



EDB ODBC Connector

Version 12.2.0.2

| | | |
|-----|--------------------------------|----|
| 1 | What's New | 3 |
| 2 | Requirements Overview | 3 |
| 3 | EDB-ODBC Overview | 4 |
| 3.1 | Installing EDB-ODBC | 4 |
| 4 | Creating a Data Source | 19 |
| 5 | EDB-ODBC Connection Properties | 20 |
| 6 | EDB-ODBC Driver Functionality | 40 |
| 7 | Scram Compatibility | 70 |

1 What's New

The following features are added to create the EDB ODBC Connector **12.02.0000.02**:

- Support for EDB Postgres Advanced Server 13.
- Support for Ubuntu 20.04 LTS platform.

2 Requirements Overview

Supported Versions

The EDB ODBC Connector is certified with Advanced Server version 9.6 and above.

Supported Platforms

The EDB ODBC Connector native packages are supported on the following platforms:

64 bit Linux:

- Red Hat Enterprise Linux (x86_64) 8.x and 7.x
- CentOS (x86_64) 8.x and 7.x
- OEL Linux 8.x and 7.x
- PPC-LE 8 running RHEL or CentOS 7.x
- SLES 12.x
- Debian 10.x and 9.x
- Ubuntu 20.04 and 18.04 LTS

The EDB ODBC Connector graphical installers are supported on the following Windows platforms:

64-bit Windows:

- Windows Server 2019
- Windows Server 2016
- Windows Server 2012 R2
- Windows 10
- Windows 8.1

32-bit Windows:

- Windows 10
 - Windows 8.1
-

3 EDB-ODBC Overview

EDB ODBC is an interface that allows an ODBC compliant client application to connect to an Advanced Server database. The EDB-ODBC connector allows an application that was designed to work with other databases to run on Advanced Server; EDB ODBC provides a way for the client application to establish a connection, send queries and retrieve results from Advanced Server.

While EDB ODBC provides a level of application portability, it should be noted that the portability is limited; EDB ODBC provides a connection, but does not guarantee command compatibility. Commands that are acceptable in another database, may not work in Advanced Server.

The major components in a typical ODBC application are:

- The client application - written in a language that has a binding for ODBC
- The ODBC Administrator - handles named connections for Windows or Linux
- The database specific ODBC driver - EDB ODBC
- The ODBC compliant server - EDB Postgres Advanced Server

Client applications can be written in any language that has a binding for ODBC; C, MS-Access, and C++ are just a few.

3.1 Installing EDB-ODBC

The EDB ODBC Connector is distributed and installed with the EDB Postgres Advanced Server graphical or RPM installer.

Installing the Connector with an RPM Package

You can install the ODBC Connector using an RPM package on the following platforms:

- [RHEL 7](#)
- [RHEL 8](#)
- [CentOS 7](#)
- [CentOS 8](#)

On RHEL 7

Before installing the ODBC Connector, you must install the following prerequisite packages, and request credentials from EDB:

Install the `epel-release` package:

```
yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
```

Enable the optional, extras, and HA repositories:

```
subscription-manager repos --enable "rhel-*-optional-rpms" --enable "rhel-*-extras-rpms" --enable "rhel-ha-for-rhel-*-server-rpms"
```

You must also have credentials that allow access to the EDB repository. For information about requesting credentials, visit:

<https://info.enterprisedb.com/rs/069-ALB-339/images/Repository%20Access%2004-09-2019.pdf>

After receiving your repository credentials you can:

1. Create the repository configuration file.
2. Modify the file, providing your user name and password.
3. Install `edb-odbc`.

Creating a Repository Configuration File

To create the repository configuration file, assume superuser privileges, and invoke the following command:

```
yum -y install https://yum.enterprisedb.com/edbrepos/edb-repo-latest.noarch.rpm
```

The repository configuration file is named `edb.repo`. The file resides in `/etc/yum.repos.d`.

Modifying the file, providing your user name and password

After creating the `edb.repo` file, use your choice of editor to ensure that the value of the `enabled` parameter is `1`, and replace the `username` and `password` placeholders in the `baseurl` specification with the name and password of a registered EDB user.

```
[edb]
name=EnterpriseDB RPMs $releasever - $basearch
baseurl=https://<username>:
<password>@yum.enterprisedb.com/edb/redhat/rhel-$releasever-$basearch
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/ENTERPRISEDB-GPG-KEY
```

Installing ODBC Connector

After saving your changes to the configuration file, use the following commands to install the ODBC Connector:

```
yum install edb-odbc
yum install edb-odbc-devel
```

When you install an RPM package that is signed by a source that is not recognized by your system, yum may ask for your permission to import the key to your local server. If prompted, and you are satisfied that the packages come from a trustworthy source, enter **y**, and press **Return** to continue.

During the installation, yum may encounter a dependency that it cannot resolve. If it does, it will provide a list of the required dependencies that you must manually resolve.

On RHEL 8

Before installing the ODBC Connector, you must install the following prerequisite packages, and request credentials from EDB:

Install the **epel-release** package:

```
dnf -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
```

Enable the **codeready-builder-for-rhel-8-*rpms** repository:

```
ARCH=$( /bin/arch )
subscription-manager repos --enable "codeready-builder-for-rhel-8-
${ARCH}-rpms"
```

You must also have credentials that allow access to the EDB repository. For information about requesting credentials, visit:

<https://info.enterprisedb.com/rs/069-ALB-339/images/Repository%20Access%2004-09-2019.pdf>

After receiving your repository credentials you can:

1. Create the repository configuration file.
2. Modify the file, providing your user name and password.
3. Install `edb-odbc`.

Creating a Repository Configuration File

To create the repository configuration file, assume superuser privileges, and invoke the following command:

```
dnf -y https://yum.enterprisedb.com/edbrepos/edb-repo-
latest.noarch.rpm
```

The repository configuration file is named `edb.repo`. The file resides in `/etc/yum.repos.d`.

Modifying the file, providing your user name and password

After creating the `edb.repo` file, use your choice of editor to ensure that the value of the `enabled` parameter is `1`, and replace the `username` and `password` placeholders in the `baseurl` specification with the name and password of a registered EDB user.

```
[edb]
name=EnterpriseDB RPMs $releasever - $basearch
baseurl=https://<username>:
<password>@yum.enterprisedb.com/edb/redhat/rhel-$releasever-$basearch
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/ENTERPRISEDB-GPG-KEY
```

Installing ODBC Connector

After saving your changes to the configuration file, use the below command to install the ODBC Connector:

```
dnf install edb-odbc

dnf install edb-odbc-devel
```

When you install an RPM package that is signed by a source that is not recognized by your system, yum may ask for your permission to import the key to your local server. If prompted, and you are satisfied that the packages come from a trustworthy source, enter `y`, and press `Return` to continue.

During the installation, yum may encounter a dependency that it cannot resolve. If it does, it will provide a list of the required dependencies that you must manually resolve.

On CentOS 7

Before installing the ODBC Connector, you must install the following prerequisite packages, and request credentials from EDB:

Install the `epel-release` package:

```
yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
```

!!! Note You may need to enable the `[extras]` repository definition in the `CentOS-Base.repo` file (located in `/etc/yum.repos.d`).

You must also have credentials that allow access to the EDB repository. For information about requesting credentials, visit:

<https://info.enterprisedb.com/rs/069-ALB-339/images/Repository%20Access%2004-09-2019.pdf>

After receiving your repository credentials you can:

1. Create the repository configuration file.
2. Modify the file, providing your user name and password.
3. Install `edb-odbc`.

Creating a Repository Configuration File

To create the repository configuration file, assume superuser privileges, and invoke the following command:

```
yum -y install https://yum.enterprisedb.com/edbrepos/edb-repo-latest.noarch.rpm
```

The repository configuration file is named `edb.repo`. The file resides in `/etc/yum.repos.d`.

Modifying the file, providing your user name and password

After creating the `edb.repo` file, use your choice of editor to ensure that the value of the `enabled` parameter is `1`, and replace the `username` and `password` placeholders in the `baseurl` specification with the name and password of a registered EDB user.

```
[edb]
name=EnterpriseDB RPMs $releasever - $basearch
baseurl=https://<username>:
<password>@yum.enterprisedb.com/edb/redhat/rhel-$releasever-$basearch
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/ENTERPRISEDB-GPG-KEY
```


Installing ODBC Connector

After saving your changes to the configuration file, use the following command to install the ODBC Connector:

```
yum install edb-odbc  
  
yum install edb-odbc-devel
```

When you install an RPM package that is signed by a source that is not recognized by your system, yum may ask for your permission to import the key to your local server. If prompted, and you are satisfied that the packages come from a trustworthy source, enter **y**, and press **Return** to continue.

During the installation, yum may encounter a dependency that it cannot resolve. If it does, it will provide a list of the required dependencies that you must manually resolve.

On CentOS 8

Before installing the ODBC Connector, you must install the following prerequisite packages, and request credentials from EDB:

Install the **epel-release** package:

```
dnf -y install https://dl.fedoraproject.org/pub/epel/epel-release-  
latest-8.noarch.rpm
```

Enable the **PowerTools** repository:

```
dnf config-manager --set-enabled PowerTools
```

You must also have credentials that allow access to the EDB repository. For information about requesting credentials, visit:

<https://info.enterprisedb.com/rs/069-ALB-339/images/Repository%20Access%2004-09-2019.pdf>

After receiving your repository credentials you can:

1. Create the repository configuration file.
2. Modify the file, providing your user name and password.
3. Install **edb-odbc**.

Creating a Repository Configuration File

To create the repository configuration file, assume superuser privileges, and invoke the following command:

```
dnf -y install https://yum.enterprisedb.com/edbrepos/edb-repo-latest.noarch.rpm
```

The repository configuration file is named `edb.repo`. The file resides in `/etc/yum.repos.d`.

Modifying the file, providing your user name and password

After creating the `edb.repo` file, use your choice of editor to ensure that the value of the `enabled` parameter is `1`, and replace the `username` and `password` placeholders in the `baseurl` specification with the name and password of a registered EDB user.

```
[edb]
name=EnterpriseDB RPMs $releasever - $basearch
baseurl=https://<username>:
<password>@yum.enterprisedb.com/edb/redhat/rhel-$releasever-$basearch
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/ENTERPRISEDB-GPG-KEY
```

Installing ODBC Connector

After saving your changes to the configuration file, use the following command to install the ODBC Connector:

```
dnf install edb-odbc

dnf install edb-odbc-devel
```

When you install an RPM package that is signed by a source that is not recognized by your system, yum may ask for your permission to import the key to your local server. If prompted, and you are satisfied that the packages come from a trustworthy source, enter `y`, and press `Return` to continue.

During the installation, yum may encounter a dependency that it cannot resolve. If it does, it will provide a list of the required dependencies that you must manually resolve.

Updating an RPM Installation

If you have an existing EDB ODBC connector RPM installation, you can use yum or dnf to upgrade your repository configuration file and update to a more recent product version. To update the `edb.repo` file, assume superuser privileges and enter:

- On RHEL or CentOS 7:

```
yum upgrade edb-repo
```

- On RHEL or CentOS 8:

```
dnf upgrade edb-repo
```

yum or dnf will update the `edb.repo` file to enable access to the current EDB repository, configured to connect with the credentials specified in your `edb.repo` file. Then, you can use yum or dnf to upgrade any installed packages:

- On RHEL or CentOS 7:

```
yum upgrade edb-odbc
```

```
yum upgrade edb-odbc-devel
```

- On RHEL or CentOS 8:

```
dnf upgrade edb-odbc
```

```
dnf upgrade edb-odbc-devel
```

Installing the Connector on an SLES 12 Host

You can use the zypper package manager to install the connector on an SLES 12 host. zypper will attempt to satisfy package dependencies as it installs a package, but requires access to specific repositories that are not hosted at EDB. Before installing the connector, use the following commands to add EDB repository configuration files to your SLES host:

```
zypper addrepo https://zypp.enterprisedb.com/suse/edb-sles.repo
```

After creating the repository configuration files, use the `zypper refresh` command to refresh the metadata on your SLES host to include the EDB repositories.

When prompted for a `User Name` and `Password`, provide your connection credentials for the EDB repository. To request credentials for the repository, visit [the EDB website](#).

Before installing EDB Postgres Advanced Server or supporting components, you must also add SUSEConnect and the SUSE Package Hub extension to the SLES host, and register the host with SUSE, allowing access to SUSE repositories. Use the commands:

```
zypper install SUSEConnect
SUSEConnect -r 'REGISTRATION_CODE' -e 'EMAIL'
SUSEConnect -p PackageHub/12.4/x86_64
SUSEConnect -p sle-sdk/12.4/x86_64
```

For detailed information about registering a SUSE host, visit the [SUSE website](#).

Then, you can use the zypper utility to install the connector:

```
zypper install edb-odbc
```

```
zypper install edb-odbc-devel
```

Installing the Connector on a Debian or Ubuntu Host

To install a DEB package on a Debian or Ubuntu host, you must have credentials that allow access to the EDB repository. To request credentials for the repository, visit the [EDB website](#).

The following steps will walk you through on using the EDB apt repository to install a DEB package. When using the commands, replace the `username` and `password` with the credentials provided by EDB.

1. Assume superuser privileges:

```
sudo su -
```

2. Configure the EDB repository:

On Debian 9:

```
sh -c 'echo "deb
https://username:password@apt.enterprisedb.com/$(lsb_release -cs)-
edb/ $(lsb_release -cs) main" > /etc/apt/sources.list.d/edb-
$(lsb_release -cs).list'
```

On Debian 10:

1. Set up the EDB repository:

```
-c 'echo "deb [arch=amd64]
https://apt.enterprisedb.com/$(lsb_release -cs)-edb/
$(lsb_release -cs) main" > /etc/apt/sources.list.d/edb-
$(lsb_release -cs).list'
```

2. Substitute your EDB credentials for the `username` and `password` in the following command:

```
-c 'echo "machine apt.enterprisedb.com login <username> password
<password>" > /etc/apt/auth.conf.d/edb.conf'
```

3. Add support to your system for secure APT repositories:

```
apt-get install apt-transport-https
```

4. Add the EDB signing key:

```
wget -q -O - https://<username>:  
<password>@apt.enterprisedb.com/edb-deb.gpg.key | apt-key add -
```

5. Update the repository metadata:

```
apt-get update
```

6. Install DEB package:

```
apt-get install edb-odbc  
apt-get install edb-odbc-dev
```

Using the Graphical Installer to Install the Connector

You can use the EDB Connectors Installation wizard to add the ODBC connector to your system; the wizard is available at the [EDB website](#).

Download the installer, and then, right-click on the installer icon, and select **Run As Administrator** from the context menu.

When the **Language Selection** popup opens, select an installation language and click **OK** to continue to the **Setup** window (shown in Figure below).



The ODBC Connectors Installation wizard.

Click **Next** to continue.



The Installation dialog

Use the **Installation Directory** dialog to specify the directory in which the connector will be installed, and click **Next** to continue.



The Ready to Install dialog.

Click **Next** on the **Ready to Install** dialog to start the installation; popup dialogs confirm the progress of the installation wizard.



he installation is complete.

When the wizard informs you that it has completed the setup, click the **Finish** button to exit the dialog.

You can also use StackBuilder Plus to add or update the connector on an existing Advanced Server installation; to open StackBuilder Plus, select **StackBuilder Plus** from the **Windows Apps** menu.



Starting StackBuilder Plus

When StackBuilder Plus opens, follow the onscreen instructions. Select the **EnterpriseDB ODBC Connector** option from the **Database Drivers** node of the tree control.



Selecting the Connectors installer.

Follow the directions of the onscreen wizard to add or update an installation of the EDB Connectors.

4 Creating a Data Source

When a client application tries to establish a connection with a server, it typically provides a data source name (also known as a "DSN"). The driver manager looks through the ODBC configuration database for a data source whose name matches the DSN provided by the application.

On a Linux or Unix host, data sources are defined in a file; that file is usually named `/etc/odbc.ini`, but the name (and location) may vary. Use the following command to find out where unixODBC is searching for data source definitions:

```
$ odbc_config --odbcini --odbcinstini
```

On a Windows host, data sources are typically defined in the Windows registry.

You can also store a data source definition (called a "File DSN") in a plain-text file of your choice. A typical data source definition for the EDB-ODBC driver looks like this:

```
$ cat /etc/odbc.ini
[EnterpriseDB]
Description = EnterpriseDB DSN
Driver = EnterpriseDB
Trace = yes
TraceFile = /tmp/odbc.log
Database = edb
Servername = localhost
Username = enterprisedb
Password = manager
Port = 5444
```

The first line in the data source is the data source name. The name is a unique identifier, enclosed in square brackets. The data source name is followed by a series of 'keyword=value' pairs that identify individual connection properties that make up the data source.

The ODBC administrator utility creates named data sources for ODBC connections. In most cases, an ODBC administrator utility is distributed with the operating system (if you're using Windows or unixODBC, the tool is called the **ODBC Data Source Administrator**). If your operating system doesn't include an ODBC administrator, third-party options are available online.

Sections **Adding a Data Source Definition in Windows** and **Adding a Data Source Definition in Linux** walk you through adding a data source in Windows and Linux using the graphical tools available for each operating system. During the process of defining a data source, you'll be asked to specify a set of connection properties. Section **EDB-ODBC Connection Properties** contains information about **optional** data source connection properties; you can specify connection properties with graphical tools or edit the **odbc.ini** file with a text editor.

5 EDB-ODBC Connection Properties

The following table describes the connection properties that you can specify through the dialogs in the graphical connection manager tools, or in the **odbc.ini** file that defines a named data source. The columns identify the connection property (as it appears in the ODBC Administrator dialogs), the corresponding keyword (as it appears in the **odbc.ini** file), the default value of the property, and a description of the connection property.

| Property | Keyword name | Default value | Description |
|---|---|-------------------|---|
| Database | Database | None | The name of the database to which you are connecting. |
| Driver | Driver | EDB-ODBC | The name of the ODBC driver. |
| Server | Servername | Localhost | The name or IP address of the server that you are connecting to. |
| dbms_name Description User Name Password | dbms_name Description Username Password | EnterpriseDB | Database system. Either EnterpriseDB or PostgreSQL. Descriptive name of the data source. The name of the user that this data source uses to connect to the server. The password of the user associated with this named data source. |
| CPOutput | CPOutput | 0 | Number of seconds before a connection times out (in a connection pooling environment). |
| Port | Port | 5444 | The TCP port that the postmaster is listening on. |
| Protocol | Protocol | 7.4 | If specified, forces the driver to use the given protocol version. |
| Level of Rollback on Errors | Use the Protocol option to specify rollback behavior. | Transaction Level | Specifies how the driver handles errors: 0 - Don't rollback 1 - Rollback the transaction 2 - Rollback the statement |
| Usage Count | UsageCount | 1 | The number of installations using this driver. |
| Read Only | ReadOnly | No | Specifies that the connection is READONLY. |
| Show System Tables | ShowSystemTables | No | If enabled, the driver reports system tables in the result set of the SQLTables() function. |
| OID Options: Show Column | ShowOidColumn | No | If enabled, the SQLColumns() function reports the OID column. |
| OID Options: Fake Index | FakeOidIndex | No | If enabled, the SQLStatistics() function reports that a unique index exists on each OID column. |
| Keyset Query Optimization | Ksqo | On | If enabled, enforces server-side support for keyset queries (generated by the MS Jet database engine). |

| Property | Keyword name | Default value | Description |
|--------------------------|-------------------|---------------|--|
| Recognize Unique Indexes | UniqueIndex | On | If enabled, the SQLStatistics() function will report unique indexes. If not enabled, the SQLStatistics() function reports that indexes allow duplicate values. |
| Use Declare/Fetch | UseDeclareFetch | Off | If enabled, the driver will use server-side cursors. To enable UseDeclareFetch, specify a value of 1; to disable UseDeclareFetch, specify a value of 0. |
| CommLog | CommLog | Off | If enabled, records all client/server traffic in a log file. |
| Parse Statements | Parse | Off | If enabled, the driver parses simple SELECT statements when you call the SQLNumResultCols(), SQLDescribeCol() or SQLColAttributes() functions. |
| Cancel as FreeStmt | CancelAsFreeStmt | Off | If enabled, the SQLCancel() function will call SQLFreeStmt(SQL_Close) on your behalf. |
| MyLog | Debug | Off | If enabled, the driver records its work in a log file. On Windows, the file name is C:mylog<process-id>; and on Linux the file name is /tmp/mylog<username><process-id>.log. |
| Unknown Sizes | UnknownSizes | Maximum | Determines how the SQLDescribeCol() and SQLColAttributes() functions compute the size of a column. Specify 0 to force the driver to report the maximum size allowed for the type; specify 1 to force the driver to report an unknown length or 2 to force the driver to search the result set to find the longest value. Do not specify 2 if you have enabled UseDeclareFetch. |
| Text as LongVarchar | TextAsLongVarChar | 8190 | If enabled, the driver treats TEXT columns as if they are of type SQL_LONGVARCHAR. If disabled, the driver treats TEXT columns as SQL_VARCHAR values. |
| Unknown as Long Varchar | LongVarChar | False | If enabled, the driver treats values of unknown type as SQL_LONGVARCHAR values. If unchecked, the driver will treat values of unknown type as SQL_VARCHAR values. By default, values of unknown type are treated as Y values. |

| Property | Keyword name | Default value | Description |
|---------------------------------|-----------------------|------------------|---|
| Bools as Char | BoolsAsChar | On | If enabled, the driver treats BOOL columns as SQL_CHAR values. If disabled, BOOL columns are treated as SQL_BIT values. |
| Max Varchar | MaxVarcharSize | 255 | If enabled, the driver treats VARCHAR and BPCHAR values longer than MaxVarCharSize as SQL_LONGVARCHAR values |
| Max Long Varchar Size | MaxLongVarcharSize | 8190 | <p>If TextAsLongVarChar is on, the driver reports TEXT values are MaxLongVarcharSize bytes long.</p> <p>If UnknownAsLongVarChar is on, columns of unknown type are MaxLongVarcharSize bytes long; otherwise, they are reported to be MaxVarCharSize bytes in length.</p> |
| Cache Size | Fetch | 100 | Determines the number of rows fetched by the driver when UseDeclareFetch is enabled. |
| SysTable Prefixes | ExtraSysTablePrefixes | dd; | Use the SysTablePrefixes field to specify a semi-colon delimited list of prefixes that indicate that a table is a system table. By default, the list contains dd;. |
| Cumulative Row Count for Insert | MapSqlParcNoBatch | Off/0 | If enabled, the SQLRowCount() function will return a single, cumulative row count for the entire array of parameter settings for an INSERT statement. If disabled, an individual row count will be returned for each parameter setting. By default, this option is disabled. |
| LF<-> CR/LF conversion | LFConversion | System Dependent | The LF<->CR/LF conversion option instructs the driver to convert line-feed characters to carriage-return/line-feed pairs when fetching character values from the server and convert carriage-return/line-feed pairs back to line-feed characters when sending character values to the server. By default, this option is enabled. |

| Property | Keyword name | Default value | Description |
|----------------------------|----------------------|---------------|--|
| Updatable Cursors | UpdatableCursors | Off | Permits positioned UPDATE and DELETE operations using the SQLSetPos() or SQLBulkOperations() functions. |
| Bytea as Long VarBinary | ByteaAsLongVarBinary | Off | If enabled, the driver treats BYTEA values as if they are of type SQL_LONGVARIABLE. If disabled, BYTEA values are treated as SQL_VARIABLE values. |
| Bytea as LO | ByteaAsLO | False | If enabled, the driver treats BYTEA values as if they are large objects. |
| Row versioning | RowVersioning | Off | The Row Versioning option specifies if the driver should include the xmin column when reporting the columns in a table. The xmin value is the ID of the transaction that created the row. You must use row versioning if you plan to create cursors where SQL_CONCURRENCY = SQL_CONCUR_ROWVER. |
| Disallow Premature | DisallowPremature | No/0 | Determines driver behavior if you try to retrieve information about a query without executing the query. If Yes, the driver declares a cursor for the query and fetches the meta-data from the cursor. If No, the driver executes the command as soon as you request any meta-data. |
| True is -1 | TruelsMinus1 | Off/0 | TruelsMinus1 tells the driver to return BOOL values of TRUE as -1. If this option is not enabled, the driver will return BOOL values of TRUE as 1. The driver always returns BOOL values of FALSE as 0. |
| Server side prepare | UseServerSidePrepare | No/0 | If enabled, the driver uses the PREPARE and EXECUTE commands to implement the Prepare/Execute model. |
| Use GSSAPI for GSS request | GssAuthUseGSS | False/0 | If set to True/1, the driver will send a GSSAPI authentication request to the server. Windows only. |

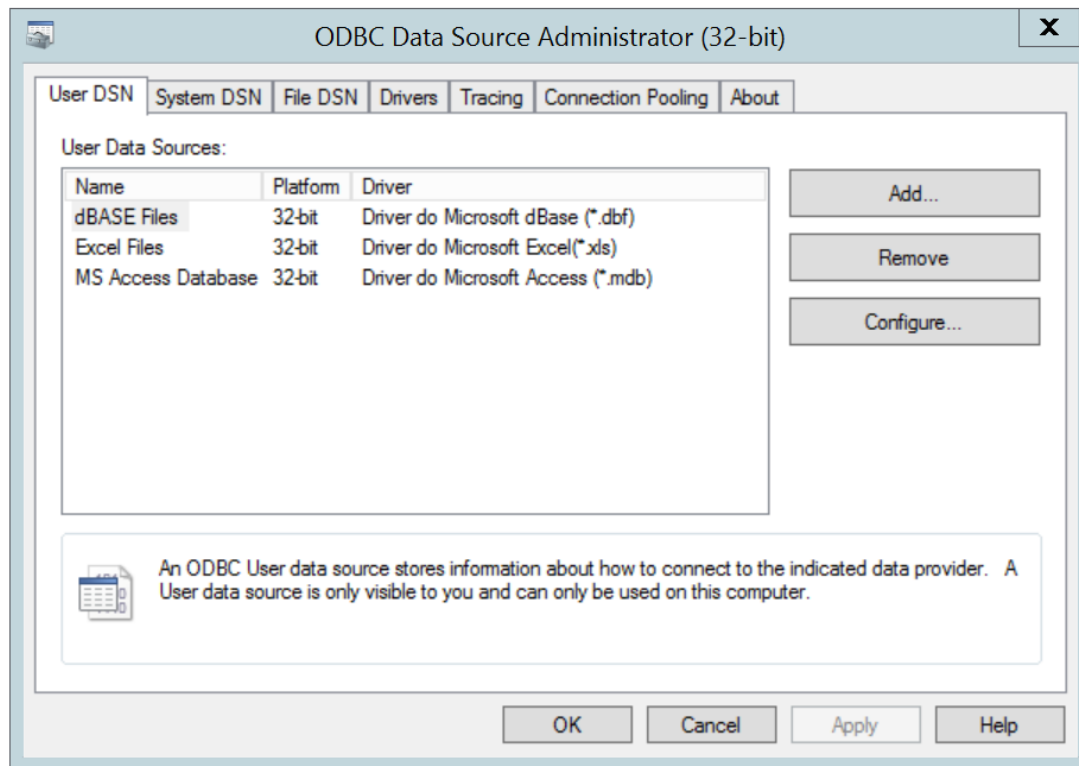
| Property | Keyword name | Default value | Description |
|----------|--------------|---------------|---|
| Int8 As | BI | 0 | <p>The value of BI determines how the driver treats BIGINT values:</p> <p>If -5 as a SQL_BIGINT,</p> <p>If 2 as a SQL_NUMERIC,</p> <p>If 8 as a SQL_DOUBLE,</p> <p>If 4 as a SQL_INTEGER,</p> <p>If 12 as a SQL_VARCHAR,</p> <p>If 0 (on an MS Jet client), as a SQL_NUMERIC,</p> <p>If 0 on any other client, as a SQL_BIGINT.</p> |

| Property | Keyword name | Default value | Description |
|---------------------------------------|------------------------|---------------|---|
| Extra options Connect Settings | AB ConnSettings | 0x0 | 0x1 - Forces the output of short-length formatted connection strings. Specify this option if you are using the MFC CDatabase class. |
| | | | 0x2 - Allows MS Access to recognize PostgreSQL's serial type as AutoNumber type. |
| | | | 0x4 - Return ANSI character types for the inquiries from applications. Specify this option for applications that have difficulty handling Unicode data. |
| | | | 0x8 - If set, NULL dates are reported as empty strings and empty strings are interpreted as NULL dates on input. |
| | | | 0x10 - Determines if SQLGetInfo returns information about all tables, or only accessible tables. If set, only information is returned for accessible tables. |
| | | | 0x20 - If set, each SQL command is processed in a separate network round-trip, otherwise, SQL commands are grouped into as few round-trips as possible to reduce network latency. Contains a semicolon-delimited list of SQL commands that are executed when the driver connects to the server. |
| | Socket | 4096 | Specifies the buffer size that the driver uses to connect to the client. |
| | Lie | Off | If enabled, the driver claims to support unsupported ODBC features. |
| Lowercase Identifier | LowerCaseIdentifier | Off | If enabled, the driver translates identifiers to lowercase. |
| Disable Genetic Optimizer | Optimizer | Yes/1 | Disables the genetic query optimizer. |
| Allow Keyset | UpdatableCursors | Yes/1 | Allow Keyset driven cursors |

| Property | Keyword name | Default value | Description |
|-------------------------------------|--------------|---------------|--|
| SSL mode | SSLMode | Disabled | If libpq (and its dependencies) are installed in the same directory as the EDB-ODBC driver, enabling SSL Mode allows you to use SSL and other utilities. |
| Force Abbreviated Connection String | CX | No/0 | Enables the option to force abbreviation of connection string. |
| Fake MSS | FakeOidIndex | No/0 | Impersonates MS SQL Server enabling MS Access to recognize PostgreSQL's serial type as AutoNumber type. |
| BDE Environment | BDE | No/0 | Enabling this option tunes EDB-ODBC to cater to Borland Database Engine compliant output (related to Unicode). |
| XA_Opt | INI_XAOPT | Yes/1 | If enabled, calls to SQL_TABLES only include user-accessible tables. |

Adding a Data Source Definition in Windows

The Windows ODBC **Data Source Administrator** is a graphical interface that creates named data sources. You can open the **ODBC Data Source Administrator** by navigating to the **Control Panel**, opening the **Administrative Tools** menu, and double-clicking the appropriate **ODBC Data Sources** icon (32- or 64- bit).



The Windows Data Source Administrator

Click the **Add** button to open the **Create New Data Source** dialog. Choose **EnterpriseDB (ANSI)** or **EnterpriseDB (UNICODE)** from the list of drivers and click **Finish**.



The Create New Data Source dialog

The EnterpriseDB ODBC Driver dialog opens.



The image shows the 'EnterpriseDB ODBC Driver' configuration window. It contains the following fields and controls:

- Data Source:** EnterpriseDB
- Description:** My EDB-AS Datasource
- Database:** edb
- SSL Mode:** disable (dropdown menu)
- Server:** 127.0.0.1
- Port:** 5444
- User Name:** enterisedb
- Password:** masked with 10 dots
- Options:** A group box containing two buttons: 'Datasource' and 'Global'.
- Action Buttons:** 'Test', 'Save', and 'Cancel'.

Define the data source

Use the fields on the dialog to define the named data source:

- Enter the Database name in the **Database** field.
- Enter the host name or IP address of Advanced Server in the **Server** field.
- Enter the name of a user in the **User Name** field.
- Enter a descriptive name for the named data source in the **Description** field.
- If libpq is installed in the same directory as the EDB-ODBC driver, the drop-down listbox next to the **SSL Mode** label will be active, allowing you to use SSL and other Advanced Server utilities.
- Accept the default port number (5444), or enter an alternative number in the **Port** field.
- Enter the password of the user in the **Password** field.

Use the **Datasource** button (located in the **Options** box) to open the **Advanced Options** dialog and specify connection properties.

The **Global** button opens a dialog on which you can specify logging options for the EDB-ODBC driver (not the data source, but the driver itself).



Page 1 of the Advanced Options dialog

- Check the box next to **Disable Genetic Optimizer** to disable the genetic query optimizer. By default, the query optimizer is **on**.
- Check the box next to **KSQO (Keyset Query Optimization)** to enable server-side support for keyset queries. By default, **Keyset Query Optimization** is **on**.
- Check the box next to **Recognize Unique Indexes** to force the **SQLStatistics()** function to report unique indexes; if the option is not checked, the **SQLStatistics()** function will report that all indexes allow duplicate values. By default, **Recognize Unique Indexes** is **on**.
- Check the box next to **Use Declare/Fetch** to specify that the driver should use server-side cursors whenever your application executes a **SELECT** command. By default, **Use Declare/Fetch** is **off**.
- Check the box next to **CommLog (C:\psqlodbc_XXXX.log)** to record all client/server traffic in a log file. By default, logging is **off**.
- Check the box next to **Parse Statements** to specify that the driver (rather than the server) should attempt to parse simple **SELECT** statements when you call the **SQLNumResultCols()**, **SQLDescribeCol()**, or **SQLColAttributes()** function. By default, this option is **off**.
- Check the box next to **Cancel as FreeStmt (Exp)** to specify that the **SQLCancel()** function should call **SQLFreeStmt(SQLClose)** on your behalf. By default, this option is **off**.
- Check the box next to **MyLog (C:\mylog_XXXX.log)** to record a detailed record of driver activity in a log file. The log file is named **c:\mylog_ \ *process-id*.log**. By default, logging is **off**.

The radio buttons in the Unknown Sizes box specify how the **SQLDescribeCol()** and **SQLColAttributes()** functions compute the size of a column of unknown type (see Section **Supported Data Types** for a list of known data types).

- Choose the button next to **Maximum** to specify that the driver report the maximum size allowed for a **VARCHAR** or **LONGVARCHAR** (dependent on the **Unknowns as LongVarChar** setting). If **Unknowns as LongVarChar** is enabled, the driver returns the maximum size of a **LONGVARCHAR** (specified in the **Max LongVarChar** field in the **Miscellaneous** box). If **Unknowns as LongVarChar** is not enabled, the driver returns the size specified in the **Max VarChar** field (in the **Miscellaneous** box).
- Choose the button next to **Don't know** to specify that the driver report a length of "unknown".
- Choose the button next to **Longest** to specify that the driver search the result set and report the longest value found. (Note: you should not specify **Longest** if **UseDeclareFetch** is enabled.)

The properties in the **Data Type Options** box determine how the driver treats columns of specific types:

- Check the box next to **Text as LongVarChar** to treat **TEXT** values as if they are of type **SQL_LONGVARCHAR**. If the box is not checked, the driver will treat **TEXT** values as **SQL_VARCHAR** values. By default, **TEXT** values are treated as **SQL_LONGVARCHAR** values.
- Check the box next to **Unknowns as LongVarChar** to specify that the driver treat values of unknown type as **SQL_LONGVARCHAR** values. If unchecked, the driver will treat values of unknown type as **SQL_VARCHAR** values. By default, values of unknown type are treated as **SQL_VARCHAR** values.
- Check the box next to **Bools as Char** to specify that the driver treat **BOOL** values as **SQL_CHAR** values. If unchecked, **BOOL** values are treated as **SQL_BIT** values. By default, **BOOL** values are treated as **SQL_CHAR** values.

You can specify values for some of the properties associated with the named data source in the fields in the **Miscellaneous** box:

- Indicate the maximum length allowed for a **VARCHAR** value in the **Max VarChar** field. By default, this value is set to **255**.
- Enter the maximum length allowed for a **LONGVARCHAR** value in the **Max LongVarChar** field. By default, this value is set to **8190**.
- Specify the number of rows fetched by the driver (when **UseDeclareFetch** is enabled) in the **Cache Size** field. The default value is **100**.
- Use the **SysTablePrefixes** field to specify a semi-colon delimited list of prefixes that indicate that a table is a system table. By default, the list contains **dd_;**.

You can reset the values on this dialog to their default settings by choosing the **Defaults** button.

Click the **Apply** button to apply any changes to the data source properties, or the **Cancel** button to exit the dialog without applying any changes. Choose the **OK** button to apply any changes to the dialog and exit.

Select the **Page 2** button (in the upper-left hand corner of the **Advanced Options** dialog) to access a second set of advanced options.



Page 2 of the Advanced Options dialog

- Check the box next to **Read Only** to prevent the driver from executing the following commands: **INSERT**, **UPDATE**, **DELETE**, **CREATE**, **ALTER**, **DROP**, **GRANT**, **REVOKE** or **LOCK**. Invoking the **Read Only** option also prevents any calls that use ODBC's procedure call escape syntax (**call=procedure-name?**). By default, this option is **off**.
- Check the box next to **Show System Tables** to include system tables in the result set of the **SQLTables()** function. If the option is enabled, the driver will include any table whose name starts with **pg** or any of the prefixes listed in the **SysTablePrefixes** field of **Page 1** of the **Advanced Options** dialog. By default, this option is **off**.
- Check the box next to **Show sys/dbo Tables [Access]** to access objects in the **sys** schema and **dbo** schema through the ODBC data source. By default, this option is enabled (checked).
- Check the box next to **Cumulative Row Count for Insert** to cause a single, cumulative row count to be returned for the entire array of parameter settings for an **INSERT** statement when a call to the **SQLRowCount()** method is performed. If this option is not enabled (the box is not checked), then an individual row count is available for each parameter setting in the array, and thus, a call to **SQLRowCount()** returns the count for the last inserted row.
- Check the box next to **LF<->CR/LF** conversion to instruct the driver to convert line-feed characters to carriage-return/line-feed pairs when fetching character values from the server and convert carriage-return/line-feed pairs back to line-feed characters when sending character values to the server. By default, this option is enabled.
- Check the box next to **Updatable Cursors** to specify that the driver should permit positioned **UPDATE** and **DELETE** operations with the **SQLSetPos()** or **SQLBulkOperations()** functions. By default, this option is enabled.
- Check the box next to **bytea as LO** to specify that the driver should treat **BYTEA** values as if they are **SQL_LONGVARBINARY** values. If the box is not checked, EDB-ODBC will treat **BYTEA**

values as if they are `SQL_VARBINARY` values. By default, `BYTEA` values are treated as `SQL_VARBINARY` values.

- Check the box next to `Row Versioning` to include the `xmin` column when reporting the columns in a table. The `xmin` column is the ID of the transaction that created the row. You must use row versioning if you plan to create cursors where `SQL_CONCURRENCY = SQL_CONCUR_ROWVER`. By default, `Row Versioning` is off.
- Check the box next to `Disallow Premature` to specify that the driver should retrieve meta-data about a query (i.e., the number of columns in a result set, or the column types) without actually executing the query. If this option is not specified, the driver executes the query when you request meta-data about the query. By default, `Disallow Premature` is off.
- Check the box next to `True is -1` to tell the driver to return `BOOL` values of `True` as a `-1`. If this option is not enabled, the driver will return `BOOL` values of `True` as `1`. The driver always returns `BOOL` values of `False` as `0`.
- Check the box next to `Server side prepare` to tell the driver to use the `PREPARE` and `EXECUTE` commands to implement the `Prepare/Execute` model. By default, this box is checked.
- Check the box next to `use gssapi for GSS request` to instruct the driver to send a GSSAPI connection request to the server.
- Enter the database system (either `EnterpriseDB` or `PostgreSQL`) in the `dbms_name` field. The value entered here is returned in the `SQL_DBMS_NAME` argument when the `SQLGetInfo()` function is called. The default is `EnterpriseDB`.

Use the radio buttons in the `Int8 As` box to specify how the driver should return `BIGINT` values to the client. Select the radio button next to `default` to specify the default type of `NUMERIC` if the client is MS Jet, `BIGINT` if the client is any other ODBC client. You can optionally specify that the driver return `BIGINT` values as a `bigint` (`SQL_BIGINT`), `numeric` (`SQL_NUMERIC`), `varchar` (`SQL_VARCHAR`), `double` (`SQL_DOUBLE`), or `int4` (`SQL_INTEGER`).

The default value of the `Extra Opts` field is `0x0`. `Extra Opts` may be:

Option Specifies

| | |
|------|--|
| 0x1 | Forces the output of short-length formatted connection string. Select this option when you are using the MFC CDatabase class. |
| 0x2 | Allows MS Access to recognize PostgreSQL's serial type as AutoNumber type. |
| 0x4 | Return ANSI character types for the inquiries from applications. Select this option for applications that have difficulty handling Unicode data. |
| 0x8 | If set, NULL dates are reported as empty strings and empty strings are interpreted as NULL dates on input. |
| 0x10 | Determines if SQLGetInfo returns information about all tables, or only accessible tables. If set, only information is returned for accessible tables. |
| 0x20 | If set, each SQL command is processed in a separate network round-trip, otherwise, SQL commands are grouped into as few round-trips as possible to reduce network latency. |

The **Protocol** box contains radio buttons that tell the driver to interact with the server using a specific front-end/back-end protocol version. By default, the **Protocol** selected is **7.4+**; you can optionally select from versions **6.4+**, **6.3** or **6.2**.

The **Level of Rollback on errors** box contains radio buttons that specify how the driver handles error handling:

| Option | Specifies |
|-------------|--|
| Transaction | If the driver encounters an error, it will rollback the current transaction. |
| Statement | If the driver encounters an error, it will rollback the current statement. |
| Nop | If the driver encounters an error, you must manually rollback the current transaction before the application can continue. |

The **OID Options** box contains options that control the way the driver exposes the OID column contained in some tables:

- Check the box next to **Show Column** to include the **OID** column in the result set of the **SQLColumns()** function. If this box is not checked, the **OID** column is hidden from **SQLColumns()**.
- Check the box next to **Fake Columns** to specify that the **SQLStatistics()** function should report that a unique index exists on each **OID** column.

Use the **Connect Settings** field to specify a list of parameter assignments that the driver will use when opening this connection. Any configuration parameter that you can modify with a **SET** statement can be included in the semi-colon delimited list. For example:

```
set search_path to company1,public;
```

When you've defined the connection properties for the named data source, click the **Apply** button to apply the options; you can optionally exit without saving any options by choosing **Cancel**. Select the **OK** button to save the options and exit.

Choose the **Global** button (on the **EnterpriseDB ODBC Driver** dialog) to open the **Global Settings** dialog. The options on this dialog control logging options for the EDB-ODBC driver. Use this dialog to enforce logging when the driver is used without a named data source, or for logging driver operations that occur before the connection string is parsed.



The Global Settings dialog

- Check the box next to the **CommLog** field to record all client/server traffic in a log file. The logfile is named **C:\psqlodbc_process-id** where **process-id** is the name of the process in use.
- Check the box next to the **Mylog** field to keep a logfile of the driver's activity. The logfile is named **c:\mylog_process-id** where **process-id** is the name of the process in use.
- Specify a location for the logfiles in the **Folder for logging** field.

When you've entered the connection information for the named data source, click the **Test** button to verify that the driver manager can connect to the defined data source.



The Connection is successful

Click the OK button to exit **Connection Test** dialog. If the connection is successful, click the **Save** button to save the named data source. If there are problems establishing a connection, adjust the parameters and test again.

Adding a Data Source Definition in Linux

The Linux **ODBC Administrator** is a graphical tool that is distributed with unixODBC; you can use the **ODBC Administrator** to manage ODBC drivers and named resources. To add the ODBC Administrator to your system, open a terminal window, assume superuser privileges, and enter:

```
yum install unixODBC
```

followed by:

```
yum install unixODBC-kde
```

To invoke the **ODBC Administrator**, open a terminal window and enter `ODBCConfig`.



The unixODBC Data Source Administrator

When you install the Advanced Server **Connectors** component, the EDB-ODBC driver is added to the list of drivers in the ODBC Administrator. Click **Advanced**, and then select the **Drivers** tab to verify that the **enterprisedb** driver appears in the list.



The Drivers tab shows the installed EDB-ODBC driver

If the EDB-ODBC driver does not appear in the list of drivers, you can add it using the **ODBC Administrator**. To add a driver definition, select the **Drivers** tab, and click **Add**. The **Driver Properties (new)** window opens, as shown below:



The Driver Properties window

Complete the **Driver Properties** window to register the EDB-ODBC driver with the driver manager:

- Add a unique name for the driver to the **Name** field.
- Add a driver description to the **Description** field.
- Add the path to the location of the EDB-ODBC driver in the **Driver** field. By default, the complete path to the driver is:

```
/usr/edb/odbc/lib/edb-odbc.so
```

- Add the path to the location of the EDB-ODBC driver setup file in the **Setup** field. By default, the complete path to the driver setup file is:

```
/usr/edb/odbc/lib/libodbcdbS.so
```

When you've described the driver properties for the EDB-ODBC driver, click **OK**. The ODBC Data Source Administrator window now includes the EDB-ODBC driver in the list of available ODBC drivers.

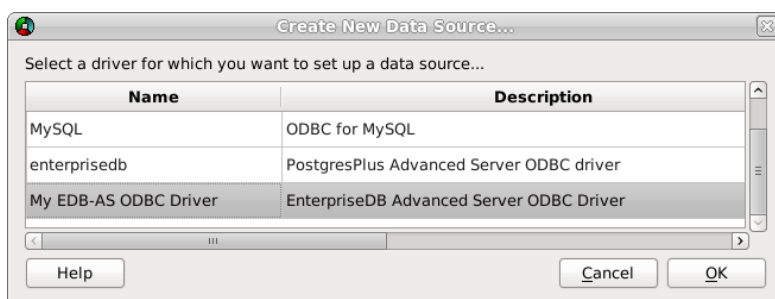


The Drivers tab shows the new driver definition

With the EDB-ODBC driver available to the driver manager, you can add a data source. Click the **Data Source** Names option in the left panel, and then choose the appropriate DSN tab for the type of data source name you would like to add:

- Choose the **User** tab to add a named data source that is available only to the current user (the data source will be stored in `/user/.odbc.ini`).
- Choose the **System** tab add a named data source that is available to all users. All system data sources are stored in a single file (usually `/etc/odbc.ini`).
- Choose the **File** tab to add a named data source that is available to all users, but that is stored in a file of your choosing.

Select the appropriate tab and click **Add**. The **Create a New Data Source...** window opens, as shown below:



Select a driver for the named data source

Select the EDB-ODBC driver from the list, and click **OK** to open the **Data Source Properties** window.

Complete the **Data Source Properties (new)** window, specifying the connection properties

for the EDB-ODBC driver.

| Name | Value |
|------------------|--|
| Name | My EDB-AS ODBC Driver |
| Description | EDB Postgres Advanced Server Data Source |
| Driver | enterprisedb |
| Trace | No |
| TraceFile | |
| Database | edb |
| Servename | localhost |
| Username | enterprisedb |
| Password | my_password |
| Port | 5444 |
| Protocol | 7.4 |
| ReadOnly | No |
| RowVersioning | No |
| ShowSystemTables | No |
| ShowOidColumn | No |
| FakeOidIndex | No |
| ConnSettings | |

Buttons: Help, Cancel, OK

The Data Source Properties window

- Enter the data source name in the **Name** field.
- Enter a description of the named data source in the **Description** field.
- The unixODBC driver includes a trace utility that records the sequence of calls made an ODBC application to a log file. Specify **Yes** in the **Trace** field to turn the trace utility on. Note that using the trace utility can slow down an application.
- Use the **TraceFile** field to specify a file to receive information returned by the **Trace** utility.
- Enter the name of the Advanced Server database in the **Database** field.
- Enter the host name or IP address of Advanced Server in the **Servename** field.
- Enter the name of a user in the **Username** field.
- Enter the password for the user in the **Password** field.
- Enter a port number (or accept the default value of 5444) in the **Port** field.
- Use the **Protocol** field to specify a front-end/back-end protocol version; the default value is 7.4. You can optionally select from protocol versions 7.4, 6.4, 6.3 or 6.2.
- Use the **ReadOnly** field to specify **Yes** to prevent the driver from executing the following commands: **INSERT**, **UPDATE**, **DELETE**, **CREATE**, **ALTER**, **DROP**, **GRANT**, **REVOKE** or **LOCK**. Enabling the **Read Only** option also prevents any calls that use the ODBC procedure call escape syntax (**call=procedure-name?**). By default, **ReadOnly** is set to **No**.
- Use the **RowVersioning** field to specify **Yes** if the driver should include the **xmin** column when reporting the columns in a table. The **xmin** column is the ID of the transaction that created the row. You must use row versioning if you plan to create cursors where **SQL_CONCURRENCY = SQL_CONCUR_ROWVER**. By default, **Row Versioning** is set to **No**.
- Use the **ShowSystemTables** field to specify **Yes** if the driver should include system tables in the result set of the **SQLTables()** function. By default, this field is set to **No**.
- Use the **ShowOidColumn** field to specify **Yes** if the driver should include the **OID** column in

the result set of the `SQLColumns()` function. If `ShowOidColumn` is set to `No`, the `OID` column is hidden from `SQLColumns()`. By default, this option is set to `No`.

- Use the `FakeOidIndex` field to specify Yes if the `SQLStatistics()` function should report that a unique index exists on each `OID` column. This is useful when your application needs a unique identifier and your table doesn't include one. The default value is `No`.
- Use the `ConnSettings` field to specify a list of parameter assignments that the driver will use when opening this connection.

When you've defined the connection properties, click `OK`.

The new data source is added to the list of data source names:



The new data source is included on the Data Source Names list

6 EDB-ODBC Driver Functionality

You can use ODBC functions to query ODBC for specific information about the various attributes of the connection between EDB-ODBC and the server.

- `SQLGetInfo()` returns information about the EDB-ODBC driver and Advanced Server.
- `SQLGetEnvAttr()` returns information about ODBC environment attributes.
- `SQLGetConnectAttr()` returns information about attributes specific to an individual connection.
- `SQLGetStmtAttr()` returns information about the attributes specific to an individual

statement.

You can also use ODBC functions to set various attributes of the objects that you use to interface with ODBC:

- Use the `SQLSetConnectAttr()` function to set connection attributes.
- Use the `SQLSetEnvAttr()` function to set environment attributes.
- Use the `SQLSetStmtAttr()` function to set statement attributes.

SQLGetInfo()

The ODBC `SQLGetInfo()` function returns information about the EDB-ODBC driver and Advanced Server. You must have an open connection to call `SQLGetInfo()`, unless you specify `SQL_ODBC_VER` as the `info_type`. The signature for `SQLGetInfo()` is:

```
SQLRETURN SQLGetInfo
(
    SQLHDBC conn_handle , // Input
    SQLUSMALLINT info_type , // Input
    SQLPOINTER info_pointer , // Output
    SQLSMALLINT buffer_len , // Input
    SQLSMALLINT * string_length_pointer // Output
);
```

- `conn_handle` The connection handle.
- `info_type` The type of information `SQLGetInfo()` is retrieving.
- `info_pointer` A pointer to a memory buffer that will hold the retrieved value.

If the `info_type` argument is `SQL_DRIVER_HDESC` or `SQL_DRIVER_HSTMT`, the `info_pointer` argument is both `Input` and `Output`.

- `buffer_len` is the length of the allocated memory buffer pointed to by `info_pointer`. If `info_pointer` is `NULL`, `buffer_len` is ignored. If the returned value is a fixed size, `buffer_len` is ignored. `buffer_len` is only used if the requested value is returned in the form of a character string.
- `string_length_pointer` is a pointer to an `SQLSMALLINT` value. `SQLGetInfo()` writes the size of the requested value in this integer.

A typical usage is to call `SQLGetInfo()` with a `NULL info_pointer` to obtain the length of the requested value, allocate the required number of bytes, and then call `SQLGetInfo()` again (providing the address of the newly allocated buffer) to obtain the actual value. The first call retrieves the number of bytes required to hold the value; the second call retrieves the value.

If the size of the returned value exceeds `buffer_len`, the information is truncated and `NULL` terminated. If the returned value is a fixed size, `string_length` is ignored (and the size of the requested value is not provided by `SQLGetInfo()`).

`SQLGetInfo()` writes information in one of the following formats:

- a `SQLINTEGER` bitmask
- a `SQLINTEGER` flag
- a `SQLINTEGER` binary value
- a `SQLUSMALLINT` value
- a `NULL` terminated character string

`SQLGetInfo()` returns `SQL_SUCCESS`, `SQL_SUCCESS_WITH_INFO`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

The following table lists the information returned by EDB-ODBC about the Advanced Server connection:

| SQL info_type Argument and Description | EDB_ODBC/Advanced Server Returns: |
|---|---|
| SQL_ACCESSIBLE_PROCEDURES: Indicates if procedures returned by <code>SQLProcedures()</code> can be executed by the application. | Returns N. Some procedures executed by the <code>SQLProcedures()</code> function may be executed by the application. |
| SQL_ACCESSIBLE_TABLES: Indicates if the user has SELECT privileges on all table names returned by <code>SQLTables()</code> . | Returns N. The user may not have select privileges on one or more tables returned by the <code>SQLTables()</code> function. |
| SQL_ACTIVE_CONNECTIONS prev. SQL_MAX_DRIVER_CONNECTIONS: Indicates the maximum number of connections EDB-ODBC can support. | Returns 0. There is no specified limit to the number of connections allowed. |
| SQL_ACTIVE_ENVIRONMENTS: The number of active environments EDB-ODBC can support. | Returns 0. There is no specified limit to the number of environments allowed. |
| SQL_ACTIVE_STATEMENTS prev. SQL_MAX_CONCURRENT_ACTIVITIES: Indicates the maximum number of active statements EDB-ODBC can support. | Returns 0. There is no specified limit to the number of active statements allowed. |
| SQLAggregate_FUNCTION: Identifies the aggregate functions supported by the server and driver. | Returns SQL_AF_ALL |
| SQL_ALTER_DOMAIN: Identifies the ALTER DOMAIN clauses supported by the server. | Returns 0. ALTER DOMAIN clauses are not supported. |

| SQL info_type Argument and Description | EDB_ODBC/Advanced Server Returns: |
|---|--|
| SQL_ALTER_TABLE: Identifies the ALTER TABLE clauses supported by the server. | Returns SQL_AT_ADD_COLUMN, SQL_AT_DROP_TABLE_CONSTRAINT_CASCADE, SQL_AT_DROP_TABLE_CONSTRAINT, SQL_AT_CONSTRAINT_INITIALLY_DEFERRED, SQL_AT_CONSTRAINT_INITIALLY_IMMEDIATE, SQL_AT_CONSTRAINT_DEFERRABLE |
| SQL_ASYNC_MODE: Level of Asynchronous Mode Supported by EDB-ODBC. | Returns SQL_AM_NONE. Asynchronous mode is not supported. |
| SQL_BATCH_ROW_COUNT: Indicates how the driver returns row counts. | Returns SQL_BRC_EXPLICIT. Row Counts are available when executed by calling SQLExecute or SQLExecDirect. |
| SQL_BATCH_SUPPORT: Indicates support for batch statement execution. | Returns: SQL_BS_SELECT_EXPLICIT, SQL_BS_ROW_COUNT_EXPLICIT. The driver supports explicit batches with result set and row count generating statements. |
| SQL_BOOKMARK_PERSISTENCE: Indicates level of support for bookmarks. | Returns: SQL_BP_DELETE, SQL_BP_TRANSACTION, SQL_BP_UPDATE, SQL_BP_SCROLL. |
| SQL_CATALOG_LOCATION Now SQL_QUALIFIER_LOCATION: Indicates the position of the catalog in a qualified table name. | Returns SQL_CL_START. The catalog portion of a qualified table name is at the beginning of the name. |
| SQL_CATALOG_NAME Now SQL_QUALIFIER_NAME: Indicates support for catalog names. | Returns Y. The server supports catalog names. |
| SQL_CATALOG_NAME_SEPARATOR Now SQL_QUALIFIER_NAME_SEPARATOR: Character separating the catalog name from the adjacent name element. | Returns '.' The server expects a '.' character between the qualifier and the table name. |
| SQL_CATALOG_TERM Now SQL_QUALIFIER_TERM: The term used to describe a catalog. | Returns catalog. |
| SQL_CATALOG_USAGE Now SQL_QUALIFIER_USAGE: Indicates the SQL statements that may refer to catalogs. | Returns SQL_CU_DML_STATEMENTS. Catalog names can be used in SELECT, INSERT, UPDATE, DELETE, SELECT FOR UPDATE and positioned UPDATE and DELETE statements. |
| SQL_COLLATION_SEQ: Returns the name of the Collation Sequence. | Returns an empty string. The name of the default collation is unknown. |
| SQL_COLUMN_ALIAS: Indicates server support for column aliases. | Returns Y. The server supports column aliases. |

| SQL info_type Argument and Description | EDB_ODBC/Advanced Server Returns: |
|---|--|
| SQL_CONCAT_NULL_BEHAVIOR: Indicates how the server handles concatenation of NULL values. | Returns SQL_CB_NON_NULL. Concatenation of a NULL value and a non NULL value will result in a NULL value. |
| SQL_CONVERT_BIGINT: Indicates conversion support from the BIGINT type using the CONVERT function. | Returns 0. The server does not support conversion. |
| SQL_CONVERT_BINARY: Indicates conversion support from the BINARY type using the CONVERT function. | Returns 0. The server does not support conversion. |
| SQL_CONVERT_BIT: Indicates conversion support from the BIT type using the CONVERT function. | Returns: SQL_CVT_INTEGER, SQL_CVT_BIT. |
| SQL_CONVERT_CHAR: Indicates conversion support from the CHAR type using the CONVERT function. | Returns 0. The server does not support conversion. |
| SQL_CONVERT_DATE: Indicates conversion support from the DATE type using the CONVERT function. | Returns 0. The server does not support conversion. |
| SQL_CONVERT_DECIMAL: Indicates conversion support from the DECIMAL type using the CONVERT function. | Returns 0. The server does not support conversion. |
| SQL_CONVERT_DOUBLE: Indicates conversion support from the DOUBLE type using the CONVERT function. | Returns 0. The server does not support conversion. |
| SQL_CONVERT_FLOAT: Indicates conversion support from the FLOAT type using the CONVERT function. | Returns 0. The server does not support conversion. |
| SQL_CONVERT_FUNCTIONS: Lists the scalar conversion functions supported by the server and driver using the CONVERT function. | Returns: SQL_FN_CVT_CONVERT. |
| SQL_CONVERT_INTEGER: Lists the conversion support from the INTEGER type using the CONVERT function. | Returns: SQL_CVT_INTEGER, SQL_CVT_BIT. |
| SQL_CONVERT_INTERVAL_DAY_TIME: Indicates conversion support from the INTERVAL_DAY_TIME type using the CONVERT function. | This info_type is not currently supported. |
| SQL_CONVERT_INTERVAL_YEAR_MONTH: Indicates conversion support from the INTERVAL_YEAR_MONTH type using the CONVERT function. | This info_type is not currently supported. |

| SQL info_type Argument and Description | EDB_ODBC/Advanced Server Returns: |
|---|--|
| SQL_CONVERT_LONGVARBINARY: Indicates conversion support for the LONG_VARBINARY type using the CONVERT function. | Returns 0. The server does not support conversion. |
| SQL_CONVERT_LONGVARCHAR: Indicates conversion support for the LONGVARCHAR type using the CONVERT function. | Returns 0. The server does not support conversion. |
| SQL_CONVERT_NUMERIC: Indicates conversion support for the NUMERIC type using the CONVERT function. | Returns 0. The server does not support conversion. |
| SQL_CONVERT_REAL: Indicates conversion support for the REAL type using the CONVERT function | Returns 0. The server does not support conversion. |
| SQL_CONVERT_SMALLINT: Indicates conversion support for the SMALLINT type using the CONVERT function. | Returns: SQL_CVT_INTEGER, SQL_CVT_BIT. |
| SQL_CONVERT_TIME: Indicates conversion support for TIME type using the CONVERT function. | Returns 0. The server does not support conversion. |
| SQL_CVT_TIMESTAMP: Indicates conversion support for TIMESTAMP type using the CONVERT function. | Returns 0. The server does not support conversion. |
| SQL_CONVERT_TINYINT: Indicates conversion support for the TINYINT type using the CONVERT function. | Returns: SQL_CVT_INTEGER, SQL_CVT_BIT. |
| SQL_CONVERT_VARBINARY: Indicates conversion support for the VARBINARY type using the CONVERT function. | Returns 0. The server does not support conversion. |
| SQL_CONVERT_VARCHAR: Indicates conversion support for VARCHAR type using the CONVERT function. | Returns: SQL_CVT_INTEGER, SQL_CVT_BIT. |
| SQL_CONVERT_WCHAR: Indicates conversion support for the WCHAR type using the CONVERT function. | This info_type is valid only when using the Unicode driver. Returns 0. The server does not support conversion. |
| SQL_CONVERT_WLONGVARCHAR: Indicates conversion support for the WLONGVARCHAR type using the CONVERT function. | This info_type is valid only when using the Unicode driver. Returns 0. The server does not support conversion. |
| SQL_CONVERT_WVARCHAR: Indicates conversion support for the WVARCHAR type using the CONVERT function. | This info_type is valid only when using the Unicode driver. Returns 0. The server does not support conversion. |
| SQL_CORRELATION_NAME: Indicates server support for correlation names. | Returns SQL_CN_ANY. Correlation names are supported and can be any valid name. |

| SQL info_type Argument and Description | EDB_ODBC/Advanced Server Returns: |
|---|---|
| SQL_CREATE_ASSERTION: Indicates support for the CREATE ASSERTION statement. | Returns 0. The CREATE ASSERTION statement is not supported. |
| SQL_CREATE_CHARACTER_SET: Indicates support for CREATE CHARACTER statement. | Returns 0. The CREATE CHARACTER statement is not supported. |
| SQL_CREATE_COLLATION: Indicates support for the CREATE COLLATION. | Returns 0. The CREATE COLLATION statement is not supported. |
| SQL_CREATE_DOMAIN: Indicates support for the CREATE DOMAIN statement. | Returns 0. The CREATE DOMAIN statement is not supported. |
| SQL_CREATE_SCHEMA: Indicates support for the CREATE SCHEMA statement. | Returns: SQL_CS_CREATE_SCHEMA, SQL_CS_AUTHORIZATION. |
| SQL_CREATE_TABLE: Indicates support for the CREATE TABLE statement. | Returns: SQL_CT_CREATE_TABLE, SQL_CT_GLOBAL_TEMPORARY, SQL_CT_CONSTRAINT_INITIALLY_DEFERRED, SQL_CT_CONSTRAINT_INITIALLY_IMMEDIATE, SQL_CT_CONSTRAINT_DEFERRABLE, SQL_CT_COLUMN_CONSTRAINT, SQL_CT_COLUMN_DEFAULT, SQL_CT_TABLE_CONSTRAINT, SQL_CT_CONSTRAINT_NAME_DEFINITION |
| SQL_CREATE_TRANSLATION: Indicates support for the CREATE TRANSLATION statement. | Returns 0. The CREATE TRANSLATION statement is not supported. |
| SQL_CREATE_VIEW: Indicates support for the CREATE VIEW statement. | Returns SQL_CV_CREATE_VIEW. |
| SQL_CURSOR_COMMIT_BEHAVIOR: Indicates how a COMMIT operation affects the cursor. | Returns SQL_CB_PRESERVE. Cursors are unchanged, and can continue to fetch data. |
| SQL_CURSOR_ROLLBACK_BEHAVIOR: Indicates the server behavior after a ROLLBACK operation. | Returns SQL_CB_PRESERVE. Cursors are unchanged, and can continue to fetch data. |
| SQL_CURSOR_SENSITIVITY: Indicates how the server synchronizes changes to a result set. | This info_type is not currently supported. |
| SQL_DATA_SOURCE_NAME: Returns the server name used during connection. | The value returned is determined by the connection properties. |
| SQL_DATA_SOURCE_READ_ONLY: Indicates if the connection is in READ ONLY mode. | The value returned is determined by the connection properties. |
| SQL_DATABASE_NAME: Returns the name of the database. | The value returned is determined by the connection properties. |
| SQL_DATETIME_LITERALS: Indicates the DATETIME LITERALS supported by the server. | This info_type is not supported. |

| SQL info_type Argument and Description | EDB_ODBC/Advanced Server Returns: |
|---|---|
| SQL_DBMS_NAME: Returns the name of the DBMS system. | Returns the value given by the dbms_name parameter from the odbc.ini file on Linux or the dbms_name field of page 2 of the Advanced Options dialog box when defining a data source in Windows. The default is EnterpriseDB. |
| SQL_DBMS_VER: Returns the server version. | Determined by the server. |
| SQL_DDL_INDEX: Indicates support for creating and dropping indexes. | Returns: SQL_DI_CREATE_INDEX, SQL_DI_DROP_INDEX. |
| SQL_DEFAULT_TXN_ISOLATION: Indicates support for transaction isolation by the server. | Returns TXN_READ_COMMITTED. Non-repeatable or phantom reads are possible; Dirty reads are not. |
| SQL_DESCRIBE_PARAMETER: Indicates support for the DESCRIBE INPUT statement. | Returns N. The DESCRIBE INPUT statement is not supported. |
| SQL_DM_VER: The version of the Driver Manager. | Determined by driver manager. |
| SQL_DRIVER_HDBC: The Driver's connection handle. | Returns an SQLULEN value that contains the driver's connection handle. |
| SQL_DRIVER_HDESC: The Driver descriptor handle. | Returns an SQLULEN value that contains driver's descriptor handle. |
| SQL_DRIVER_HENV: The Driver's environment handle. | Returns an SQLULEN value that contains the driver's environment handle. |
| SQL_DRIVER_HLIB: The Driver handle. | Returns an SQLULEN value that contains the library handle (returned to the ODBC driver manager when the manager loaded the driver). |
| SQL_DRIVER_HSTMT: The Driver's statement handle. | Returns an SQLULEN value that contains the driver's statement handle. |
| SQL_DRIVER_NAME: The name of the driver. | Returns EDB-ODBC.DLL |
| SQL_DRIVER_ODBC_VER: Identifies the ODBC version that the driver supports. | Returns 03.50 |
| SQL_DRIVER_VER: Identifies the driver version. | Returns 9.0.0.6 |
| SQL_DROP_ASSERTION: Lists the DROP ASSERTION clauses supported by the server. | Returns 0 |
| SQL_DROP_CHARACTER_SET: Lists the DROP CHARACTER clauses supported by the server. | Returns 0 |
| SQL_DROP_COLLATION: Lists the DROP COLLATION clauses supported by the server. | Returns 0 |
| SQL_DROP_DOMAIN: Lists the DROP DOMAIN clauses supported by the server. | Returns 0 |
| SQL_DROP_SCHEMA: Lists the DROP SCHEMA clauses supported by the server. | Returns: SQL_DS_DROP_SCHEMA, SQL_DS_RESTRICT, SQL_DS_CASCADE. |

| SQL_info_type Argument and Description | EDB_ODBC/Advanced Server Returns: |
|---|--|
| SQL_DROP_TABLE: Lists the DROP TABLE clauses supported by the server. | Returns: SQL_DT_DROP_TABLE, SQL_DS_RESTRICT, SQL_DS_CASCADE. |
| SQL_DROP_TRANSLATION: Lists the DROP TRANSLATION clauses supported by the server. | Returns 0. |
| SQL_DROP_VIEW: Lists the DROP VIEW clauses supported by the server. | Returns: SQL_DV_DROP_VIEW, SQL_DS_RESTRICT, SQL_DS_CASCADE. |
| SQL_DYNAMIC_CURSOR_ATTRIBUTES1: Describes the first set of dynamic cursor attributes supported by the driver. | Returns 0 |
| SQL_DYNAMIC_CURSOR_ATTRIBUTES2: Describes the second set of dynamic cursor attributes supported by the driver. | Returns 0 |
| SQL_EXPRESSIONS_IN_ORDERBY: Indicates server support for ORDER BY. | Returns Y. |
| SQL_FETCH_DIRECTION: Indicates FETCH order options (deprecated in ODBC 3.0). | Returns: SQL_FD_FETCH_NEXT, SQL_FD_FETCH_FIRS, SQL_FD_FETCH_LAST, SQL_FD_FETCH_PRIOR, SQL_FD_FETCH_ABSOLUTE, SQL_FD_FETCH_RELATIVE, SQL_FD_FETCH_BOOKMARK. |
| SQL_FILE_USAGE: Indicates how a single-tier driver treats files on the server. | Returns SQL_FILE_NOT_SUPPORTED. The driver is not a single-tier file. |
| SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1: Describes the forward-only cursor attributes supported by the driver. | Returns SQL_CA1_NEXT. |
| SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2: Describes extended attributes for the forward-only cursor designated by SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1. | Returns: SQL_CA2_READ_ONLY_CONCURRENCY, SQL_CA2_CRC_EXACT. |
| SQL_GETDATA_EXTENSIONS: Lists supported extensions to SQLGetData. | Returns: SQL_GD_ANY_COLUMN, SQL_GD_ANY_ORDER, SQL_GD_BLOCK, SQL_GD_BOUND. |
| SQL_GROUP_BY: Indicates the relationship between a GROUP BY clause and columns in the SELECT list. | Returns SQL_GB_GROUP_BY_EQUALS_SELECT. |
| SQL_IDENTIFIER_CASE: Indicates case-sensitivity and case-storage of SQL identifiers. | Returns SQL_IC_LOWER. |
| SQL_INDEX_KEYWORDS: Indicates support for the CREATE INDEX statement. | Returns SQL_IK_NONE. |
| SQL_INFO_SCHEMA_VIEWS: Lists the views supported in the INFORMATION_SCHEMA. | Returns 0. |

| SQL info_type Argument and Description | EDB_ODBC/Advanced Server Returns: |
|---|---|
| SQL_INTEGRITY Prev. SQL_ODBC_SQL_OPT_IEF: Indicates server support for referential integrity syntax checking. | Returns N. |
| SQL_INSERT_STATEMENT: Indicates level of support for the INSERT statement. | Returns: SQL_IS_INSERT_LITERALS, SQL_IS_INSERT_SEARCHED, SQL_IS_SELECT_INTRO. |
| SQL_KEYSET_CURSOR_ATTRIBUTES1: Describes the first set of keyset cursor attributes supported by the driver. | Returns: SQL_CA1_NEXT, SQL_CA1_ABSOLUTE, SQL_CA1_RELATIVE, SQL_CA1_BOOKMARK, SQL_CA1_LOCK_NO_CHANGE, SQL_CA1_POS_POSITION, SQL_CA1_POS_UPDATE, SQL_CA1_POS_DELETE, SQL_CA1_POS_REFRESH, SQL_CA1_BULK_ADD, SQL_CA1_BULK_UPDATE_BY_BOOKMARK, SQL_CA1_BULK_DELETE_BY_BOOKMARK, SQL_CA1_BULK_FETCH_BY_BOOKMARK. |
| SQL_KEYSET_CURSOR_ATTRIBUTES2: Describes the second set of keyset cursor attributes supported by the driver. | Returns: SQL_CA2_READ_ONLY_CONCURRENCY, SQL_CA2_OPT_ROWVER_CONCURRENCY, SQL_CA2_SENSITIVITY_ADDITIONS, SQL_CA2_SENSITIVITY_DELETIONS, SQL_CA2_SENSITIVITY_UPDATES, SQL_CA2_CRC_EXACT. |
| SQL_KEYWORDS: Identifies the server specific reserved keywords. | Returns "". There are no server specific reserved keywords. |
| SQL_LIKE_ESCAPE_CLAUSE: Indicates support for an escape character in LIKE predicates. | Returns N. Advanced Server does not support escape characters in LIKE predicates. |
| SQL_LOCK_TYPES: Lists supported lock types (deprecated in ODBC 3.0). | Returns SQL_LCK_NO_CHANGE. |
| SQL_MAX_ASYNC_CONCURRENT_STATEMENTS: The number of active concurrent statements that the driver can support. | This info_type is currently unsupported. |
| SQL_MAX_BINARY_LITERAL_LEN: The maximum length of a binary literal. | Returns 0. The maximum length is unspecified. |
| SQL_MAX_CATALOG_NAME_LEN: The maximum length of a catalog name on the server. | Returns 0. The maximum length is unspecified. |
| SQL_MAX_QUALIFIER_NAME_LEN: The maximum length of a qualifier. | Returns 0. The maximum length is unspecified. |
| SQL_MAX_CHAR_LITERAL_LEN: The maximum number of characters in a character string. | Returns 0. The maximum length is unspecified. |

| SQL info_type Argument and Description | EDB_ODBC/Advanced Server Returns: |
|--|--|
| SQL_MAX_COLUMN_NAME_LEN: The maximum length of a column name. | Returns 64. Column names cannot exceed 64 characters in length. |
| SQL_MAX_COLUMNS_IN_GROUP_BY: The maximum number of columns allowed in a GROUP BY clause. | Returns 0. The maximum length is unspecified. |
| SQL_MAX_COLUMNS_IN_INDEX: The maximum number of columns allowed in an index. | Returns 0. The maximum length is unspecified. |
| SQL_MAX_COLUMNS_IN_ORDER_BY: The maximum number of columns allowed in an ORDER BY clause. | Returns 0. The maximum length is unspecified. |
| SQL_MAX_COLUMNS_IN_SELECT: The maximum number of columns allowed in a SELECT list. | Returns 0. The maximum length is unspecified. |
| SQL_MAX_COLUMNS_IN_TABLE: The maximum number of columns allowed in a table. | Returns 0. The maximum length is unspecified. |
| SQL_MAX_CONCURRENT_ACTIVITIES prev. SQL_MAX_ACTIVE_STATEMENTS: The maximum number of active SQL statements that the driver can support. | Returns 0. The maximum length is unspecified. |
| SQL_MAX_CURSOR_NAME_LEN: The maximum length of a cursor name. | Returns 32. A cursor name cannot exceed 32 characters in length. |
| SQL_MAX_DRIVER_CONNECTIONS prev. SQL_ACTIVE_CONNECTIONS: The maximum number of active connections the driver can support. | Returns 0. There is no specified limit to the number of connections supported. |
| SQL_MAX_IDENTIFIER_LEN: The maximum identifier length allowed by the server. | Returns 64. Identifiers cannot exceed 64 characters in length. |
| SQL_MAX_INDEX_SIZE: The maximum number of bytes allowed in the (combined) fields of an index. | Returns 0. The maximum size is unspecified. |
| SQL_MAX_OWNER_NAME_LEN Now SQL_MAX_SCHEMA_NAME_LEN: The maximum length of an owner name allowed by the server. | Returns 64. The maximum length of an owner name is 64 characters. |
| SQL_MAX_PROCEDURE_NAME_LEN: The maximum length of a procedure name allowed by the server. | Returns 0. The maximum length is unspecified. |
| SQL_MAX_QUALIFIER_NAME_LEN Now SQL_MAX_CATALOG_NAME_LEN: The maximum length of a qualifier name allowed by the server. | Returns 0. The maximum length of a qualifier is unspecified. |
| SQL_MAX_ROW_SIZE: The maximum length of a row. | Returns 0. The maximum row length is unspecified. |

| SQL info_type Argument and Description | EDB_ODBC/Advanced Server Returns: |
|--|--|
| SQL_MAX_ROW_SIZE_INCLUDES_LONG: Indicates whether the SQL_MAX_ROW_SIZE includes the length of any LONGVARCHAR or LONGVARBINARY columns in the row. | Returns Y. SQL_MAX_ROW_SIZE includes the length of any LONGVARCHAR or LONGVARBINARY columns in the row. |
| SQL_MAX_SCHEMA_NAME_LEN: The maximum length of a schema name allowed by the server. | Returns 64. The maximum length of a schema name is 64 characters. |
| SQL_MAX_STATEMENT_LEN: The maximum length of a SQL statement. | Returns 0. Maximum statement length is limited by available memory. |
| SQL_MAX_TABLE_NAME_LEN: The maximum length of a table name allowed by the server. | Returns 64. The maximum length of a table name is 64 characters. |
| SQL_MAX_TABLES_IN_SELECT: The maximum number of tables allowed in the FROM clause of a SELECT statement. | Returns 0. The maximum number of tables allowed is unspecified. |
| SQL_MAX_USER_NAME_LEN: The maximum length of the user name allowed by the server. | Returns 0. The maximum length of a user name is unspecified. |
| SQL_MULT_RESULT_SETS: Indicates server support for multiple result sets. | Returns Y. Advanced Server supports multiple result sets. |
| SQL_MULTIPLE_ACTIVE_TXN: Indicates if the server supports multiple active transactions. | Returns Y. Advanced Server supports multiple active transactions. |
| SQL_NEED_LONG_DATA_LEN: Indicates if the server needs the length of a LONG data value before receiving the value. | Returns N. Advanced Server does not need the length of a LONG data value before receiving the value. |
| SQL_NON_NULLABLE_COLUMNS: Indicates if the server supports NOT NULL values in columns. | Returns SQL_NNC_NON_NULL. Advanced Server does support NOT NULL values in columns. |
| SQL_NULL_COLLATION: Indicates where NULL values are located in a result set. | Returns SQL_NC_HIGH. The location of NULL values in a data set is determined by the ASC and DESC keywords; NULL values are sorted to the high end of the data set. |
| SQL_NUMERIC_FUNCTIONS: Lists the numeric functions supported by the driver and the server. | Returns: SQL_FN_NUM_ABS, SQL_FN_NUM_ATAN, SQL_FN_NUM_CEILING, SQL_FN_NUM_COS, SQL_FN_NUM_EXP, SQL_FN_NUM_FLOOR, SQL_FN_NUM_LOG, SQL_FN_NUM_MOD, SQL_FN_NUM_SIGN, SQL_FN_NUM_SIN, SQL_FN_NUM_SQRT, SQL_FN_NUM_TAN, SQL_FN_NUM_RAND, SQL_FN_NUM_POWER, SQL_FN_NUM_ROUND. |
| SQL_ODBC_API_CONFORMANCE: Indicates the ODBC 3.0 compliance level | Returns SQL_OAC_LEVEL1. The driver conforms to ODBC Level 1 interface. |
| SQL_ODBC_INTERFACE_CONFORMANCE: Indicates the ODBC interface that the driver adheres to. | Returns SQL_OIC_CORE. |

| SQL info_type Argument and Description | EDB_ODBC/Advanced Server Returns: |
|---|---|
| SQL_ODBC_SAG_CLI_CONFORMANCE: Indicates the SQL Access Group compliance level that the driver adheres to. | Returns SQL_OSCC_NOT_COMPLIANT. The driver is not SAG CLI compliant. |
| SQL_ODBC_SQL_CONFORMANCE: Indicates the SQL grammar level that the driver conforms to. | Returns SQL_OSC_CORE. The driver conforms to the core grammar level. |
| SQL_ODBC_SQL_OPT_IEF Now SQL_INTEGRITY: Indicates server support for referential integrity syntax checking. | Returns N. The server does not support referential integrity syntax checking. |
| SQL_ODBC_VER: The ODBC version supported by the driver manager | Returns 03.52.0000. |
| SQL_OJ_CAPABILITIES: Identifies the outer joins that are supported by the server. | Returns: SQL_OJ_LEFT, SQL_OJ_RIGHT, SQL_OJ_FULL, SQL_OJ_NESTED, SQL_OJ_NOT_ORDERED, SQL_OJ_INNER, SQL_OJ_ALL_COMPARISON_OPS. |
| SQL_OUTER_JOINS: Indicates support for outer joins and the outer join escape sequence. | Returns Y. Outer joins are supported. |
| SQL_OWNER_TERM prev. SQL_SCHEMA_TERM: The term used to describe a schema. | Returns schema. |
| SQL_ORDER_BY_COLUMNS_IN_SELECT: Indicates if the columns in an ORDER BY clause must be included in the SELECT list. | Returns N. Columns in an ORDER BY clause do not have to be in the SELECT list. |
| SQL_OWNER_USAGE prev. SQL_SCHEMA_USAGE: Returns a string that indicates which statements support schema qualifiers. | Returns: SQL_OU_DML_STATEMENTS, SQL_OU_TABLE_DEFINITION, SQL_OU_INDEX_DEFINITION, SQL_OU_PRIVILEGE_DEFINITION. |
| SQL_PARAM_ARRAY_ROW_COUNTS: Indicates if the server will return a single row count or separate row counts for each element in an array when executing a parameterized statement with at least one parameter bound to the array. | Returns SQL_PARC_BATCH, if separate row counts are available for each element in an array. SQL_PARC_NO_BATCH if a single, cumulative row count is available for the entire array. |
| SQL_PARAM_ARRAY_SELECTS: Indicates if the server will return one result set or a separate result set for each element in an array (or if the driver does not allow this feature) when executing a parameterized statement with at least one parameter bound to the array. | Returns SQL_PAS_BATCH. One data set is available for each element in an array. |
| SQL_POS_OPERATION: Lists the options supported by SQLSetPos(). | Returns: SQL_POS_POSITION, SQL_POS_REFRESH, SQL_POS_UPDATE, SQL_POS_DELETE, SQL_POS_ADD. |
| SQL_POSITIONED_STATEMENTS: Lists the supported positioned SQL statements. | Returns: SQL_PS_POSITIONED_DELETE, SQL_PS_POSITIONED_UPDATE, SQL_PS_SELECT_FOR_UPDATE. |

| SQL info_type Argument and Description | EDB_ODBC/Advanced Server Returns: |
|---|--|
| SQL_PROCEDURE_TERM: The term used to describe a procedure. | Returns procedure. |
| SQL_PROCEDURES: Indicates if the server and the driver support SQL procedures and procedure invocation syntax. | Returns Y. The server and driver support procedures and procedure invocation syntax. |
| SQL_QUALIFIER_LOCATION prev. SQL_CATALOG_LOCATION: Indicates the position of the schema name in a qualified table name. | Returns SQL_CL_START. The catalog portion of a qualified table name is at the beginning of the name. |
| SQL_QUALIFIER_NAME prev. SQL_CATALOG_NAME: Indicates server support for catalog names. | Returns Y. The server supports catalog names. |
| SQL_QUALIFIER_NAME_SEPARATOR prev. SQL_CATALOG_NAME_SEPARATOR: Character separating the qualifier name from the adjacent name element. | Returns '!'. The server expects a '.' character between the qualifier and the table name. |
| SQL_QUALIFIER_TERM prev. SQL_CATALOG_TERM: The term used to describe a qualifier. | Returns catalog. |
| SQL_QUALIFIER_USAGE prev. SQL_CATALOG_USAGE: Indicates the SQL statements that may refer to qualifiers. | Returns SQL_CU_DML_STATEMENTS. Catalog names can be used in SELECT, INSERT, UPDATE, DELETE, SELECT FOR UPDATE and positioned UPDATE and DELETE statements. |
| SQL_QUALIFIER_USAGE Now SQL_CATALOG_USAGE: Identifies DML statements that support qualifier names. | Returns SQL_CU_DML_STATEMENTS. Qualifiers can be used in all DML statements (SELECT, INSERT, UPDATE, DELETE, SELECT FOR UPDATE). |
| SQL_QUOTED_IDENTIFIER_CASE: Indicates case sensitivity of quoted identifiers. | Returns SQL_IC_SENSITIVE. Quoted identifiers are case sensitive. |
| SQL_QUALIFIER_NAME_SEPARATOR Now SQL_CATALOG_NAME_SEPARATOR: The character that separates the name qualifier from the name element. | Returns . The '.' character is used as a separator in qualified names. |
| SQL_QUALIFIER_TERM: The term used to describe a qualifier. | Returns catalog |
| SQL_QUALIFIER_LOCATION: The position of the qualifier in a qualified table name. | Returns SQL_CL_START. The qualifier precedes the table name in a qualified table name. |
| SQL_ROW_UPDATES: Indicates if keyset-driven or mixed cursors maintain row versions or values. | Returns Y. Cursors maintain values for all fetched rows and can detect updates to the row values. |
| SQL_SCHEMA_TERM: The term used to describe a schema. | Returns schema |

| SQL info_type Argument and Description | EDB_ODBC/Advanced Server Returns: |
|--|--|
| SQL_SCHEMA_USAGE: Indicates the SQL statements that may refer to schemas. | Returns: SQL_OU_DML_STATEMENTS, SQL_OU_TABLE_DEFINITION, SQL_OU_INDEX_DEFINITION, SQL_OU_PRIVILEGE_DEFINITION. |
| SQL_SCROLL_CONCURRENCY: Indicates the cursor concurrency control options supported by the server. | Returns: SQL_SCCO_READ_ONLY, SQL_SCCO_OPT_ROWVER. |
| SQL_SCROLL_OPTIONS: Indicates the cursor scroll options supported by the server. | Returns: SQL_SO_FORWARD_ONLY, SQL_SO_KEYSET_DRIVEN, SQL_SO_STATIC. |
| SQL_SEARCH_PATTERN_ESCAPE: The escape character that allows use of the wildcard characters % and _ in search patterns. | Returns . The " character is used as an escape character for the '%' and '_' characters in search patterns. |
| SQL_SERVER_NAME: Indicates the name of the host. | The returned value is determined by connection properties. |
| SQL_SPECIAL_CHARACTERS: Indicates any special characters allowed in identifier names. | Returns _. The underscore character is allowed in identifier names. |
| SQL_SQL_CONFORMANCE: Indicates the level of SQL-92 compliance. | Returns SQL_SC_SQL92_ENTRY. The driver is SQL92 Entry level compliant. |
| SQL_SQL92_DATETIME_FUNCTIONS: Lists the datetime functions supported by the server. | Returns: SQL_SDF_CURRENT_DATE, SQL_SDF_CURRENT_TIME, SQL_SDF_CURRENT_TIMESTAMP. |
| SQL_SQL92_FOREIGN_KEY_DELETE_RULE: Indicates the server-enforced rules for using a foreign key in a DELETE statement. | Returns: SQL_SFKD_CASCADE, SQL_SFKD_NO_ACTION, SQL_SFKD_SET_DEFAULT, SQL_SFKD_SET_NULL. |
| SQL_SQL92_FOREIGN_KEY_UPDATE_RULE: Indicates the server-enforced rules for using a foreign key in an UPDATE statement. | Returns: SQL_SFKU_CASCADE, SQL_SFKU_NO_ACTION, SQL_SFKU_SET_DEFAULT, SQL_SFKU_SET_NULL. |
| SQL_SQL92_GRANT: Indicates the supported GRANT statement clauses. | Returns: SQL_SG_DELETE_TABLE, SQL_SG_INSERT_TABLE, SQL_SG_REFERENCES_TABLE, SQL_SG_SELECT_TABLE, SQL_SG_UPDATE_TABLE. |
| SQL_SQL92_NUMERIC_VALUE_FUNCTIONS: Lists the scalar numeric functions supported by the server and driver. | Returns: SQL_SNVF_BIT_LENGTH, SQL_SNVF_CHAR_LENGTH, SQL_SNVF_CHARACTER_LENGTH, SQL_SNVF_EXTRACT, SQL_SNVF_OCTET_LENGTH, SQL_SNVF_POSITION. |

| SQL_info_type Argument and Description | EDB_ODBC/Advanced Server Returns: |
|--|---|
| SQL_SQL92_PREDICATES: Identifies the predicates of a SELECT statement supported by the server. | Returns: SQL_SP_EXISTS, SQL_SP_ISNOTNULL, SQL_SP_ISNULL, SQL_SP_OVERLAPS, SQL_SP_LIKE, SQL_SP_IN, SQL_SP_BETWEEN, SQL_SP_COMPARISON, SQL_SP_QUANTIFIED_COMPARISON. |
| SQL_SQL92_RELATIONAL_JOIN_OPERATORS: Identifies the relational join operators supported by the server. | Returns: SQL_SRJO_CROSS_JOIN, SQL_SRJO_EXCEPT_JOIN, SQL_SRJO_FULL_OUTER_JOIN, SQL_SRJO_INNER_JOIN, SQL_SRJO_INTERSECT_JOIN, SQL_SRJO_LEFT_OUTER_JOIN, SQL_SRJO_NATURAL_JOIN, SQL_SRJO_RIGHT_OUTER_JOIN, SQL_SRJO_UNION_JOIN. |
| SQL_SQL92_REVOKE: Identifies the clauses in a REVOKE statement that are supported by the server. | Returns: SQL_SR_DELETE_TABLE, SQL_SR_INSERT_TABLE, SQL_SR_REFERENCES_TABLE, SQL_SR_SELECT_TABLE, SQL_SR_UPDATE_TABLE. |
| SQL_SQL92_ROW_VALUE_CONSTRUCTOR: Indicates the row value constructor expressions in a SELECT statement that are supported by the server. | Returns: SQL_SRVC_VALUE_EXPRESSION, SQL_SRVC_NULL. |
| SQL_SQL92_STRING_FUNCTIONS: Lists the string scalar functions supported by the server and driver. | Returns: SQL_SSF_CONVERT, SQL_SSF_LOWER, SQL_SSF_UPPER, SQL_SSF_SUBSTRING, SQL_SSF_TRANSLATE, SQL_SSF_TRIM_BOTH, SQL_SSF_TRIM_LEADING, SQL_SSF_TRIM_TRAILING. |
| SQL_SQL92_VALUE_EXPRESSIONS: Indicates the value expressions supported by the server. | Returns: SQL_SVE_CASE, SQL_SVE_CAST, SQL_SVE_COALESCE, SQL_SVE_NULLIF. |
| SQL_STANDARD_CLI_CONFORMANCE: Indicates the CLI standard the driver conforms to. | This info_type is currently unsupported. |
| SQL_STATIC_CURSOR_ATTRIBUTES1: Describes the first set of static cursor attributes supported by the driver. | Returns: SQL_CA1_NEXT, SQL_CA1_ABSOLUTE, SQL_CA1_RELATIVE, SQL_CA1_BOOKMARK, SQL_CA1_LOCK_NO_CHANGE, SQL_CA1_POS_POSITION, SQL_CA1_POS_UPDATE, SQL_CA1_POS_DELETE, SQL_CA1_POS_REFRESH, SQL_CA1_BULK_ADD, SQL_CA1_BULK_UPDATE_BY_BOOKMARK, SQL_CA1_BULK_DELETE_BY_BOOKMARK, SQL_CA1_BULK_FETCH_BY_BOOKMARK. |

| SQL info_type Argument and Description | EDB_ODBC/Advanced Server Returns: |
|--|---|
| SQL_STATIC_CURSOR_ATTRIBUTES2: Describes the second set of static cursor attributes supported by the driver. | Returns: SQL_CA2_READ_ONLY_CONCURRENCY, SQL_CA2_OPT_ROWVER_CONCURRENCY, SQL_CA2_SENSITIVITY_ADDITIONS, SQL_CA2_SENSITIVITY_DELETIONS, SQL_CA2_SENSITIVITY_UPDATES, SQL_CA2_CRC_EXACT. |
| SQL_STATIC_SENSITIVITY: Indicates whether changes made to a static cursor by SQLSetPos() or UPDATE or DELETE statements are detected by the application. | Returns: SQL_SS_ADDITIONS, SQL_SS_DELETIONS, SQL_SS_UPDATES. |
| SQL_STRING_FUNCTIONS: Lists the scalar string functions supported by the server and driver. | Returns: SQL_FN_STR_CONCAT, SQL_FN_STR_LTRIM, SQL_FN_STR_LENGTH, SQL_FN_STR_LOCATE, SQL_FN_STR_LCASE, SQL_FN_STR_RTRIM, SQL_FN_STR_SUBSTRING, SQL_FN_STR_UCASE. |
| SQL_SUBQUERIES: Identifies the subquery predicates to a SELECT statement supported by the server. | Returns: SQL_SQ_COMPARISON, SQL_SQ_EXISTS, SQL_SQ_IN, SQL_SQ_QUANTIFIED. |
| SQL_SYSTEM_FUNCTIONS: Lists the scalar system functions supported by the server and driver. | Returns 0. |
| SQL_TABLE_TERM: The term used to describe a table. | Returns table. |
| SQL_TIMESTAMP_ADD_INTERVALS: Indicates the timestamp intervals supported by the server for the TIMESTAMPADD scalar function. | Returns 0. |
| SQL_TIMESTAMP_DIFF_INTERVALS: Indicates the timestamp intervals supported by the server for the TIMESTAMPDIFF scalar function. | Returns 0 |
| SQL_TIMESTAMP_FUNCTIONS: Indicates the date and time functions supported by the server. | Returns: SQL_FN_TD_NOW, SQL_FN_TD_CURDATE, SQL_FN_TD_CURTIME. |
| SQL_TXN_CAPABLE: Identifies the transaction support offered by the server and driver. | Returns SQL_TC_ALL. Transactions can contain both DML and DDL statements. |
| SQL_TXN_ISOLATION_OPTION: Indicates the transaction isolation level supported by the server. | Returns: SQL_TXN_READ_COMMITTED, SQL_TXN_SERIALIZABLE. |
| SQL_UNION: Indicates server support for the UNION clause. | Returns: SQL_U_UNION, SQL_U_UNION_ALL. |

SQL info_type Argument and Description**EDB_ODBC/Advanced Server Returns:**

SQL_USER_NAME: Identifies the name of the user connected to a database; may be different than the login name.

This value is determined by the connection properties.

SQL_XOPEN_CLI_YEAR: The publication year of the X/Open specification that the driver manager complies with.

This info_type is currently unsupported.

Connection Attributes

You can use the ODBC `SQLGetConnectAttr()` and `SQLSetConnectAttr()` functions to retrieve or set the value of a connection attribute.

SQLGetConnectAttr()

The `SQLGetConnectAttr()` function returns the current value of a connection attribute. The signature is:

```
SQLRETURN SQLGetConnectAttr
(
    SQLHDBC conn_handle, //Input
    SQLINTEGER attribute, //Input
    SQLPOINTER value_pointer, //Output
    SQLINTEGER buffer_length, //Input
    SQLINTEGER * string_length_pointer //Output
);
```

- `conn_handle` The connection handle.
- `attribute` identifies the attribute whose value you wish to retrieve.
- `value_pointer` A pointer to the location in memory that will receive the `attribute` value.
- `buffer_length` If `attribute` is defined by ODBC and `value_pointer` points to a character string or binary buffer, `buffer_length` is the length of `value_pointer`. If `value_pointer` points to a fixed-size value (such as an integer), `buffer_length` is ignored.

If EDB-ODBC defines the attribute, `SQLGetConnectAttr()` sets the `buffer_length` parameter. `buffer_length` can be:

Value type**Meaning**

Character string

The length of the character string

| Value type | Meaning |
|------------------------|---|
| Binary buffer | The result of SQL_LEN_BINARY_ATTR(length) |
| Fixed length data type | SQL_IS_INTEGER or SQL_IS_UINTEGER |
| Any other type | SQL_IS_POINTER |

- `string_length_pointer` A pointer to a `SQLINTEGER` that receives the number of bytes available to return in `value_pointer`. If `value_pointer` is `NULL`, `string_length_pointer` is not returned.

This function returns `SQL_SUCCESS`, `SQL_SUCCESS_WITH_INFO`, `SQL_NO_DATA`, `SQL_ERROR` or `SQL_INVALID_HANDLE`.

The following table lists the connection attributes supported by EDB-ODBC.

| Attribute | Supported? | Notes |
|------------------------------|------------|---|
| SQL_ATTR_ACCESS_MODE | NO | SQL_MODE_READ_WRITE |
| SQL_ATTR_ASYNC_ENABLE | NO | SQL_ASYNC_ENABLE_OFF |
| SQL_ATTR_AUTO_IPD | NO | |
| SQL_ATTR_AUTOCOMMIT | YES | SQL_AUTOCOMMIT, SQL_AUTOCOMMIT_ON, SQL_AUTOCOMMIT_OFF |
| SQL_ATTR_CONNECTION_TIMEOUT | NO | |
| SQL_ATTR_CURRENT_CATALOG | NO | |
| SQL_ATTR_DISCONNECT_BEHAVIOR | NO | |
| SQL_ATTR_ENLIST_IN_DTC | YES | For win32 and with conditional compilation |
| SQL_ATTR_ENLIST_IN_XA | NO | |
| SQL_ATTR_LOGIN_TIMEOUT | NO | SQL_LOGIN_TIMEOUT |
| SQL_ATTR_ODBC_CURSORS | NO | |
| SQL_ATTR_PACKET_SIZE | NO | |
| SQL_ATTR_QUIET_MODE | NO | |
| SQL_ATTR_TRACE | NO | |
| SQL_ATTR_TRACEFILE | NO | |
| SQL_ATTR_TRANSLATE_LIB | NO | |
| SQL_ATTR_TRANSLATE_OPTION | NO | |
| SQL_ATTR_TXN_ISOLATION | YES | SQL_TXN_ISOLATION, SQL_DEFAULT_TXN_ISOLATION |

SQLSetConnectAttr()

You can use the ODBC `SQLSetConnectAttr()` function to set the values of connection attributes. The signature of the function is:

```
SQLRETURN SQLSetConnectAttr
(
    SQLHDBC conn_handle , // Input
    SQLINTEGER attribute , // Input
    SQLPOINTER value_pointer , // Input
    SQLINTEGER string_length , // Input
);
```

`conn_handle`

The connection handle

`attribute`

`attribute` identifies the attribute whose value you wish to set

`value_pointer`

A pointer to the value that the `attribute` will assume.

`string_length`

If `attribute` is defined by ODBC and `value_pointer` points to a binary buffer or character string, `string_length` is the length of `value_pointer`. If `value_pointer` points to a fixed-length value (such as an integer), `string_length` is ignored.

If EDB-ODBC defines the attribute, the application sets the `string_length` parameter. Possible `string_length` values are:

| Value Type | Meaning |
|------------------------|---|
| Character string | The length of the character string or SQL_NTS |
| Binary buffer | The result of SQL_LEN_BINARY_ATTR(length) |
| Fixed length data type | SQL_IS_INTEGER or SQL_IS_UINTEGER |
| Any other type | SQL_IS_POINTER |

`SQLSetConnectAttr()` returns `SQL_SUCCESS`, `SQL_SUCCESS_WITH_INFO`, `SQL_ERROR`, `SQL_STILL_EXECUTING` or `SQL_INVALID_HANDLE`.

You can call `SQLSetConnectAttr()` any time after the connection handle is allocated, until the time that the connection is closed with a call to `SQLFreeHandle()`. All attributes set by the call persist until the call to `SQLFreeHandle()`.

Connection attributes have a specific time frame in which they can be set. Some attributes must be

set before the connection is established, while others can only be set after a connection is established.

The following table lists the connection attributes and the time frame in which they can be set:

| Attribute | Set Before or After establishing a connection? |
|-----------------------------|--|
| SQL_ATTR_ACCESS_MODE | Before or After |
| SQL_ATTR_ASYNC_ENABLE | Before or After |
| SQL_ATTR_AUTO_IPD | Before or After |
| SQL_ATTR_AUTOCOMMIT | Before or After |
| SQL_ATTR_CONNECTION_TIMEOUT | Before or After |
| SQL_ATTR_CURRENT_CATALOG | Before or After |
| SQL_ATTR_ENLIST_IN_DTC | After |
| SQL_ATTR_ENLIST_IN_XA | After |
| SQL_ATTR_LOGIN_TIMEOUT | Before |
| SQL_ATTR_ODBC_CURSORS | Before |
| SQL_ATTR_PACKET_SIZE | Before |
| SQL_ATTR_QUIET_MODE | Before or After |
| SQL_ATTR_TRACE | Before or After |
| SQL_ATTR_TRACEFILE | Before or After |
| SQL_ATTR_TRANSLATE_LIB | After |
| SQL_ATTR_TRANSLATE_OPTION | After |
| SQL_ATTR_TXN_ISOLATION | Before or After |

Environment Attributes

You can use the ODBC `SQLGetEnvAttr()` and `SQLSetEnvAttr()` functions to retrieve or set the value of an environment attribute.

SQLGetEnvAttr()

Use the `SQLGetEnvAttr()` function to find the current value of environment attributes on your system. The signature of the function is:

```
SQLRETURN SQLGetConnectAttr
(
    SQLHDBC env_handle, // Input
    SQLINTEGER attribute, // Input
    SQLPOINTER value_ptr, // Output
    SQLINTEGER buffer_length, // Input
    SQLINTEGER * string_length_pointer // Output
);
```

env_handle

The environment handle.

attribute

attribute identifies the attribute whose value you wish to retrieve.

value_pointer

A pointer to the location in memory that will receive the **attribute** value.

buffer_length

If the attribute is a character string, **buffer_length** is the length of **value_ptr**. If the value of the attribute is not a character string, **buffer_length** is unused.

string_length_pointer

A pointer to a **SQLINTEGER** that receives the number of bytes available to return in **value_pointer**. If **value_pointer** is NULL, **string_length_pointer** is not returned.

This function returns **SQL_SUCCESS**, **SQL_SUCCESS_WITH_INFO**, **SQL_NO_DATA**, **SQL_ERROR** or **SQL_INVALID_HANDLE**.

The following table lists the environment attributes supported by EDB-ODBC.

| Attribute | Supported? | Restrictions? |
|-----------------------------|-------------------------------------|-------------------------------------|
| SQL_ATTR_CONNECTION_POOLING | SQL_CP_ONE_PER_DRIVER or SQL_CP_OFF | Determined by connection properties |
| SQL_ATTR_ODBC_VERSION | (SQL_OV_ODBC3), (SQL_OV_ODBC2) | NONE |
| SQL_ATTR_OUTPUT_NTS | SQL_SUCCESS | NONE |

SQLSetEnvAttr()

You can use the `SQLSetEnvAttr()` function to set the values of environment attributes. The signature of the function is:

```
SQLRETURN SQLSetEnvAttr
(
    SQLHENV  env_handle , //Input
    SQLINTEGER attribute , //Input
    SQLPOINTER value_pointer , //Input
    SQLINTEGER string_length //Input
);
```

- `env_handle` The environment handle.
- `attribute` identifies the attribute whose value you wish to set.
- `value_pointer` A pointer to the value assigned to the `attribute`.
- The value will be a `NULL` terminated character string or a 32 bit integer value depending on the specified `attribute`.
- `string_length` If `value_pointer` is a pointer to a binary buffer or character string, `string_length` is the length of `value_pointer`. If the value being assigned to the attribute is a character, `string_length` is the length of that character string. If `value_pointer` is `NULL`, `string_length` is not returned. If `value_pointer` is an integer, `string_length` is ignored.

`SQLSetEnvAttr()` returns `SQL_SUCCESS`, `SQL_INVALID_HANDLE`, `SQL_ERROR` or `SQL_SUCCESS_WITH_INFO`. The application must call `SQLSetEnvAttr()` before allocating a connection handle; all values applied to environment attributes will persist until `SQLFreeHandle()` is called for the connection. ODBC version 3.x allows you to allocate multiple environment handles simultaneously.

The following table lists the environment attributes you can set with `SQLSetAttr()`.

| Attribute | Value_pointer type | Restrictions? |
|-----------------------|--------------------|--|
| SQL_ATTR_ODBC_VERSION | 32 bit Integer | Set this attribute before the application calls any function that includes an SQLHENV argument. |
| SQL_ATTR_OUTPUT_NTS | 32-bit Integer | Defaults to SQL_TRUE. Calls that set this attribute to SQL_FALSE return SQL_ERROR/SQLSTATEHYC00 (feature not implemented). |

Statement Attributes

You can use the ODBC `SQLGetStmtAttr()` and `SQLSetStmtAttr()` functions to retrieve and set the value of a statement attribute.

SQLGetStmtAttr()

The `SQLGetStmtAttr()` function returns the current value of statement attribute. The signature is:

```
SQLRETURN SQLGetConnectAttr
(
    SQLHDBC stmt_handle, //Input
    SQLINTEGER attribute, //Input
    SQLPOINTER value_ptr, //Output
    SQLINTEGER buffer_length, //Input
    SQLINTEGER * string_length_pointer //Output
);
```

- `stmt_handle` The statement handle
- `attribute` is the attribute value
- `value_pointer` A pointer to the location in memory that will receive the `attribute` value.
- `buffer_length` If the attribute is defined by ODBC, `buffer_length` is the length of `value_pointer` (if `value_pointer` points to a character string or binary buffer). If `value_pointer` points to an integer, `buffer_length` is ignored.

If EDB-ODBC defines the attribute, the application sets the `buffer_length` parameter. `buffer_length` can be:

| Value Type | Meaning |
|------------------------|---|
| Character string | The length of the character string |
| Binary buffer | The result of <code>SQL_LEN_BINARY_ATTR(length)</code> |
| Fixed length data type | <code>SQL_IS_INTEGER</code> or <code>SQL_IS_UINTEGER</code> |
| Any other type | <code>SQL_IS_POINTER</code> |

- `string_length_pointer` A pointer to an `SQLINTEGER` that receives the number of bytes required to hold the requested value. If `value_pointer` is NULL, `string_length_pointer` is not returned.

This function returns `SQL_SUCCESS`, `SQL_SUCCESS_WITH_INFO`, `SQL_ERROR` or `SQL_INVALID_HANDLE`.

| Attribute | Supported? | Restrictions? |
|--------------------------------------|------------|---------------|
| <code>SQL_ATTR_APP_PARAM_DESC</code> | YES | |
| <code>SQL_ATTR_APP_ROW_DESC</code> | YES | |

| Attribute | Supported? | Restrictions? |
|--------------------------------|------------|-------------------------|
| SQL_ATTR_ASYNC_ENABLE | NO | |
| SQL_ATTR_CONCURRENCY | YES | SQL_CONCUR_READ_ONLY |
| SQL_ATTR_CURSOR_SCROLLABLE | YES | |
| SQL_ATTR_CURSOR_TYPE | YES | SQL_CURSOR_FORWARD_ONLY |
| SQL_ATTR_CURSOR_SENSITIVITY | YES | SQL_INSENSITIVE |
| SQL_ATTR_ENABLE_AUTO_IPD | NO | |
| SQL_ATTR_FETCH_BOOKMARK_PTR | YES | |
| SQL_ATTR_IMP_PARAM_DESC | YES | |
| SQL_ATTR_IMP_ROW_DESC | YES | |
| SQL_ATTR_KEYSET_SIZE | NO | |
| SQL_ATTR_MAX_LENGTH | NO | |
| SQL_ATTR_MAX_ROWS | NO | |
| SQL_ATTR_METADATA_ID | YES | |
| SQL_ATTR_NOSCAN | NO | |
| SQL_ATTR_PARAM_BIND_OFFSET_PTR | YES | ODBC V2.0 |
| SQL_ATTR_PARAM_BIND_TYPE | YES | |
| SQL_ATTR_PARAM_OPERATION_PTR | YES | |
| SQL_ATTR_PARAM_STATUS_PTR | YES | |
| SQL_ATTR_PARAMS_PROCESSED_PTR | YES | |
| SQL_ATTR_PARAMSET_SIZE | YES | |
| SQL_ATTR_QUERY_TIMEOUT | NO | |
| SQL_ATTR_RETRIEVE_DATA | NO | |
| SQL_ATTR_ROW_BIND_OFFSET_PTR | YES | |
| SQL_ATTR_ROW_BIND_TYPE | NO | |
| SQL_ATTR_ROW_NUMBER | YES | |
| SQL_ATTR_ROW_OPERATION_PTR | YES | |
| SQL_ATTR_ROW_STATUS_PTR | YES | |
| SQL_ATTR_ROWS_FETCHED_PTR | YES | |
| SQL_ATTR_ROW_ARRAY_SIZE | YES | |
| SQL_ATTR_SIMULATE_CURSOR | NO | |
| SQL_ATTR_USE_BOOKMARKS | YES | |
| SQL_ROWSET_SIZE | YES | |

SQLSetStmtAttr()

You can use the `SQLSetStmtAttr()` function to set the values of environment attributes. The signature is:

```
SQLRETURN SQLSetStmtAttr
(
    SQLHENV  stmt_handle , //Input
    SQLINTEGER attribute , //Input
    SQLPOINTER value_pointer , //Input
    SQLINTEGER string_length //Input
);
```

- `stmt_handle` is the environment handle.
- `attribute` identifies the statement attribute whose value you wish to set.
- `value_pointer` is a pointer to the location in memory that holds the value that will be assigned to the attribute. `value_pointer` can be a pointer to:

- A null-terminated character string
- A binary buffer
- A value defined by the driver
- A value of the type `SQLLEN`, `SQLULEN` or `SQLUSMALLINT`

Value-pointer can also optionally hold one of the following values:

- An ODBC descriptor handle
- A `SQLINTEGER` value
- A `SQLULEN` value
- A signed INTEGER (if attribute is a driver-specific value)
- `string_length` If `attribute` is defined by ODBC and `value_pointer` points to a binary buffer or character string, `string_length` is the length of `value_pointer`. If `value_pointer` points to an integer, `string_length` is ignored. If EDB-ODBC defines the attribute, the application sets the `string_length` parameter. Possible `string_length` values are:

| Value Type | Meaning |
|------------------------|---|
| Character string | The length of the character string or SQL_NTS |
| Binary buffer | The result of SQL_LEN_BINARY_ATTR(length) |
| Fixed length data type | SQL_IS_INTEGER or SQL_IS_INTEGER |

| Value Type | Meaning |
|----------------|----------------|
| Any other type | SQL_IS_POINTER |

Error Handling

Diagnostic information for the ODBC functions mentioned in this guide can be retrieved via the ODBC `SQLGetDiagRec()` function.

SQLGetDiagRec()

The `SQLGetDiagRec()` function returns status and error information from a diagnostic record written by the ODBC functions that retrieve or set attribute values. The signature is:

```
SQLRETURN SQLGetDiagRec
(
    SQLSMALLINT handle_type, // Input
    SQLHANDLE handle, // Input
    SQLSMALLINT record_number, // Input
    SQLCHAR *SQLState_pointer, // Output
    SQLINTEGER *native_error_pointer, // Output
    SQLCHAR *error_text_pointer, // Output
    SQLSMALLINT buffer_length, // Input
    SQLSMALLINT *text_length_pointer // Output
);
```

- `handle_type` The handle type of the `handle` argument. `handle_type` must be one of the following:
 - `SQL_HANDLE_ENV` specifies an environment handle.
 - `SQL_HANDLE_STMT` specifies a statement handle.
 - `SQL_HANDLE_DBC` specifies a connection handle.
 - `SQL_HANDLE_DESC` specifies a descriptor handle.
- `handle` The handle associated with the attribute error message.
- `record_number` The status record that the application is seeking information from (must be greater than or equal to 1).
- `SQLState_pointer` Pointer to a memory buffer that receives the `SQLState` error code from the record.
- `native_error_pointer` Pointer to a buffer that receives the native error message for the data source (contained in the `SQL_DIAG_NATIVE` field).

- `error_text_pointer` Pointer to a memory buffer that receives the error text (contained in the `SQL_DIAG_MESSAGE_TEXT` field)
- `buffer_length` The length of the `error_text` buffer.
- `text_length_pointer` Pointer to the buffer that receives the size (in characters) of the `error_text_pointer` field. If the number of characters in the `error_text_pointer` parameter exceeds the number available (in `buffer_length`), `error_text_pointer` will be truncated.

`SQLGetDiagRec()` returns `SQL_SUCCESS`, `SQL_ERROR`, `SQL_INVALID_HANDLE`, `SQL_SUCCESS_WITH_DATA` or `SQL_NO_DATA`.

Supported ODBC API Functions

The following table lists the ODBC API functions; the right column specifies **Yes** if the API is supported by the EDB-ODBC driver. Use the ODBC `SQLGetFunctions()` function (specifying a function ID of `SQL_API_ODBC3_ALL_FUNCTIONS`) to return a current version of this list.

| ODBC API Function Name | Supported by EDB-ODBC? |
|---------------------------------|------------------------|
| <code>SQLAllocConnect()</code> | Yes |
| <code>SQLAllocEnv()</code> | Yes |
| <code>SQLAllocStmt()</code> | Yes |
| <code>SQLBindCol()</code> | Yes |
| <code>SQLCancel()</code> | Yes |
| <code>SQLColAttributes()</code> | Yes |
| <code>SQLConnect()</code> | Yes |
| <code>SQLDescribeCol()</code> | Yes |
| <code>SQLDisconnect()</code> | Yes |
| <code>SQLError()</code> | Yes |
| <code>SQLExecDirect()</code> | Yes |
| <code>SQLExecute()</code> | Yes |
| <code>SQLFetch()</code> | Yes |
| <code>SQLFreeConnect()</code> | Yes |
| <code>SQLFreeEnv()</code> | Yes |
| <code>SQLFreeStmt()</code> | Yes |
| <code>SQLGetCursorName()</code> | Yes |
| <code>SQLNumResultCols()</code> | Yes |
| <code>SQLPrepare()</code> | Yes |

| ODBC API Function Name | Supported by EDB-ODBC? |
|------------------------|------------------------|
| SQLRowCount() | Yes |
| SQLSetCursorName() | Yes |
| SQLSetParam() | Yes |
| SQLTransact() | Yes |
| SQLColumns() | Yes |
| SQLDriverConnect() | Yes |
| SQLGetConnectOption() | Yes |
| SQLGetData() | Yes |
| SQLGetFunctions() | Yes |
| SQLGetInfo() | Yes |
| SQLGetStmtOption() | Yes |
| SQLGetTypeInfo() | Yes |
| SQLParamData() | Yes |
| SQLPutData() | Yes |
| SQLSetConnectOption() | Yes |
| SQLSetStmtOption() | Yes |
| SQLSpecialColumns() | Yes |
| SQLStatistics() | Yes |
| SQLTables() | Yes |
| SQLBrowseConnect() | No |
| SQLColumnPrivileges() | No |
| SQLDataSources() | Yes |
| SQLDescribeParam() | No |
| SQLExtendedFetch() | Yes |
| SQLForeignKeys() | Yes |
| SQLMoreResults() | Yes |
| SQLNativeSQL() | Yes |
| SQLNumParams() | Yes |
| SQLParamOptions() | Yes |
| SQLPrimaryKeys() | Yes |
| SQLProcedureColumns() | Yes |
| SQLProcedures() | Yes |
| SQLSetPos() | Yes |
| SQLSetScrollOptions() | No |
| SQLTablePrivileges() | Yes |
| SQLDrivers() | Yes |

| ODBC API Function Name | Supported by EDB-ODBC? |
|------------------------|------------------------|
| SQLBindParameter() | Yes |
| SQLAllocHandle() | Yes |
| SQLBindParam() | Yes |
| SQLCloseCursor() | Yes |
| SQLColAttribute() | Yes |
| SQLCopyDesc() | Yes |
| SQLEndTran() | Yes |
| SQLFetchScroll() | Yes |
| SQLFreeHandle() | Yes |
| SQLGetConnectAttr() | Yes |
| SQLGetDescField() | Yes |
| SQLGetDescRec() | Yes |
| SQLGetDiagField() | Yes |
| SQLGetDiagRec() | Yes |
| SQLGetEnvAttr() | Yes |
| SQLGetStmtAttr() | Yes |
| SQLSetConnectAttr() | Yes |
| SQLSetDescField() | Yes |
| SQLSetDescRec() | No |
| SQLSetEnvAttr() | Yes |
| SQLSetStmtAttr() | Yes |
| SQLBulkOperations() | Yes |

Supported Data Types

EDB-ODBC supports the following ODBC data types:

| ODBC Data Type | Corresponding Advanced Server Data Type |
|----------------|---|
| SQL_BIGINT | PG_TYPE_INT8 |
| SQL_BINARY | PG_TYPE_BYTEA |
| SQL_BIT | PG_TYPE_BOOL or PG_TYPE_CHAR |
| SQL_CHAR | PG_TYPE_BPCHAR |
| SQL_TYPE_DATE | PG_TYPE_DATE |
| SQL_DECIMAL | PG_TYPE_NUMERIC |
| SQL_DOUBLE | PG_TYPE_FLOAT8 |
| SQL_FLOAT | PG_TYPE_FLOAT8 |

| ODBC Data Type | Corresponding Advanced Server Data Type |
|--------------------|---|
| SQL_INTEGER | PG_TYPE_INT4 |
| SQL_LONGVARBINARY | PG_TYPE_BYTEA |
| SQL_LONGVARCHAR | PG_TYPE_VARCHAR or PG_TYPE_TEXT |
| SQL_NUMERIC | PG_TYPE_NUMERIC |
| SQL_NUMERIC | PG_TYPE_NUMERIC |
| SQL_REAL | PG_TYPE_FLOAT4 |
| SQL_SMALLINT | PG_TYPE_INT2 |
| SQL_TYPE_TIME | PG_TYPE_TIME |
| SQL_TYPE_TIMESTAMP | PG_TYPE_DATETIME |
| SQL_TINYINT | PG_TYPE_INT2 |
| SQL_VARBINARY | PG_TYPE_BYTEA |
| SQL_VARCHAR | PG_TYPE_VARCHAR |

Thread Safety

EDB-ODBC is thread safe.

7 Scram Compatibility

The EDB ODBC Connector provides SCRAM-SHA-256 support for Advanced Server versions 10 and above. This support is available from EDB ODBC 10.01.0000.01 release onwards.