Contents

---

0 EDB Postgres Advanced Server ODBC Connector Guide

ODBC (Open Database Connectivity) is a programming interface that allows a client application to connect to any database that provides an ODBC driver. EDB-ODBC provides connectivity between EDB Postgres Advanced Server (Advanced Server) and ODBC-compliant applications.

This guide contains installation information for EDB-ODBC as well as information about creating data source definitions for EDB-ODBC. This guide also contains reference information that details the ODBC functionality supported by EDB-ODBC.

---

1 What's New

The following features are added to create Advanced Server ODBC Connector `12.00.0000.01` :

- The EDB ODBC Connector now supports EDB Postgres Advanced Server 12.
- The EDB ODBC Connector is now also supported on Windows Server 2019 platform.
- The project is merged with the upstream community driver version 12.00.0000.00.

---

2 Requirements Overview

Supported Versions

The Advanced Server ODBC Connector is certified with Advanced Server version 9.4 and above.

Supported Platforms

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

64 bit Linux:

- Red Hat Enterprise Linux (x86_64) 6.x and 7.x
- CentOS (x86_64) 6.x and 7.x
- OEL Linux 6.x and 7.x
- PPC-LE 8 running RHEL or CentOS 7.x
- SLES 12.x
- Debian 9.x
- Ubuntu 18.04

The Advanced Server 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
- Windows 7

32-bit Windows:

- Windows 10
- Windows 8
- Windows 7

3.0 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

Before installing the ODBC connector, you must:

Install the `epel-release` package:

- On RHEL or CentOS 7:

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

- On RHEL or CentOS 8:

```
dnf -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 EnterpriseDB repository. For information about requesting credentials, visit:

https://info.enterprisedb.com/rs/069-ALB-3images/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:

- On RHEL or CentOS 7:

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

- On RHEL or CentOS 8:

```
dnf -y install https://yum.enterprisedb.com/edb-repo-rpms/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 EnterpriseDB 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 the ODBC Connector

After saving your changes to the configuration file, you can use the `yum install` command to install the ODBC connector on RHEL or CentOS 7. For example, the following commands install the ODBC connector:

- On RHEL or CentOS 7:

```
yum install edb-odbc
```

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

- On RHEL or CentOS 8:

```
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 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 EnterpriseDB.

Before installing the connector, use the following commands to add EnterpriseDB repository configuration files to your SLES host:

```
zypper addrepo https://zypp.enterprisedb.com/suse/epas12-sles.repo
zypper addrepo https://zypp.enterprisedb.com/suse/epas-sles-tools.repo
zypper addrepo  https://zypp.enterprisedb.com/suse/epas-sles-dependencies.repo
```

Each command creates a repository configuration file in the `/etc/zypp/repos.d directory` . The files are named:

- `Edbas12suse.repo`
- `edbasdependencies.repo`
- `edbastools.repo`

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

```
/etc/zypp/repos.d # zypper refresh
Repository 'SLES12-12-0' is up to date.
Repository 'SLES12-Pool' is up to date.
Repository 'SLES12-Updates' is up to date.
Retrieving repository 'EDB Postgres Advanced Server 12 12 - x86_64'
metadata ----------------------[\]

Authentication required for
'https://zypp.enterprisedb.com/12/suse/suse-12-x86_64'
User Name:
Password:
```

```
Retrieving repository 'EDB Postgres Advanced Server 12 12 - x86_64'
metadata.................................[done]
Building repository 'EDB Postgres Advanced Server 12 12 - x86_64'
cache.......................[done]
All repositories have been refreshed.
...
```

When prompted for a `User Name` and `Password`, provide your connection credentials for the EnterpriseDB repository. If you need credentials, contact EnterpriseDB.

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 -p PackageHub/12/x86_64
SUSEConnect -p sle-sdk/12/x86_64
```

For detailed information about registering a SUSE host, click here.

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

```
zypper install edb-odbc
```

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

Installing a DEB Package 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 EnterpriseDB repository. To request credentials for the repository, click here.

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

1. Assume superuser privileges:

```
sudo su –
```

2. Configure the EnterpriseDB repository:

```
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'
```

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

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

4. Add the EBD 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 EnterpriseDB Connectors Installation wizard to add the ODBC connector to your system; the wizard is available here.

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.

The 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 or through Linux `Start` 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 EnterpriseDB 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 Linu` 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.

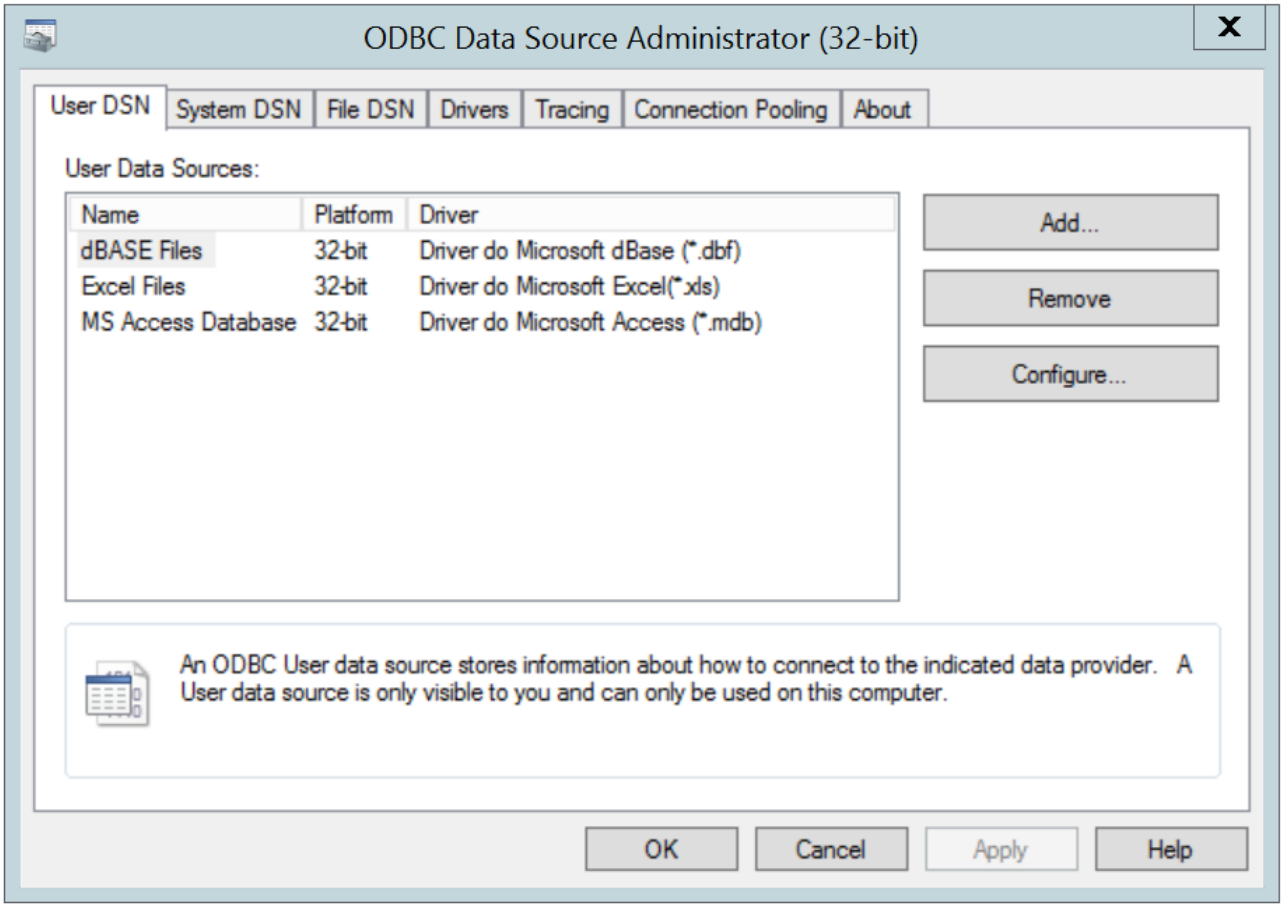
---

5 EDB-ODBC Connection Properties

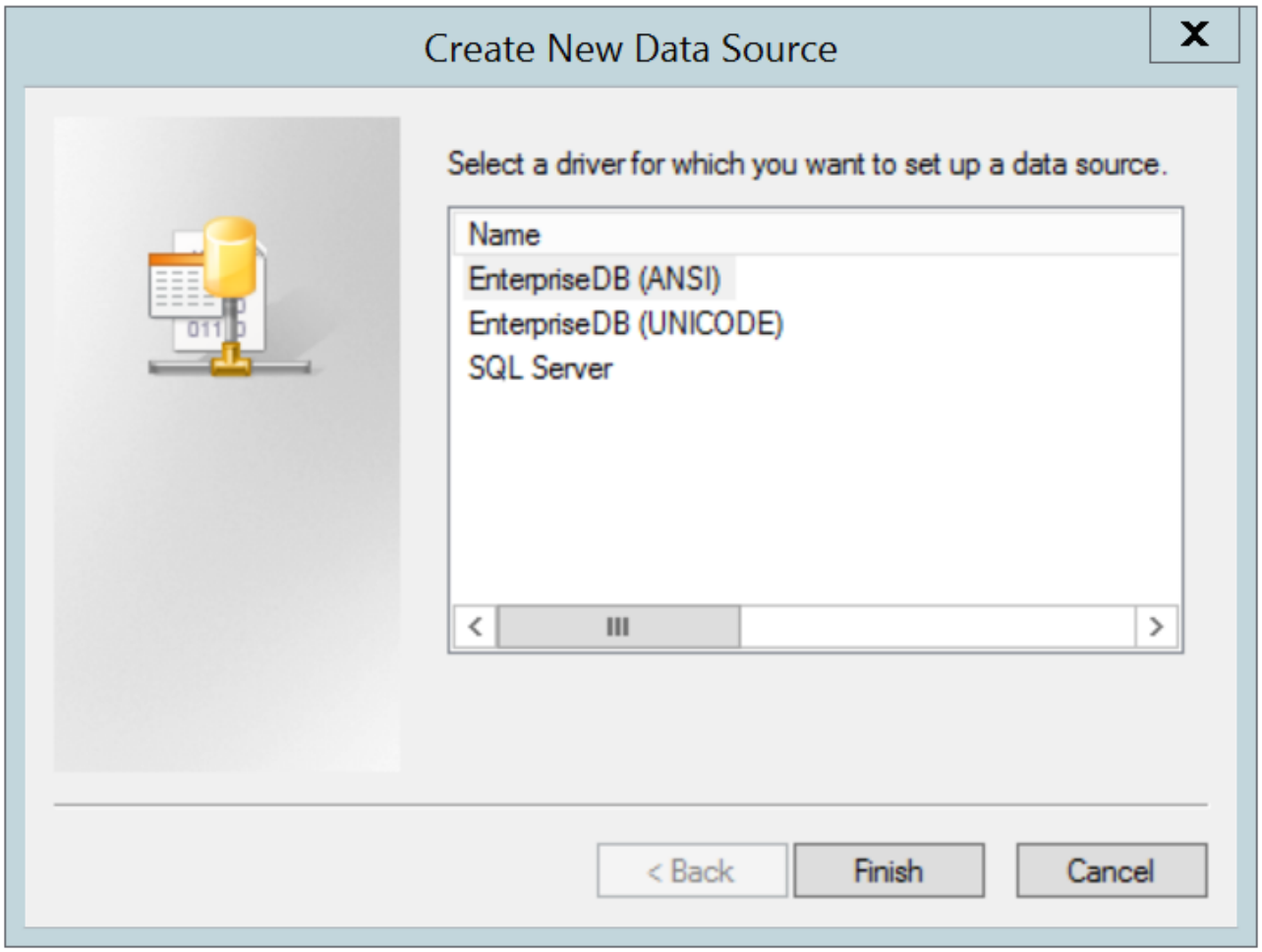
The following table describes the connection properties that you can specify through the dialogs in the graphical connection manager tools, or in the `odbc.ini` file that defines a named data source. The columns identify the connection property (as it appears in the ODBC Administrator dialogs), the corresponding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

Adding a Data Source Definition in Windows

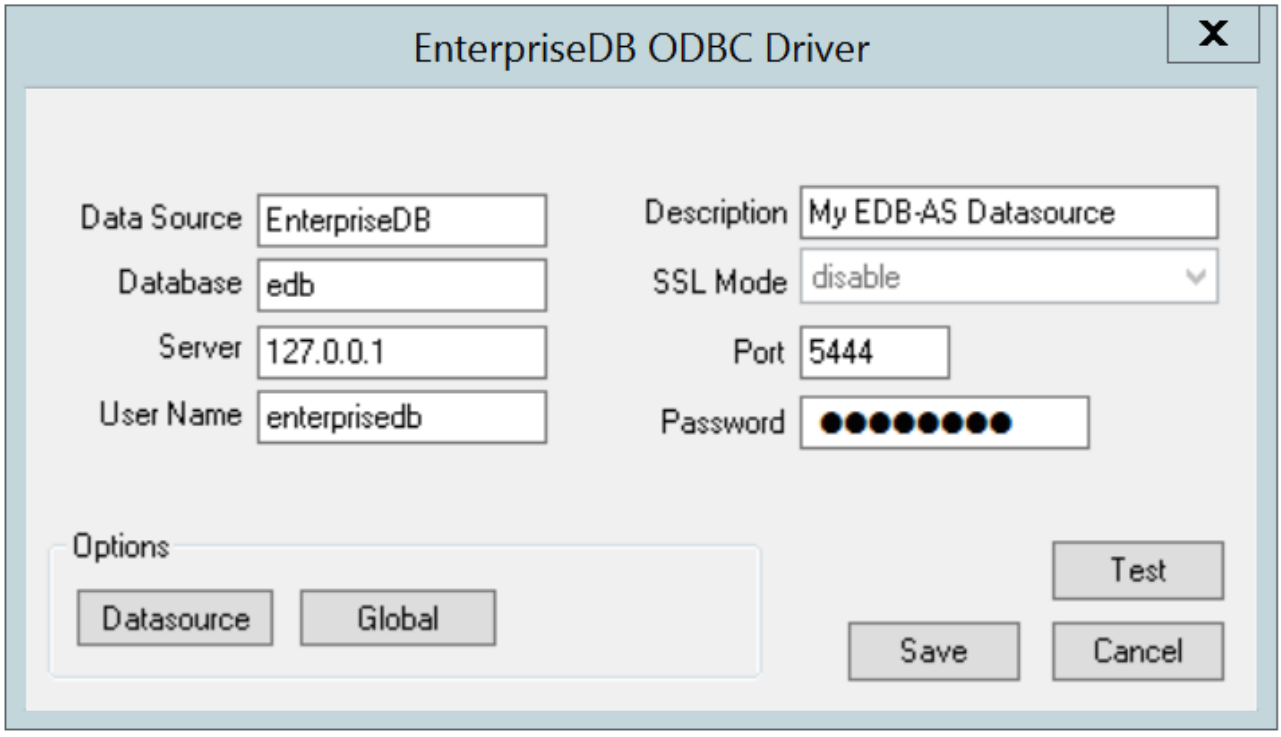
The Windows ODBC `Data Source Administrator` is a graphical interface that creates named data sources. You can open the `ODBC Data Source Administrator` by navigating to the `Control Panel`, opening the `Administrative Tools` menu, and double-clicking the appropriate `ODBC Data Sources` icon (`32- or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



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 EnterpriseDB ODBC Driver dialog opens.



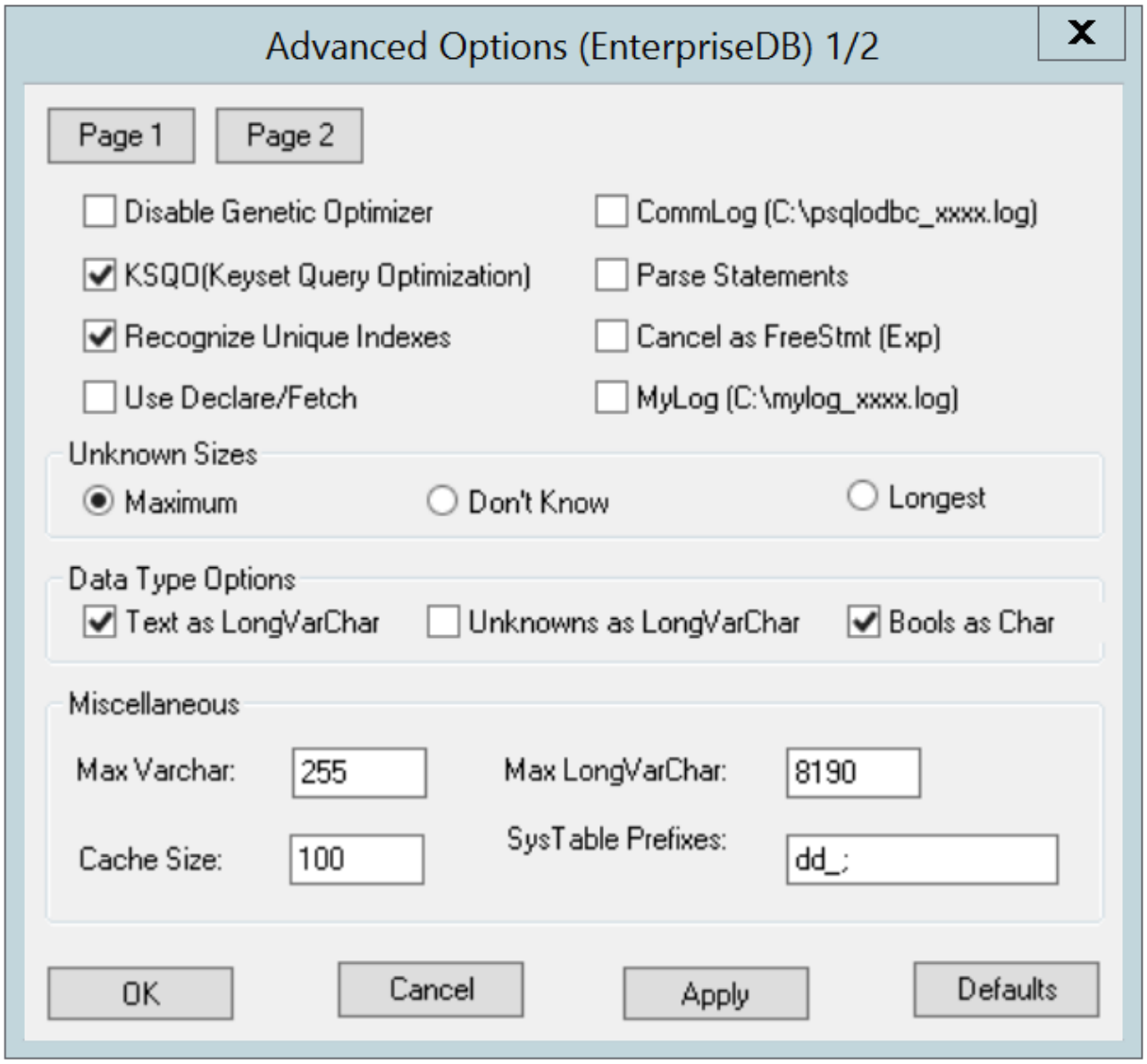
Use the fields on the dialog to define the named data source:

- 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).



- 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 key-set 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.

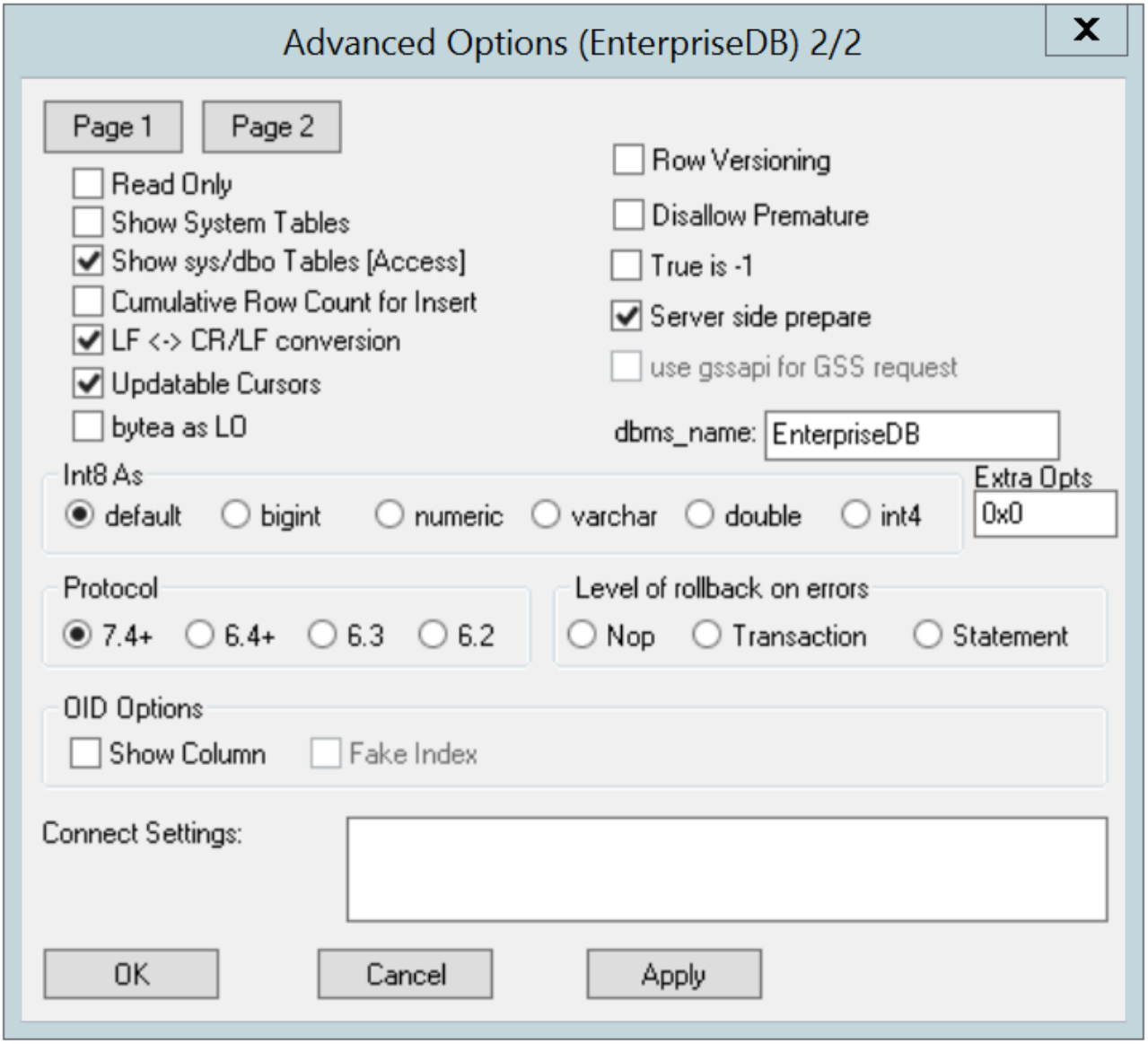
You can specify values for some of the properties associated with the named data source in the fields in the `Miscellaneous` box:

- 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.



- 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 CD
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 difficu
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
0x20	If set, each SQL command is processed in a separate network round-trip, otherwise, SQL commands are groupe

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 ca

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

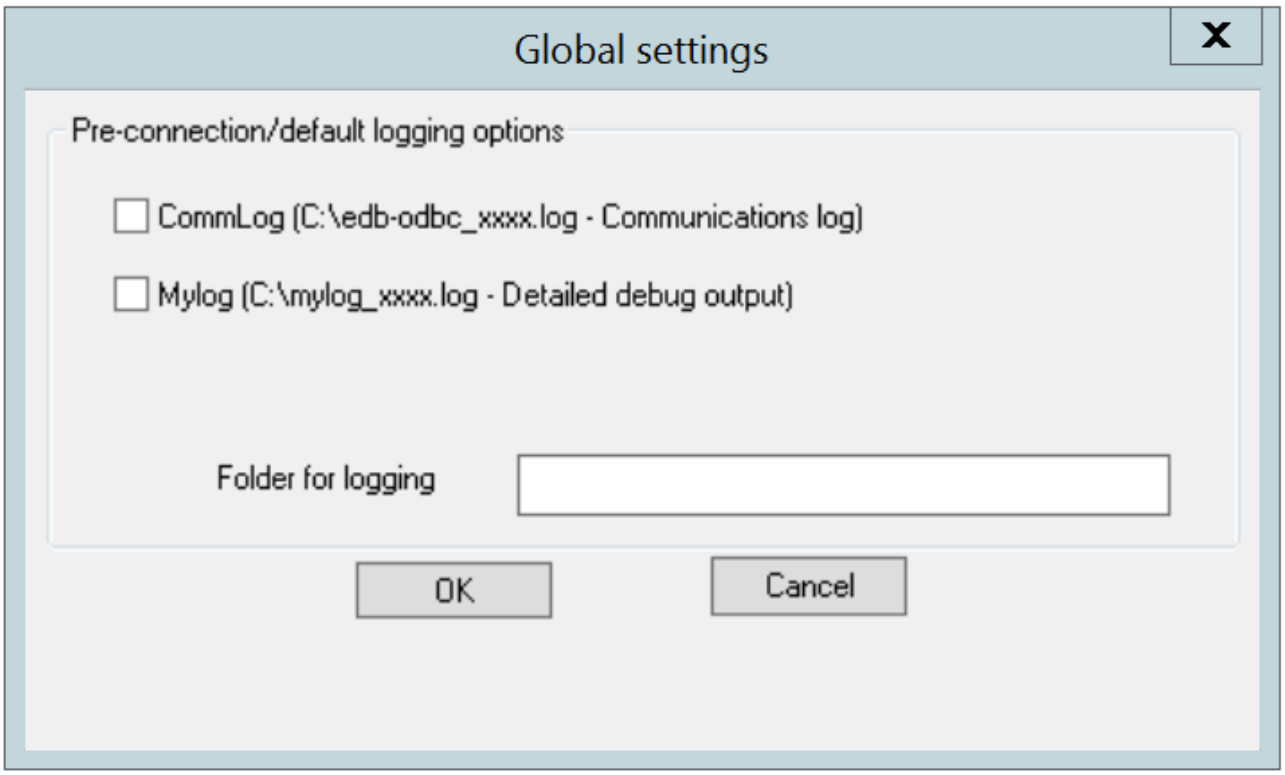
- Check the box next to `Show Column` to include the `OID` column in the result set of the `SQLColumns()` function. If this box is not checked, the `OID` column is hidden from `SQLColumns()` .
- Check the box next to `Fake Columns` to specify that the `SQLStatistics()` function should report that a unique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

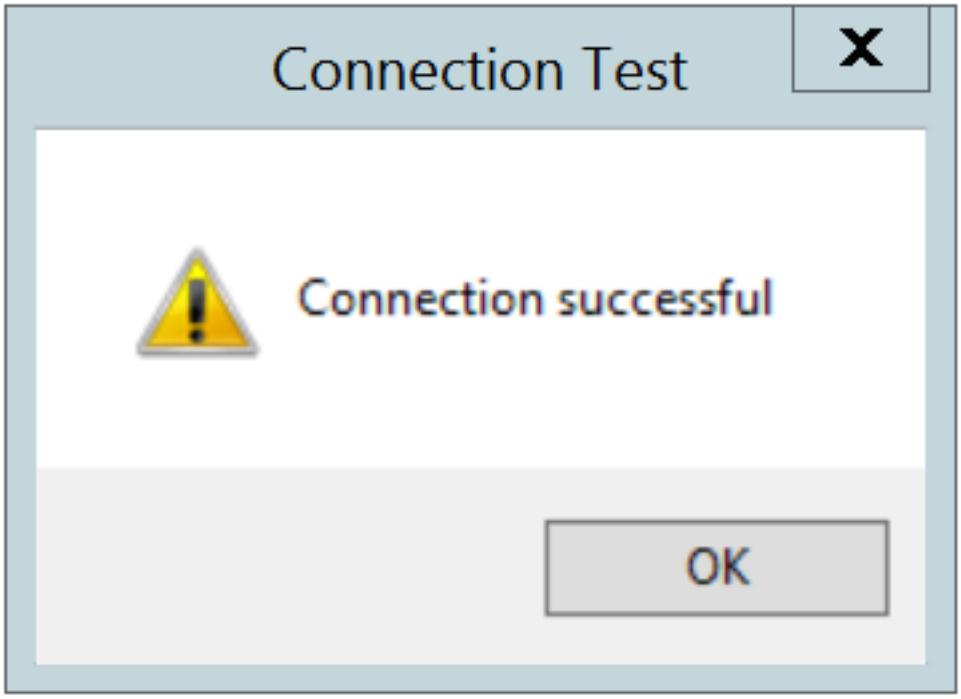
When you've defined the connection properties for the named data source, click the `Apply` button to apply the options; you can optionally exit without saving any options by choosing `Cancel` . Select the `OK` button to save the options and exit.

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.



- 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.



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:

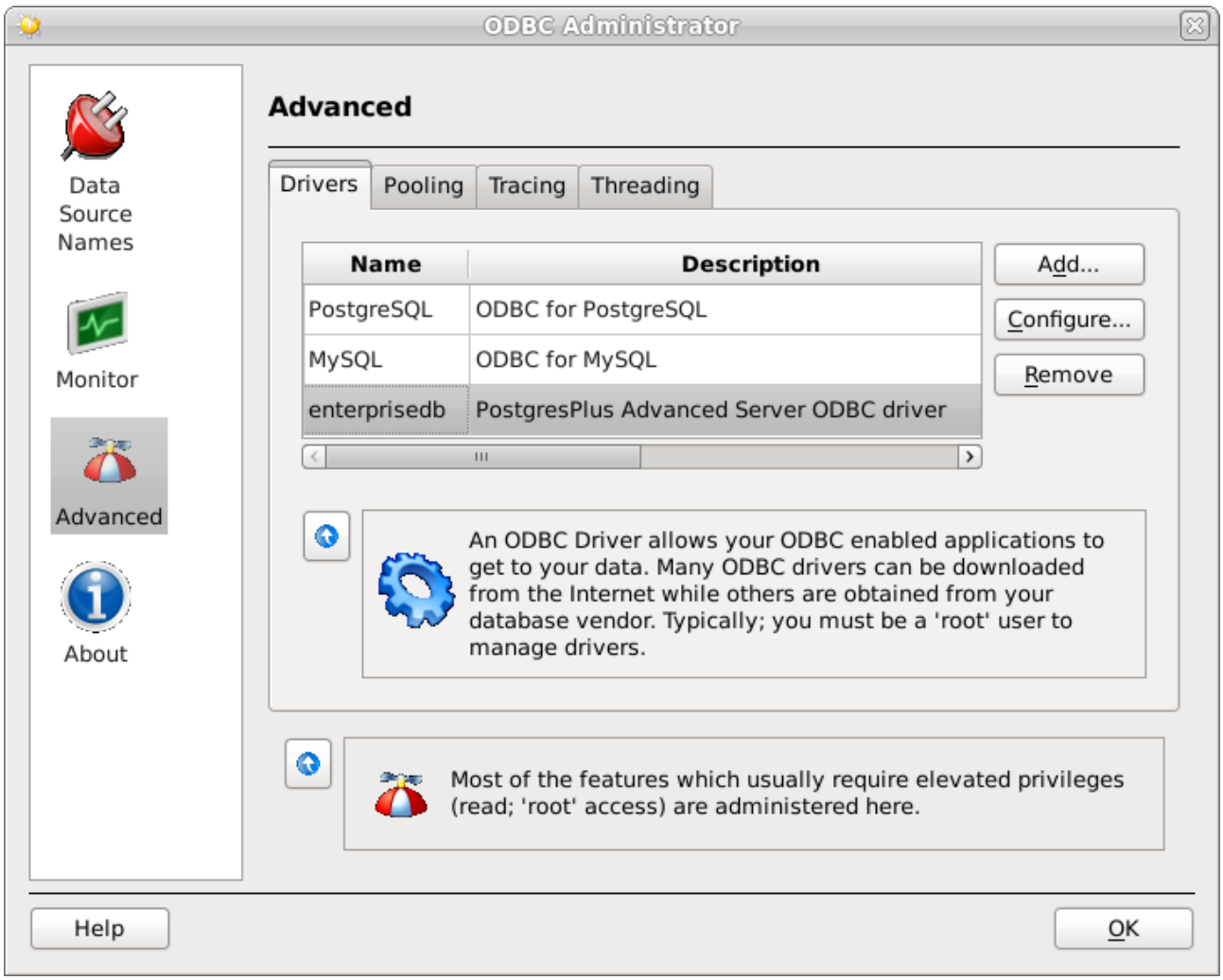
```
yum install unixODBC
```

followed by:

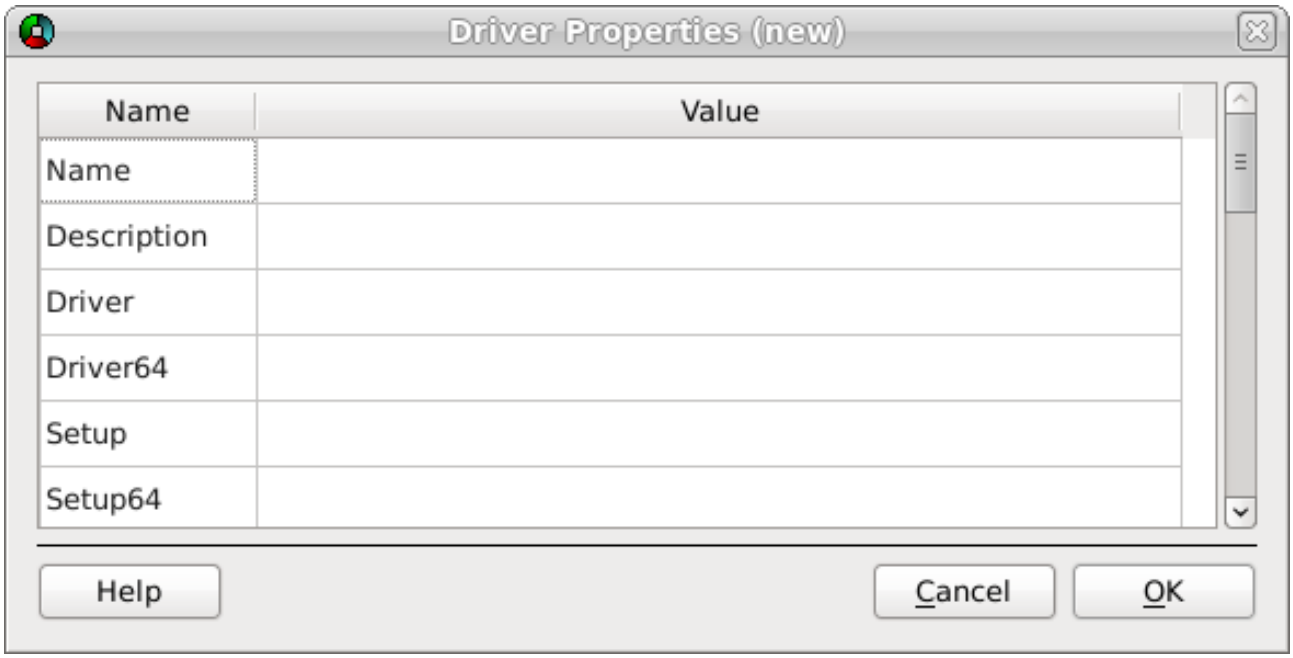
```
yum install unixODBC-kde
```

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

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.



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:



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

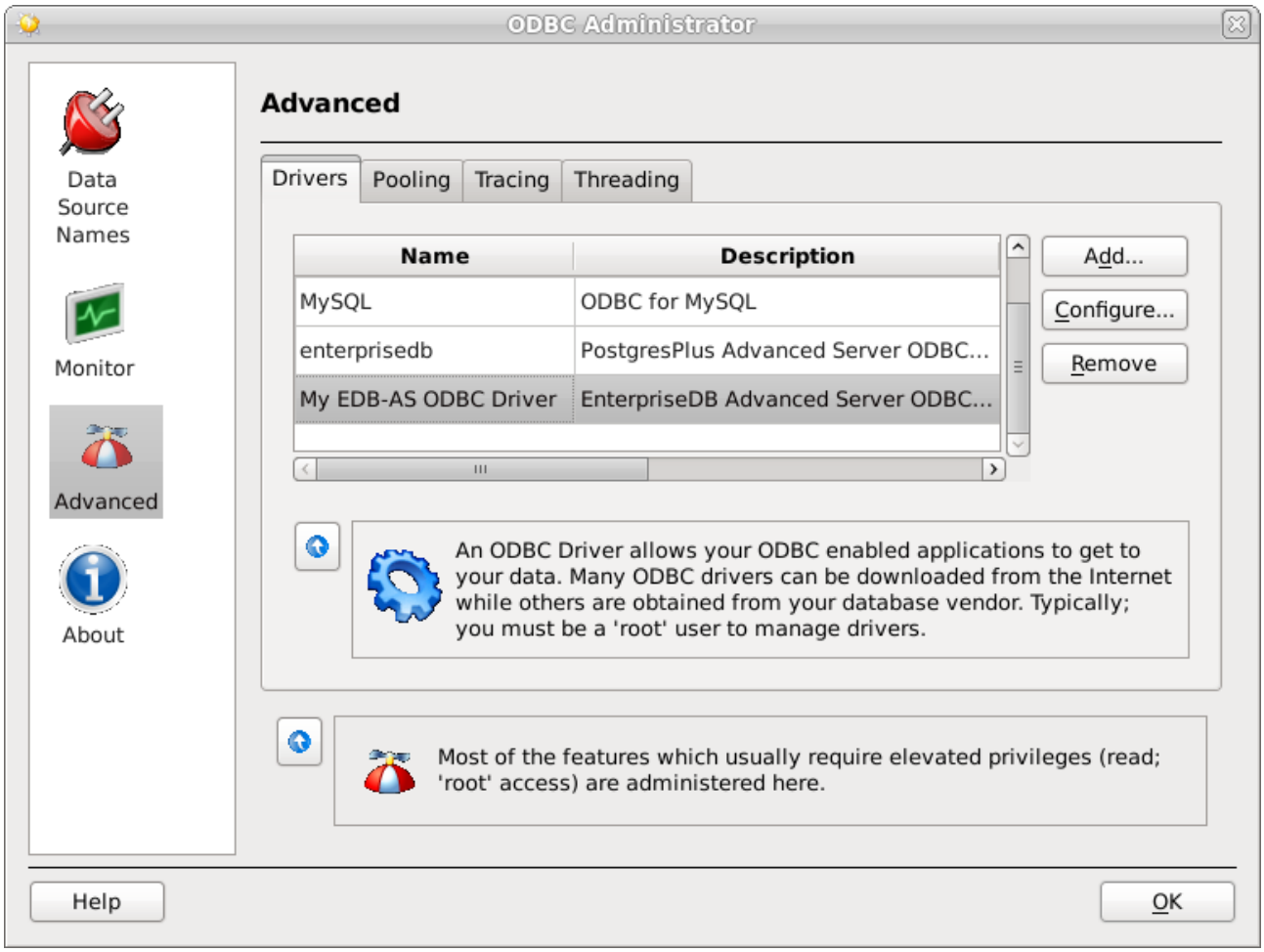
- Add a unique name for the driver to the `Name` field.

- 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/libodbcedbS.so`

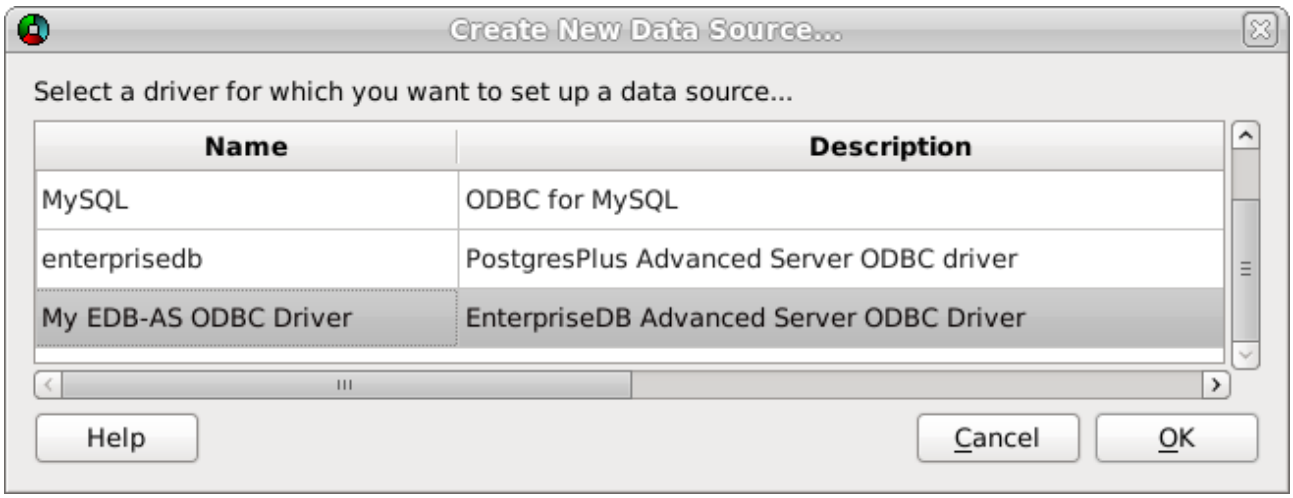
When you've described the driver properties for the EDB-ODBC driver, click `OK`. The ODBC Data Source Administrator window now includes the EDB-ODBC driver in the list of available ODBC drivers.



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:

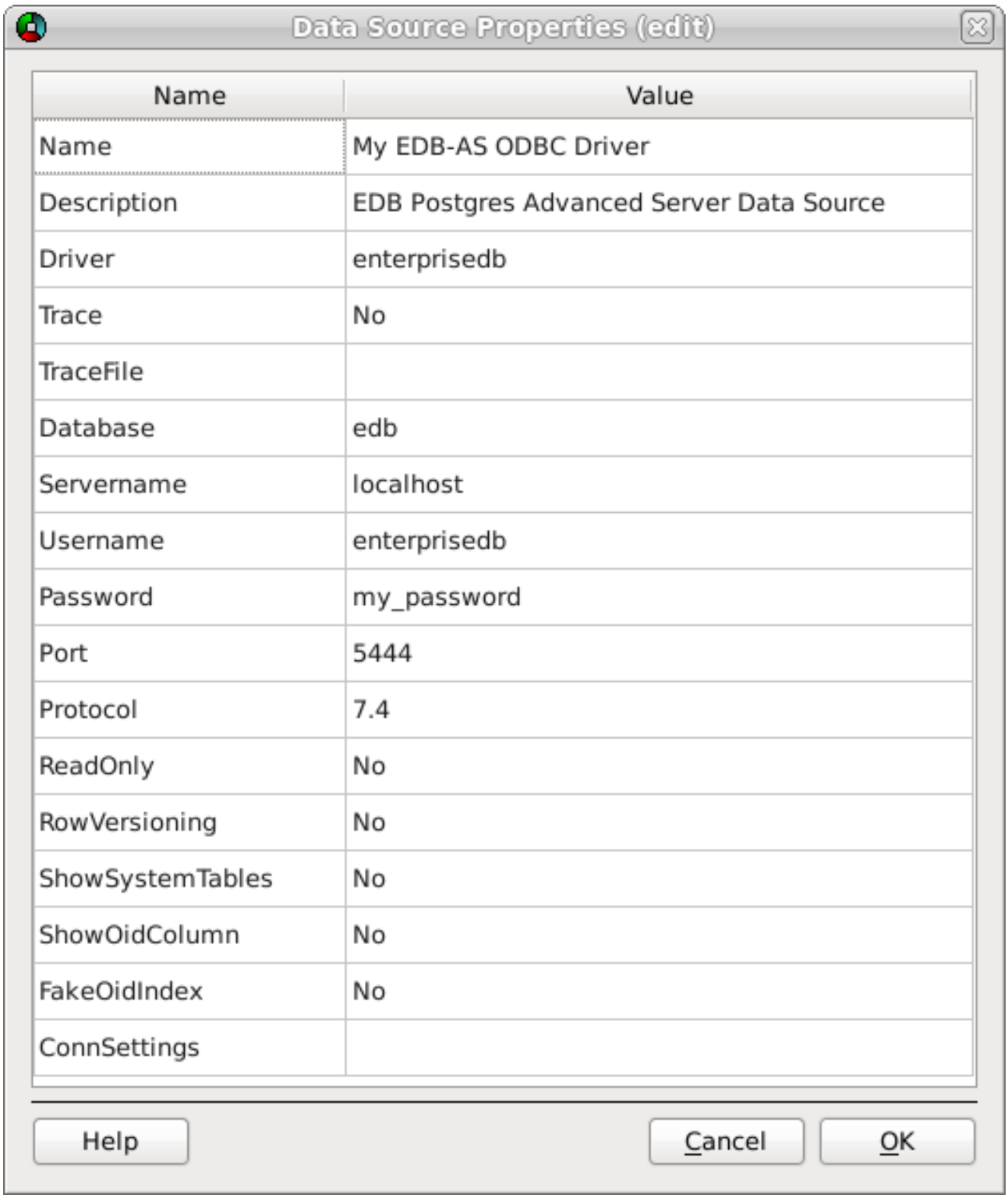
- Choose the `User` tab to add a named data source that is available only to the current user (the data source will be stored in `/user/.odbc.ini`).
- Choose the `System` tab add a named data source that is available to all users. All system data sources are stored in a single file (usually `/etc/odbc.ini`).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

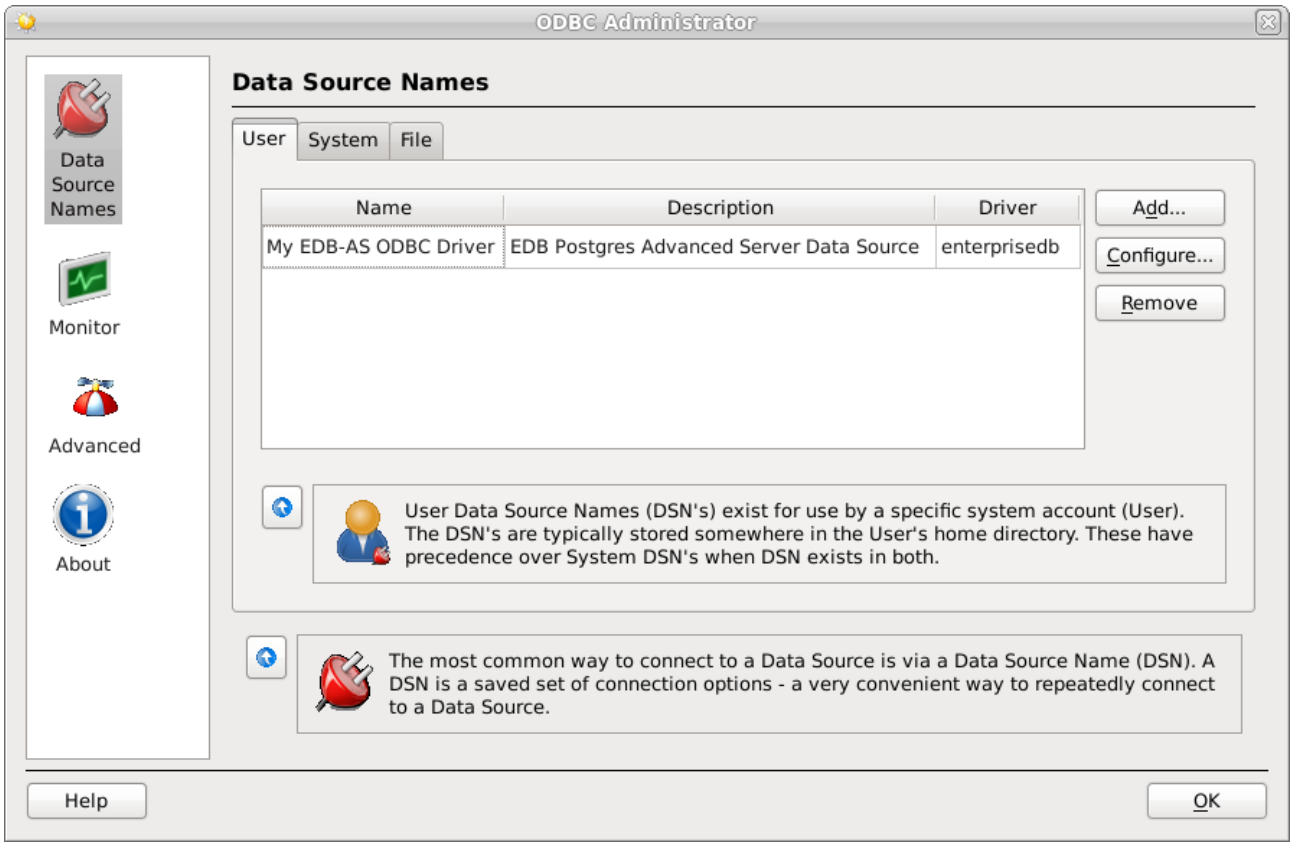


- Enter the data source name in the `Name` field.
- Enter a description of the named data source in the `Description` field.
- The unixODBC driver includes a trace utility that records the sequence of calls made an ODBC application to a log file. Specify `Yes` in the `Trace` field to turn the trace utility 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 `Servername` 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:



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`

`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`

`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 `SQLUINTEGER` bitmask
- a `SQLUINTEGER` flag
- a `SQLUINTEGER` 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
SQL_ACCESSIBLE_PROCEDURES: Indicates if procedures returned by SQLProcedures()can be executed by the applica
SQL_ACCESSIBLE_TABLES: Indicates if the user has SELECT privileges on all table names returned by SQLTables().
SQL_ACTIVE_CONNECTIONS prev. SQL_MAX_DRIVER_CONNECTIONS: Indicates the maximum number of connectio
SQL_ACTIVE_ENVIRONMENTS: The number of active environments EDB-ODBC can support.
SQL_ACTIVE_STATEMENTS prev. SQL_MAX_CONCURRENT_ACTIVITIES: Indicates the maximum number of active st
SQL_AGGREGATE_FUNCTION: Identifies the aggregate functions supported by the server and driver.
SQL_ALTER_DOMAIN: Identifies the ALTER DOMAIN clauses supported by the server.
SQL_ALTER_TABLE: Identifies the ALTER TABLE clauses supported by the server.
SQL_ASYNC_MODE: Level of Asynchronous Mode Supported by EDB-ODBC.
SQL_BATCH_ROW_COUNT: Indicates how the driver returns row counts.
SQL_BATCH_SUPPORT: Indicates support for batch statement execution.
SQL_BOOKMARK_PERSISTENCE: Indicates level of support for bookmarks.
SQL_CATALOG_LOCATION Now SQL_QUALIFIER_LOCATION: Indicates the position of the catalog in a qualified table n
SQL_CATALOG_NAME Now SQL_QUALIFIER_NAME: Indicates support for catalog names.
SQL_CATALOG_NAME_SEPARATOR Now SQL_QUALIFIER_NAME_SEPARATOR: Character separating the catalog na
SQL_CATALOG_TERM Now SQL_QUALIFIER_TERM: The term used to describe a catalog.
SQL_CATALOG_USAGE Now SQL_QUALIFIER_USAGE: Indicates the SQL statements that may refer to catalogs.
SQL_COLLATION_SEQ: Returns the name of the Collation Sequence.
SQL_COLUMN_ALIAS: Indicates server support for column aliases.
SQL_CONCAT_NULL_BEHAVIOR: Indicates how the server handles concatenation of NULL values.
SQL_CONVERT_BIGINT: Indicates conversion support from the BIGINT type using the CONVERT function.
SQL_CONVERT_BINARY: Indicates conversion support from the BINARY type using the CONVERT function.
SQL_CONVERT_BIT: Indicates conversion support from the BIT type using the CONVERT function.
SQL_CONVERT_CHAR: Indicates conversion support from the CHAR type using the CONVERT function.
SQL_CONVERT_DATE: Indicates conversion support from the DATE type using the CONVERT function.
SQL_CONVERT_DECIMAL: Indicates conversion support from the DECIMAL type using the CONVERT function.
SQL_CONVERT_DOUBLE: Indicates conversion support from the DOUBLE type using the CONVERT function.
SQL_CONVERT_FLOAT: Indicates conversion support from the FLOAT type using the CONVERT function.
SQL_CONVERT_FUNCTIONS: Lists the scalar conversion functions supported by the server and driver using the CONVE
SQL_CONVERT_INTEGER: Lists the conversion support from the INTEGER type using the CONVERT function.
SQL_CONVERT_INTERVAL_DAY_TIME: Indicates conversion support from the INTERVAL_DAY_TIME type using the CO
SQL_CONVERT_INTERVAL_YEAR_MONTH: Indicates conversion support from the INTERVAL_YEAR_MONTH type usi
SQL_CONVERT_LONGVARBINARY: Indicates conversion support for the LONG_VARBINARY type using the CONVERT
SQL_CONVERT_LONGVARCHAR: Indicates conversion support for the LONGVARCHAR type using the CONVERT funct
SQL_CONVERT_NUMERIC: Indicates conversion support for the NUMERIC type using the CONVERT function.
SQL_CONVERT_REAL: Indicates conversion support for the REAL type using the CONVERT function
SQL_CONVERT_SMALLINT: Indicates conversion support for the SMALLINT type using the CONVERT function.
SQL_CONVERT_TIME: Indicates conversion support for TIME type using the CONVERT function.
SQL_CVT_TIMESTAMP: Indicates conversion support for TIMESTAMP type using the CONVERT function.
SQL_CONVERT_TINYINT: Indicates conversion support for the TINYINT type using the CONVERT function.
SQL_CONVERT_VARBINARY: Indicates conversion support for the VARBINARY type using the CONVERT function.

SQL info_type Argument and Description

SQL_CONVERT_VARCHAR: Indicates conversion support for VARCHAR type using the CONVERT function.
SQL_CONVERT_WCHAR: Indicates conversion support for the WCHAR type using the CONVERT function.
SQL_CONVERT_WLONGVARCHAR: Indicates conversion support for the WLONGVARCHAR type using the CONVERT f
SQL_CONVERT_WVARCHAR: Indicates conversion support for the WVARCHAR type using the CONVERT function.
SQL_CORRELATION_NAME: Indicates server support for correlation names.
SQL_CREATE_ASSERTION: Indicates support for the CREATE ASSERTION statement.
SQL_CREATE_CHARACTER_SET: Indicates support for CREATE CHARACTER statement.
SQL_CREATE_COLLATION: Indicates support for the CREATE COLLATION.
SQL_CREATE_DOMAIN: Indicates support for the CREATE DOMAIN statement.
SQL_CREATE_SCHEMA: Indicates support for the CREATE SCHEMA statement.
SQL_CREATE_TABLE: Indicates support for the CREATE TABLE statement.
SQL_CREATE_TRANSLATION: Indicates support for the CREATE TRANSLATION statement.
SQL_CREATE_VIEW: Indicates support for the CREATE VIEW statement.
SQL_CURSOR_COMMIT_BEHAVIOR: Indicates how a COMMIT operation affects the cursor.
SQL_CURSOR_ROLLBACK_BEHAVIOR: Indicates the server behavior after a ROLLBACK operation.
SQL_CURSOR_SENSITIVITY:Indicates how the server synchronizes changes to a result set.
SQL_DATA_SOURCE_NAME: Returns the server name used during connection.
SQL_DATA_SOURCE_READ_ONLY: Indicates if the connection is in READ ONLY mode.
SQL_DATABASE_NAME: Returns the name of the database.
SQL_DATETIME_LITERALS: Indicates the DATETIME LITERALS supported by the server.
SQL_DBMS_NAME: Returns the name of the DBMS system.
SQL_DBMS_VER: Returns the server version.
SQL_DDL_INDEX: Indicates support for creating and dropping indexes.
SQL_DEFAULT_TXN_ISOLATION: Indicates support for transaction isolation by the server.
SQL_DESCRIBE_PARAMETER: Indicates support for the DESCRIBE INPUT statement.
SQL_DM_VER: The version of the Driver Manager.
SQL_DRIVER_HDBC: The Driver's connection handle.
SQL_DRIVER_HDESC: The Driver descriptor handle.
SQL_DRIVER_HENV: The Driver's environment handle.
SQL_DRIVER_HLIB: The Driver handle.
SQL_DRIVER_HSTMT: The Driver's statement handle.
SQL_DRIVER_NAME: The name of the driver.
SQL_DRIVER_ODBC_VER: Identifies the ODBC version that the driver supports.
SQL_DRIVER_VER: Identifies the driver version.
SQL_DROP_ASSERTION: Lists the DROP ASSERTION clauses supported by the server.
SQL_DROP_CHARACTER_SET: Lists the DROP CHARACTER clauses supported by the server.
SQL_DROP_COLLATION: Lists the DROP COLLATION clauses supported by the server.
SQL_DROP_DOMAIN: Lists the DROP DOMAIN clauses supported by the server.
SQL_DROP_SCHEMA: Lists the DROP SCHEMA clauses supported by the server.
SQL_DROP_TABLE: Lists the DROP TABLE clauses supported by the server.
SQL_DROP_TRANSLATION: Lists the DROP TRANSLATION clauses supported by the server.
SQL_DROP_VIEW: Lists the DROP VIEW clauses supported by the server.
SQL_DYNAMIC_CURSOR_ATTRIBUTES1: Describes the first set of dynamic cursor attributes supported by the driver.
SQL_DYNAMIC_CURSOR_ATTRIBUTES2: Describes the second set of dynamic cursor attributes supported by the driver
SQL_EXPRESSIONS_IN_ORDERBY: Indicates server support for ORDER BY.
SQL_FETCH_DIRECTION: Indicates FETCH order options (deprecated in ODBC 3.0).
SQL_FILE_USAGE: Indicates how a single-tier driver treats files on the server.
SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1: Describes the forward-only cursor attributes supported by the driver.
SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2: Describes extended attributes for the forward-only cursor designated
SQL_GETDATA_EXTENSIONS: Lists supported extensions to SQLGetData.
SQL_GROUP_BY: Indicates the relationship between a GROUP BY clause and columns in the SELECT list.
SQL_IDENTIFIER_CASE: Indicates case-sensitivity and case-storage of SQL identifiers.
SQL_INDEX_KEYWORDS: Indicates support for the CREATE INDEX statement.
SQL_INFO_SCHEMA_VIEWS: Lists the views supported in the INFORMATION_SCHEMA.
SQL_INTEGRITY Prev. SQL_ODBC_SQL_OPT_IEF: Indicates server support for referential integrity syntax checking.
SQL_INSERT_STATEMENT: Indicates level of support for the INSERT statement.
SQL_KEYSET_CURSOR_ATTRIBUTES1: Describes the first set of keyset cursor attributes supported by the driver.
SQL_KEYSET_CURSOR_ATTRIBUTES2: Describes the second set of keyset cursor attributes supported by the driver.
SQL_KEYWORDS: Identifies the server specific reserved keywords.

SQL info_type Argument and Description

SQL_LIKE_ESCAPE_CLAUSE: Indicates support for an escape character in LIKE predicates.
SQL_LOCK_TYPES: Lists supported lock types (deprecated in ODBC 3.0).
SQL_MAX_ASYNC_CONCURRENT_STATEMENTS: The number of active concurrent statements that the driver can supp
SQL_MAX_BINARY_LITERAL_LEN: The maximum length of a binary literal.
SQL_MAX_CATALOG_NAME_LEN: The maximum length of a catalog name on the server.
SQL_MAX_QUALIFIER_NAME_LEN: The maximum length of a qualifier.
SQL_MAX_CHAR_LITERAL_LEN: The maximum number of characters in a character string.
SQL_MAX_COLUMN_NAME_LEN: The maximum length of a column name.
SQL_MAX_COLUMNS_IN_GROUP_BY: The maximum number of columns allowed in a GROUP BY clause.
SQL_MAX_COLUMNS_IN_INDEX: The maximum number of columns allowed in an index.
SQL_MAX_COLUMNS_IN_ORDER_BY: The maximum number of columns allowed in an ORDER BY clause.
SQL_MAX_COLUMNS_IN_SELECT: The maximum number of columns allowed in a SELECT list.
SQL_MAX_COLUMNS_IN_TABLE: The maximum number of columns allowed in a table.
SQL_MAX_CONCURRENT_ACTIVITIES prev. SQL_MAX_ACTIVE_STATEMENTS: The maximum number of active SQL
SQL_MAX_CURSOR_NAME_LEN: The maximum length of a cursor name.
SQL_MAX_DRIVER_CONNECTIONS prev. SQL_ACTIVE_CONNECTIONS: The maximum number of active connections
SQL_MAX_IDENTIFIER_LEN: The maximum identifier length allowed by the server.
SQL_MAX_INDEX_SIZE: The maximum number of bytes allowed in the (combined) fields of an index.
SQL_MAX_OWNER_NAME_LEN Now SQL_MAX_SCHEMA_NAME_LEN: The maximum length of an owner name allowe
SQL_MAX_PROCEDURE_NAME_LEN: The maximum length of a procedure name allowed by the server.
SQL_MAX_QUALIFIER_NAME_LEN Now SQL_MAX_CATALOG_NAME_LEN: The maximum length of a qualifier name a
SQL_MAX_ROW_SIZE: The maximum length of a row.
SQL_MAX_ROW_SIZE_INCLUDES_LONG: Indicates whether the SQL_MAX_ROW_SIZE includes the length of any LON
SQL_MAX_SCHEMA_NAME_LEN: The maximum length of a schema name allowed by the server.
SQL_MAX_STATEMENT_LEN: The maximum length of a SQL statement.
SQL_MAX_TABLE_NAME_LEN: The maximum length of a table name allowed by the server.
SQL_MAX_TABLES_IN_SELECT: The maximum number of tables allowed in the FROM clause of a SELECT statement.
SQL_MAX_USER_NAME_LEN: The maximum length of the user name allowed by the server.
SQL_MULT_RESULT_SETS: Indicates server support for multiple result sets.
SQL_MULTIPLE_ACTIVE_TXN: Indicates if the 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.
SQL_NON_NULLABLE_COLUMNS: Indicates if the server supports NOT NULL values in columns.
SQL_NULL_COLLATION: Indicates where NULL values are located in a result set.
SQL_NUMERIC_FUNCTIONS: Lists the numeric functions supported by the driver and the server.
SQL_ODBC_API_CONFORMANCE: Indicates the ODBC 3.0 compliance level
SQL_ODBC_INTERFACE_CONFORMANCE: Indicates the ODBC interface that the driver adheres to.
SQL_ODBC_SAG_CLI_CONFORMANCE: Indicates the SQL Access Group compliance level that the driver adheres to.
SQL_ODBC_SQL_CONFORMANCE: Indicates the SQL grammar level that the driver conforms to.
SQL_ODBC_SQL_OPT_IEF Now SQL_INTEGRITY: Indicates server support for referential integrity syntax checking.
SQL_ODBC_VER: The ODBC version supported by the driver manager
SQL_OJ_CAPABILITIES: Identifies the outer joins that are supported by the server.
SQL_OUTER_JOINS: Indicates support for outer joins and the outer join escape sequence.
SQL_OWNER_TERM prev. SQL_SCHEMA_TERM: The term used to describe a schema.
SQL_ORDER_BY_COLUMNS_IN_SELECT: Indicates if the columns in an ORDER BY clause must be included in the SEL
SQL_OWNER_USAGE prev. SQL_SCHEMA_USAGE: Returns a string that indicates which statements support schema q
SQL_PARAM_ARRAY_ROW_COUNTS: Indicates if the server will return a single row count or separate row counts for ea
SQL_PARAM_ARRAY_SELECTS: Indicates if the server will return one result set or a separate result set for each element
SQL_POS_OPERATION: Lists the options supported by SQLSetPos().
SQL_POSITIONED_STATEMENTS: Lists the supported positioned SQL statements.
SQL_PROCEDURE_TERM: The term used to describe a procedure.
SQL_PROCEDURES: Indicates if the server and the driver support SQL procedures and procedure invocation syntax.
SQL_QUALIFIER_LOCATION prev. SQL_CATALOG_LOCATION: Indicates the position of the schema name in a qualified
SQL_QUALIFIER_NAME prev. SQL_CATALOG_NAME: Indicates server support for catalog names.
SQL_QUALIFIER_NAME_SEPARATOR prev. SQL_CATALOG_NAME_SEPARATOR: Character separating the qualifier n
SQL_QUALIFIER_TERM prev. SQL_CATALOG_TERM: The term used to describe a qualifier.
SQL_QUALIFIER_USAGE prev. SQL_CATALOG_USAGE: Indicates the SQL statements that may refer to qualifiers.
SQL_QUALIFIER_USAGE Now SQL_CATALOG_USAGE: Identifies DML statements that support qualifier names.
SQL_QUOTED_IDENTIFIER_CASE: Indicates case sensitivity of quoted identifiers.
SQL_QUALIFIER_NAME_SEPARATOR Now SQL CATALOG_NAME_SEPARATOR: The character that separates the nam

SQL info_type Argument and Description
SQL_QUALIFIER_TERM: The term used to describe a qualifier.
SQL_QUALIFIER_LOCATION: The position of the qualifier in a qualified table name.
SQL_ROW_UPDATES: Indicates if keyset-driven or mixed cursors maintain row versions or values.
SQL_SCHEMA_TERM: The term used to describe a schema.
SQL_SCHEMA_USAGE: Indicates the SQL statements that may refer to schemas.
SQL_SCROLL_CONCURRENCY: Indicates the cursor concurrency control options supported by the server.
SQL_SCROLL_OPTIONS: Indicates the cursor scroll options supported by the server.
SQL_SEARCH_PATTERN_ESCAPE: The escape character that allows use of the wildcard characters % and _ in search p
SQL_SERVER_NAME: Indicates the name of the host.
SQL_SPECIAL_CHARACTERS: Indicates any special characters allowed in identifier names.
SQL_SQL_CONFORMANCE: Indicates the level of SQL-92 compliance.
SQL_SQL92_DATETIME_FUNCTIONS: Lists the datetime functions supported by the server.
SQL_SQL92_FOREIGN_KEY_DELETE_RULE: Indicates the server-enforced rules for using a foreign key in a DELETE st
SQL_SQL92_FOREIGN_KEY_UPDATE_RULE: Indicates the server-enforced rules for using a foreign key in an UPDATE
SQL_SQL92_GRANT: Indicates the supported GRANT statement clauses.
SQL_SQL92_NUMERIC_VALUE_FUNCTIONS: Lists the scalar numeric functions supported by the server and driver.
SQL_SQL92_PREDICATES, Identifies the predicates of a SELECT statement supported by the server.
SQL_SQL92_RELATIONAL_JOIN_OPERATORS: Identifies the relational join operators supported by the server.
SQL_SQL92_REVOKE: Identifies the clauses in a REVOKE statement that are supported by the server.
SQL_SQL92_ROW_VALUE_CONSTRUCTOR: Indicates the row value constructor expressions in a SELECT statement th
SQL_SQL92_STRING_FUNCTIONS: Lists the string scalar functions supported by the server and driver.
SQL_SQL92_VALUE_EXPRESSIONS: Indicates the value expressions supported by the server.
SQL_STANDARD_CLI_CONFORMANCE: Indicates the CLI standard the driver conforms to.
SQL_STATIC_CURSOR_ATTRIBUTES1: Describes the first set of static cursor attributes supported by the driver.
SQL_STATIC_CURSOR_ATTRIBUTES2: Describes the second set of static cursor attributes supported by the driver.
SQL_STATIC_SENSITIVITY: Indicates whether changes made to a static cursor by SQLSetPos() or UPDATE or DELETE s
SQL_STRING_FUNCTIONS: Lists the scalar string functions supported by the server and driver.
SQL_SUBQUERIES: Identifies the subquery predicates to a SELECT statement supported by the server.
SQL_SYSTEM_FUNCTIONS: Lists the scalar system functions supported by the server and driver.
SQL_TABLE_TERM: The term used to describe a table.
SQL_TIMEDATE_ADD_INTERVALS: Indicates the timestamp intervals supported by the server for the TIMESTAMPADD s
SQL_TIMEDATE_DIFF_INTERVALS: Indicates the timestamp intervals supported by the server for the TIMESTAMPDIFF s
SQL_TIMEDATE_FUNCTIONS: Indicates the date and time functions supported by the server.
SQL_TXN_CAPABLE: Identifies the transaction support offered by the server and driver.
SQL_TXN_ISOLATION_OPTION: Indicates the transaction isolation level supported by the server.
SQL_UNION: Indicates server support for the UNION clause.
SQL_USER_NAME: Identifies the name of the user connected to a database; may be different than the login name.
SQL_XOPEN_CLI_YEAR: The publication year of the X/Open specification that the driver manager complies with.

EDB_ODBC

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`

`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
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.

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. ========== SQLSetEnvAttr() function to set the values of environment attributes. The signature of the funct env_handle The environment handle. attribute attribute identifies the attribute whose value you wi 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 at- tribute . 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 cha 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. SQLSetEn- vAttr() 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 a SQLFreeHandle() is called for the connection. ODBC version 3.x allows you to allocate multiple e SQLSetAttr() . ========== ========== ========== SQLGetStmtAttr() and SQLSetStmtAttr() functions to retrieve and set the value of a statement attribu SQLGetStmtAttr() function returns the current value of statement attribute. The signature is: . stmt_handle The statement handle attribute 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: ========== ========== string_length_pointer A pointer to an SQLINTEGER that receives the number of bytes required to hol

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 . The following table lists the statement attributes supported by EDB-ODBC: ========== SQLSetStmtAttr() function to set the values of environment attributes. The signature is: .. code attribute attribute identifies the statement attribute whose value you wish to set. value_pointer value_pointer is a pointer to the location in memory that holds the value that will value_pointer can be a pointer to: - A null-terminated character string - A binary buffer - SQLLEN , SQLULEN or SQLUSMALLINT Value-pointer can also optionally hold one of the following SQLUINTEGER value - A SQLULEN value - A signed INTEGER (if attribute is a driver-specifi string_length If attribute is defined by ODBC and value_pointer points to a binary buffer or character 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: ========== ========== SQLGetDiagRec() function. SQLGetDiagRec() ---------- The SQLGetDiagRec() function returns status and error information from a diagnostic record written by the ODBC fun 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 ha SQL_HANDLE_DBC specifies a connection handle. - SQL_HANDLE_DESC specifies a descriptor ha handle The handle associated with the attribute error message. record_number The status record th 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 SQL_DIAG_NATIVE field). error_text_pointer Pointer to a memory buffer that receives the error text 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 n 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 . .. raw:: latex \newpage Supported ODBC API Functions ========== .. 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?
SQLAllocConnect()	Yes
SQLAllocEnv()	Yes
SQLAllocStmt()	Yes
SQLBindCol()	Yes
SQLCancel()	Yes
SQLColAttributes()	Yes
SQLConnect()	Yes
SQLDescribeCol()	Yes
SQLDisconnect()	Yes
SQLError()	Yes
SQLExecDirect()	Yes
SQLExecute()	Yes
SQLFetch()	Yes
SQLFreeConnect()	Yes
SQLFreeEnv()	Yes
SQLFreeStmt()	Yes

ODBC API Function Name	Supported by EDB-ODBC?
SQLGetCursorName()	Yes
SQLNumResultCols()	Yes
SQLPrepare()	Yes
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
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

ODBC API Function Name	Supported by EDB-ODBC?
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
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.

7.0 Security and Encryption

7.1 Scram Compatibility

The EDB ODBC driver provides SCRAM-SHA-256 support for Advanced Server versions 12, 11 and 10. This support is available from EDB ODBC 10.01.0000.01 release onwards.