64)
PARENT_CREATED_AT	Timestamp(9)
PARENT_ID	BigInt
PARENT_NAME	VarChar(64)

Column Name	Data Type
PARENT_PRODUCT_LINE	Boolean
PARENT_REFERENCE_PREFIX	VarChar(64)
PRODUCT_LINE	Boolean
PRODUCT_LINE_TYPE	VarChar(64)
RELEASE_WORKFLOW_ID	BigInt
RELEASE_WORKFLOW_NAME	VarChar(64)
RELEASE_WORKFLOW_STATUSABLE_TYPE	VarChar(64)
RELEASE_WORKFLOW_WORKFLOW_TYPE	VarChar(64)
REQUIREMENT_WORKFLOW_ID	BigInt
REQUIREMENT_WORKFLOW_NAME	VarChar(64)
REQUIREMENT_WORKFLOW_STATUSABLE_TYPE	VarChar(64)
REQUIREMENT_WORKFLOW_WORKFLOW_TYPE	VarChar(64)
RESOURCE	VarChar(76)
STRATEGIC_IMPERATIVE_WORKFLOW_ID	BigInt
STRATEGIC_IMPERATIVE_WORKFLOW_NAME	VarChar(64)
STRATEGIC_IMPERATIVE_WORKFLOW_STATUSABLE_TYPE	VarChar(64)
STRATEGIC_IMPERATIVE_WORKFLOW_WORKFLOW_TYPE	VarChar(64)
UPDATED_AT	Timestamp(9)
URL	VarChar(76)

PRODUCTSCREENCUSTOMFIELDOPTIONS

Endpoint

<*hostname*>/api/v1/products

Parent Table

PRODUCTSCREENCUSTOMFIELDS

Columns

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

Column Name	Data Type	Notes
PRODUCTS_ID*	BigInt	References: PRODUCTSCREENCUSTOMFIELDS .PRODUCTS_ID
PRODUCTSCREENS_POSITION*	Integer	References: PRODUCTSCREENCUSTOMFIELDS .PRODUCTSCREENS_POSITION
PRODUCTSCREENCUSTOMFIELDS _POSITION*	Integer	References: PRODUCTSCREENCUSTOMFIELDS .POSITION_2
POSITION*	Integer	
ID	BigInt	
LABEL	VarChar(36)	

PRODUCTSCREENCUSTOMFIELDS

Endpoint

`<hostname>/api/v1/products`

Parent Table

PRODUCTSCREENS

Columns

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

Column Name	Data Type	Notes
PRODUCTS_ID*	BigInt	References: PRODUCTS.ID
PRODUCTSCREENS_POSITION*	Integer	References: PRODUCTSCREENS.POSITION
POSITION_2*	Integer	
API_TYPE	VarChar(9)	
CONFIGURATION_DISABLE_DELIMITERS	Boolean	
CONFIGURATION_DISPLAY	VarChar(64)	
CONFIGURATION_PREFIX	VarChar(64)	

Column Name	Data Type	Notes
CONFIGURATION_SUFFIX	VarChar(64)	
ID	BigInt	
KEY	VarChar(33)	
NAME	VarChar(33)	
POSITION	Integer	
REQUIRED	Boolean	
TYPE	VarChar(69)	

PRODUCTSCREENS

Endpoint

`<hostname>/api/v1/products`

Parent Table

PRODUCTS

Columns

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

Column Name	Data Type	Notes
PRODUCTS_ID*	BigInt	References: PRODUCTS.ID
POSITION*	Integer	
ID	BigInt	
NAME	VarChar(69)	
SCREENABLE_TYPE	VarChar(33)	

RELEASECUSTOMFIELDS

Endpoint

`<hostname>/api/v1/releases`

Parent Table

RELEASES

Columns

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

Column Name	Data Type	Notes
RELEASES_ID*	BigInt	References: RELEASES.ID
POSITION*	Integer	
KEY	VarChar(64)	
NAME	VarChar(64)	
TYPE	VarChar(64)	
VALUE	VarChar(64)	

RELEASEGOALS

Endpoint

<*hostname*>/api/v1/releases

Parent Table

RELEASES

Columns

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

Column Name	Data Type	Notes
RELEASES_ID*	BigInt	References: RELEASES.ID
POSITION*	Integer	
CREATED_AT	Timestamp(3)	
DESCRIPTION_BODY	VarChar(1024)	
DESCRIPTION_CREATED_AT	Timestamp(3)	
DESCRIPTION_ID	Integer	

Column Name	Data Type	Notes
ID	Integer	
NAME	VarChar(64)	
RESOURCE	VarChar(69)	
URL	VarChar(82)	

RELEASEINITIATIVES

Endpoint

`<hostname>/api/v1/releases`

Parent Table

RELEASES

Columns

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

Column Name	Data Type	Notes
RELEASES_ID*	BigInt	References: RELEASES.ID
POSITION*	Integer	
CREATED_AT	Timestamp(3)	
DESCRIPTION_BODY	VarChar(1024)	
DESCRIPTION_CREATED_AT	Timestamp(3)	
DESCRIPTION_ID	BigInt	
ID	BigInt	
NAME	VarChar(64)	
RESOURCE	VarChar(85)	
URL	VarChar(75)	

RELEASEINTEGRATIONFIELDS

Endpoint

<*hostname*>/api/v1/releases

Parent Table

RELEASES

Columns

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

Column Name	Data Type	Notes
RELEASES_ID*	BigInt	References: RELEASES.ID
POSITION*	Integer	
CREATED_AT	Timestamp(6)	
ID	Integer	
INTEGRATION_ID	Integer	
NAME	VarChar(64)	
SERVICE_NAME	VarChar(64)	
VALUE	VarChar(64)	

RELEASES

Endpoint

<*hostname*>/api/v1/releases

Columns

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

Column Name	Data Type
ID*	BigInt
COMMENTS_COUNT	Integer

Column Name	Data Type
CREATED_AT	Timestamp(9)
CREATED_BY_USER_AVATAR_URL	VarChar(174)
CREATED_BY_USER_CREATED_AT	Timestamp(9)
CREATED_BY_USER_EMAIL	VarChar(64)
CREATED_BY_USER_ID	BigInt
CREATED_BY_USER_NAME	VarChar(64)
CREATED_BY_USER_UPDATED_AT	Timestamp(9)
DEVELOPMENT_STARTED_ON	Date
EXTERNAL_DATE_RESOLUTION	VarChar(64)
EXTERNAL_RELEASE_DATE	Date
EXTERNAL_RELEASE_DATE_DESCRIPTION	VarChar(64)
KEY	VarChar(64)
MASTER_RELEASE	Boolean
NAME	VarChar(64)
ORIGINAL_ESTIMATE	VarChar(64)
OWNER_AVATAR_URL	VarChar(174)
OWNER_CREATED_AT	Timestamp(9)
OWNER_EMAIL	VarChar(64)
OWNER_ID	BigInt
OWNER_NAME	VarChar(64)
OWNER_UPDATED_AT	Timestamp(9)
PARENT_CREATED_AT	Timestamp(9)
PARENT_ID	BigInt
PARENT_NAME	VarChar(64)
PARENT_REFERENCE_NUM	VarChar(64)
PARENT_RELEASE_DATE	Date

Column Name	Data Type
PARENT_RESOURCE	VarChar(82)
PARENT_START_DATE	Date
PARENT_URL	VarChar(82)
PARKING_LOT	Boolean
POSITION	Integer
PRODUCT_ID	BigInt
PROGRESS	Integer
PROGRESS_SOURCE	VarChar(64)
PROJECT_CREATED_AT	Timestamp(9)
PROJECT_ID	BigInt
PROJECT_NAME	VarChar(64)
PROJECT_PRODUCT_LINE	Boolean
PROJECT_REFERENCE_PREFIX	VarChar(64)
RELEASE_DATE	Date
RELEASED	Boolean
REMAINING_ESTIMATE	Double
RESOURCE	VarChar(90)
START_DATE	Date
THEME_BODY	VarChar(232)
THEME_CREATED_AT	Timestamp(9)
THEME_ID	BigInt
UPDATED_AT	Timestamp(9)
URL	VarChar(82)
WORK_DONE	Double
WORK_UNITS	Integer
WORKFLOW_STATUS_COLOR	VarChar(64)

Column Name	Data Type
WORKFLOW_STATUS_COMPLETE	Boolean
WORKFLOW_STATUS_ID	BigInt
WORKFLOW_STATUS_NAME	VarChar(64)
WORKFLOW_STATUS_POSITION	Integer

REQUIREMENTATTACHMENTS

Endpoint

`<hostname>/api/v1/features/{feature_key}/requirements`

Parent Table

REQUIREMENTS

Columns

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

Column Name	Data Type	Notes
REQUIREMENTS_ID*	BigInt	References: REQUIREMENTS.ID
POSITION*	Integer	
CONTENT_TYPE	VarChar(64)	
CREATED_AT	Timestamp(9)	
DOWNLOAD_URL	VarChar(207)	
FILE_NAME	VarChar(64)	
FILE_SIZE	Integer	
ID	BigInt	
UPDATED_AT	Timestamp(9)	

REQUIREMENTCUSTOMFIELDS

Endpoint

<*hostname*>/api/v1/features/{feature_key}/requirements

Parent Table

REQUIREMENTS

Columns

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

Column Name	Data Type	Notes
REQUIREMENTS_ID*	BigInt	References: REQUIREMENTS.ID
POSITION*	Integer	
KEY	VarChar(64)	
NAME	VarChar(64)	
TYPE	VarChar(64)	
VALUE	Date	

REQUIREMENTINTEGRATIONFIELDS

Endpoint

<*hostname*>/api/v1/features/{feature_key}/requirements

Parent Table

REQUIREMENTS

Columns

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

Column Name	Data Type	Notes
REQUIREMENTS_ID*	BigInt	References: REQUIREMENTS.ID
POSITION*	Integer	

Column Name	Data Type	Notes
CREATED_AT	Timestamp(6)	
ID	Integer	
INTEGRATION_ID	Integer	
NAME	VarChar(64)	
SERVICE_NAME	VarChar(64)	
VALUE	VarChar(64)	

REQUIREMENTS

Endpoint

`<hostname>/api/v1/features/{feature_key}/requirements`

Columns

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

Column Name	Data Type
ID*	BigInt
ASSIGNED_TO_USER_CREATED_AT	Timestamp(3)
ASSIGNED_TO_USER_DEFAULT_ASSIGNEE	Boolean
ASSIGNED_TO_USER_EMAIL	VarChar(64)
ASSIGNED_TO_USER_ID	Integer
ASSIGNED_TO_USER_NAME	VarChar(64)
ASSIGNED_TO_USER_UPDATED_AT	Timestamp(3)
COMMENTS_COUNT	Integer
CREATED_AT	Timestamp(3)
CREATED_BY_USER_CREATED_AT	Timestamp(3)
CREATED_BY_USER_EMAIL	VarChar(64)
CREATED_BY_USER_ID	Integer

Column Name	Data Type
CREATED_BY_USER_NAME	VarChar(64)
CREATED_BY_USER_UPDATED_AT	Timestamp(3)
DESCRIPTION_BODY	VarChar(1024)
DESCRIPTION_CREATED_AT	Timestamp(3)
DESCRIPTION_ID	Integer
FEATURE_CREATED_AT	Timestamp(3)
FEATURE_ID	BigInt
FEATURE_KEY	VarChar(64)
FEATURE_NAME	VarChar(64)
FEATURE_PRODUCT_ID	Integer
FEATURE_RESOURCE	VarChar(66)
FEATURE_URL	VarChar(64)
KEY	VarChar(64)
NAME	VarChar(64)
POSITION	Integer
RELEASE_ID	Integer
RESOURCE	VarChar(75)
UPDATED_AT	Timestamp(3)
URL	VarChar(64)
WORKFLOW_STATUS_COLOR	VarChar(64)
WORKFLOW_STATUS_COMPLETE	Boolean
WORKFLOW_STATUS_ID	Integer
WORKFLOW_STATUS_NAME	VarChar(64)
WORKFLOW_STATUS_POSITION	Integer