EDB ODBC Connector

Version 16

Built at 2024-05-02T13:26:10

1 EDB ODBC Connector

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

While ODBC Connector provides a level of application portability, the portability is limited. It provides a connection but doesn't guarantee command compatibility. Commands that are acceptable in another database might not work in EDB Postgres 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, which handles named connections for Windows or odbc.ini which contains named connections for Linux
- The database-specific ODBC driver (ODBC Connector)
- The ODBC-compliant server (EDB Postgres Advanced Server)

You can write client applications in any language that has a binding for ODBC. C, MS-Access, and C++ are just a few.

2 Release notes

The EDB ODBC connector documentation describes the latest version of the EDB ODBC connector.

Release notes describe what's new in a release. When a minor or patch release introduces new functionality, indicators in the content identify the version that introduced the new feature.

Version	Release date
16.00.0000.01	09 Nov 2023

2.1 EDB ODBC Connector 16.00.0000.01 release notes

Released: 09 Nov 2023

EDB ODBC Connector 16.00.0000.01 includes the following enhancement:

Type	Description
Upstream merge	Merged with the upstream community driver version 16.00.0000 and 15.00.0000. See the community Release notes for details.
Enhancement	Added support for EDB Postgres Advanced Server version 16.1.

> **Note**
>
> If you are upgrading an earlier ODBC version on Ubuntu/Debian installation, you need to run the install command and specify the version number in the package name. For example the command for Ubuntu 20 is:

```
sudo apt-get install edb-odbc=16.00.0000.01-1.focal
```

If you get a message about downgrading, you can ignore it.

3 Supported platforms

The ODBC Connector is supported on the same platforms as EDB Postgres Advanced Server. To determine the platform support for the ODBC Connector, you can either refer to the platform support for EDB Postgres Advanced Server on the Platform Compatibility page on the EDB website or refer to Installing EDB ODBC Connector.

Supported database versions

This table lists the latest ODBC Connector versions and their supported corresponding EDB Postgres Advanced Server (EPAS) versions.

ODBC Connector	EPAS 16	EPAS 15	EPAS 14	EPAS 13	EPAS 12	EPAS 11
16.00.0000.01	Y	Y	Y	Y	Y	Y
13.02.0.02	N	Y	Y	Y	Y	Y
13.02.0.01	N	N	Y	Y	Y	Y
13.01.0.02	N	N	Y	Y	Y	Y
13.01.0.01	N	N	N	Y	Y	Y
13.00.0.01	N	N	N	Y	Y	Y

4 Installing EDB ODBC Connector

Select a link to access the applicable installation instructions:

Linux x86-64 (amd64)

Red Hat Enterprise Linux (RHEL) and derivatives

- RHEL 9, RHEL 8, RHEL 7
- Oracle Linux (OL) 9, Oracle Linux (OL) 8, Oracle Linux (OL) 7
- Rocky Linux 9, Rocky Linux 8
- AlmaLinux 9, AlmaLinux 8
- CentOS 7

SUSE Linux Enterprise (SLES)

- SLES 15, SLES 12

Debian and derivatives

- Ubuntu 22.04, Ubuntu 20.04
- Debian 11, Debian 10

Linux IBM Power (ppc64le)

Red Hat Enterprise Linux (RHEL) and derivatives

- RHEL 9, RHEL 8

SUSE Linux Enterprise (SLES)

- SLES 15, SLES 12

Windows

- Windows Server 2019

4.1 Installing EDB ODBC Connector on Linux x86 (amd64)

Operating system-specific install instructions are described in the corresponding documentation:

Red Hat Enterprise Linux (RHEL) and derivatives

- RHEL 9
- RHEL 8
- RHEL 7
- Oracle Linux (OL) 9
- Oracle Linux (OL) 8

- Oracle Linux (OL) 7
- Rocky Linux 9
- Rocky Linux 8
- AlmaLinux 9
- AlmaLinux 8
- CentOS 7

SUSE Linux Enterprise (SLES)

- SLES 15
- SLES 12

Debian and derivatives

- Ubuntu 22.04
- Ubuntu 20.04
- Debian 11
- Debian 10

4.1.1 Installing EDB ODBC Connector on RHEL 9 or OL 9 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

```
dnf repolist | grep enterprisedb
```

If no output is generated, the repository isn't installed.

To set up the EDB repository:

1. Go to EDB repositories.
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

- Install the EPEL repository:

```
sudo dnf -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm
```

Install the package

```
sudo dnf -y install edb-odbc
sudo dnf -y install edb-odbc-devel
```

4.1.2 Installing EDB ODBC Connector on RHEL 8 or OL 8 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

1. Go to EDB repositories.
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

- Install the EPEL repository:

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

Install the package

```
sudo dnf -y install edb-odbc
sudo dnf -y install edb-odbc-devel
```

4.1.3 Installing EDB ODBC Connector on AlmaLinux 9 or Rocky Linux 9 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.

4. Follow the instructions for setting up the EDB repository.

- Install the EPEL repository:

```
sudo dnf -y install epel-release
```

- Enable additional repositories to resolve dependencies:

```
sudo dnf config-manager --set-enabled crb
```

Install the package

```
sudo dnf -y install edb-odbc
sudo dnf -y install edb-odbc-devel
```

4.1.4 Installing EDB ODBC Connector on AlmaLinux 8 or Rocky Linux 8 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

- Install the EPEL repository:

```
sudo dnf -y install epel-release
```

- Enable additional repositories to resolve dependencies:

```
sudo dnf config-manager --set-enabled powertools
```

Install the package

```
sudo dnf -y install edb-odbc
sudo dnf -y install edb-odbc-devel
```

4.1.5 Installing EDB ODBC Connector on RHEL 7 or OL 7 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.
- Install the EPEL repository:

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

- Enable additional repositories to resolve dependencies:

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

Install the package

```
sudo yum -y install edb-odbc
sudo yum -y install edb-odbc-devel
```

4.1.6 Installing EDB ODBC Connector on CentOS 7 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.
- Install the EPEL repository:

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

Install the package

```
sudo yum -y install edb-odbc
sudo yum -y install edb-odbc-devel
```

4.1.7 Installing EDB ODBC Connector on SLES 15 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  zypper lr -E | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.
- Activate the required SUSE module:

  ```
  sudo SUSEConnect -p PackageHub/15.4/x86_64
  ```

- Refresh the metadata:

```
sudo zypper refresh
```

Install the package

```
sudo zypper -n install edb-odbc
sudo zypper -n install edb-odbc-devel
```

4.1.8 Installing EDB ODBC Connector on SLES 12 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  zypper lr -E | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.
- Activate the required SUSE module:

  ```
  sudo SUSEConnect -p PackageHub/12.5/x86_64
  sudo SUSEConnect -p sle-sdk/12.5/x86_64
  ```

- Refresh the metadata:

```
sudo zypper refresh
```

Install the package

```
sudo zypper -n install edb-odbc
sudo zypper -n install edb-odbc-devel
```

4.1.9 Installing EDB ODBC Connector on Ubuntu 22.04 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  apt-cache search enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

Install the package

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

4.1.10 Installing EDB ODBC Connector on Ubuntu 20.04 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  apt-cache search enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

Install the package

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

4.1.11 Installing EDB ODBC Connector on Debian 11 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:

- Installing EDB Postgres Advanced Server
- Installing PostgreSQL

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  apt-cache search enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

Install the package

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

4.1.12 Installing EDB ODBC Connector on Debian 10 x86_64

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

```
apt-cache search enterprisedb
```

If no output is generated, the repository isn't installed.

To set up the EDB repository:

1. Go to EDB repositories.
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

Install the package

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

4.2 Installing EDB ODBC Connector on Linux IBM Power (ppc64le)

Operating system-specific install instructions are described in the corresponding documentation:

Red Hat Enterprise Linux (RHEL)

- RHEL 9
- RHEL 8

SUSE Linux Enterprise (SLES)

- SLES 15
- SLES 12

4.2.1 Installing EDB ODBC Connector on RHEL 9 ppc64le

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
    - Installing EDB Postgres Advanced Server
    - Installing PostgreSQL
- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.
- Install the EPEL repository:

  ```
  sudo dnf -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm
  ```
- Refresh the cache:

  ```
  sudo dnf makecache
  ```

Install the package

```
sudo dnf -y install edb-odbc
sudo dnf -y install edb-odbc-devel
```

4.2.2 Installing EDB ODBC Connector on RHEL 8 ppc64le

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:

- Installing EDB Postgres Advanced Server
- Installing PostgreSQL

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

- Install the EPEL repository:

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

- Refresh the cache:

  ```
  sudo dnf makecache
  ```

Install the package

```
sudo dnf -y install edb-odbc
sudo dnf -y install edb-odbc-devel
```

4.2.3 Installing EDB ODBC Connector on SLES 15 ppc64le

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server

- Installing PostgreSQL

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  zypper lr -E | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

- Activate the required SUSE module:

  ```
  sudo SUSEConnect -p PackageHub/15.4/ppc64le
  ```

- Refresh the metadata:

  ```
  sudo zypper refresh
  ```

Install the package

```
sudo zypper -n install edb-odbc
sudo zypper -n install edb-odbc-devel
```

4.2.4 Installing EDB ODBC Connector on SLES 12 ppc64le

Prerequisites

Before you begin the installation process:

- Install Postgres on a host that the product can connect to using a connection string. It doesn't need to be on the same host. See:
  - Installing EDB Postgres Advanced Server
  - Installing PostgreSQL

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  zypper lr -E | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

- Activate the required SUSE module:

  ```
  sudo SUSEConnect -p PackageHub/12.5/ppc64le
  sudo SUSEConnect -p sle-sdk/12.5/ppc64le
  ```

- Refresh the metadata:

  ```
  sudo zypper refresh
  ```

Install the package

```
sudo zypper -n install edb-odbc
sudo zypper -n install edb-odbc-devel
```

4.3 Installing the ODBC Connector on Windows

EDB provides a graphical interactive installer for Windows. You can access it two ways:

- Download the graphical installer from the Downloads page, and invoke the installer directly. See Installing directly.
- Use Stack Builder (with PostgreSQL) or StackBuilder Plus (with EDB Postgres Advanced Server) to download the EDB installer package, and invoke the graphical installer. See Using Stack Builder or StackBuilder Plus.

Installing directly

After downloading the graphical installer, to start the installation wizard, assume sufficient privileges (superuser or administrator) and double-click the installer icon. If prompted, provide a password.

In some versions of Windows, to invoke the installer with Administrator privileges, you need to right-click the installer icon and select **Run as Administrator** from the context menu.

Proceed to Using the graphical installer.

Using Stack Builder or StackBuilder Plus

If you're using PostgreSQL, you can invoke the graphical installer with Stack Builder. SeeUsing Stack Builder.

If you're using EDB Postgres Advanced Server, you can invoke the graphical installer with StackBuilder Plus. SeeUsing StackBuilder Plus.

1. In Stack Builder or StackBuilder Plus, follow the prompts until you get to the module selection page.

   On the Welcome page, select the target server installation from the list of available servers. If your network requires you to use a proxy server to access the internet, select **Proxy servers** and specify a server. Select **Next**.

2. Expand the **Database Drivers** node and do one of the following:

   - In Stack Builder, select **pgODBC**.
   - In StackBuilder Plus, select **EnterpriseDB ODBC Connector**.

3. Proceed to Using the graphical installer.

Using the graphical installer

1. Select the installation language and select **OK**.
2. On the Setup ODBC page, select **Next**.
3. Browse to a directory where you want ODBC to be installed, or leave the directory set to the default location. Select **Next**.
4. On the Ready to Install page, select **Next**.

   An information box shows the installation progress of the selected components.

5. When the installation is complete, select **Finish**.

4.4 Upgrading a Linux installation

Upgrading from RPM or SLES installations

If you have an existing ODBC Connector installation on a Linux platform, you can upgrade your repository configuration file, which enables access to the current EDB repository. Then you can upgrade to a more recent version of ODBC Connector.

To update the `edb.repo` file:

```
 # Update your repository configuration file
sudo <package-manager> upgrade edb-repo

# Upgrade the installed product
sudo <package-manager> upgrade edb-odbc

sudo <package-manager> upgrade edb-odbc-devel
```

Where `<package-manager>` is the package manager used with your operating system:

Package manager	Operating system
dnf	RHEL 8/9 and derivatives
yum	RHEL 7 and derivatives, CentOS 7
zypper	SLES

Upgrading from Debian/Ubuntu installations

If you are upgrading from an earlier Debian/Ubuntun installation, run the install command and specify the version number in the package name. For example:

```
sudo apt-get install edb-odbc=13.02.0000.02-1.bionic
```

If you get a message about downgrading, you can ignore it.

5 Creating a data source

When a client application tries to establish a connection with a server, it typically provides a data source name (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 usually named `/etc/odbc.ini`. The name and location can 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. 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 on Windows creates named data sources for ODBC connections. If you're using Windows, the tool is called the ODBC Data Source Administrator. If you are using Linux, you can create the named data sources using the odbc.ini file in a text editor of your choice.

Adding a data source definition in Windows and Adding a data source definition in Linux walk you through adding a data source in Windows using the graphical tool and Linux using the command line. During the process of defining a data source, you'll be required to specify a set of connection properties. EDB-ODBC connection properties contains information about optional data source connection properties. You can specify connection properties with a graphical tool (on Windows) or edit the odbc.ini file with a text editor (on Linux).

6 Connection properties

The following table describes the connection properties that you can specify through the dialog boxes 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're connecting.
Driver	Driver	EDB-ODBC	The name of the ODBC driver.
Server	Servername	Localhost	The name or IP address of the server that you're connecting to.
dbms_name	dbms_name	EnterpriseDB	Database system. Either EnterpriseDB or PostgreSQL.
Description	Description		Descriptive name of the data source.
User Name	Username		The name of the user that this data source uses to connect to the server.
Password	Password		The password of the user associated with this named data source.
CPTimeout	CPTimeout	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 roll back 1 - Roll back the transaction 2 - Roll back the statement
Usage Count	UsageCount	1	The number of installations using this driver.
Read Only	ReadOnly	No	Specifies that the connection is READONLY.

Property	Keyword name	Default value	Description
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).
Recognize Unique Indexes	UniqueIndex	On	If enabled, the `SQLStatistics()` function reports unique indexes. If not enabled, the `SQLStatistics()` function reports that indexes allow duplicate values.
Use Declare/Fetch	UseDeclareFetch	Off	If enabled, the driver uses 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 calls `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:m[ylog](<>)<process-id>`. 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. Don't specify `2` if you 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 disabled, the driver treats values of unknown type as `SQL_VARCHAR` values. By default, values of unknown type are treated as `Y` values.
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	If `TextAsLongVarChar` is on, the driver reports TEXT values are `MaxLongVarcharSize` bytes long. If `UnknownAsLongVarChar` is on, columns of unknown type are `MaxLongVarcharSize` bytes long. Otherwise, they are reported to be `MaxVarcharSize` bytes in length.
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 semicolon-delimited list of prefixes that indicate that a table is a system table. By default, the list contains `[dd](<>);`.

Property	Keyword name	Default value	Description
Cumulative Row Count for Insert	MapSqlParcNoBatch	Off/0	If enabled, the `SQLRowCount()` function returns a single, cumulative row count for the entire array of parameter settings for an `INSERT` statement. If disabled, an individual row count is 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 them back to line-feed characters when sending character values to the server. By default, this option is enabled.
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're of type `SQL_LONGVARBINARY`. If disabled, BYTEA values are treated as `SQL_VARBINARY` values.
Bytea as LO	ByteaAsLO	False	If enabled, the driver treats BYTEA values as if they're large objects.
Row versioning	RowVersioning	Off	The `Row Versioning` option specifies if the driver includes 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 metadata from the cursor. If `No`, the driver executes the command as soon as you request any metadata.
True is -1	TrueIsMinus1	Off/0	`TrueIsMinus1` tells the driver to return BOOL values of TRUE as `-1`. If this option isn't enabled, the driver returns 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 sends a GSSAPI authentication request to the server. Windows only.
Int8 As	BI	0	The value of `BI` determines how the driver treats `BIGINT` values: If -5 as a `SQL_BIGINT`, If 2 as a `SQL_NUMERIC`, If 8 as a `SQL_DOUBLE`, If 4 as a `SQL_INTEGER`, If 12 as a `SQL_VARCHAR`, If 0 (on an MS Jet client), as a `SQL_NUMERIC`, If 0 on any other client, as a `SQL_BIGINT`.

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're 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, information is returned only 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 lower case.
Disable Genetic Optimizer	Optimizer	Yes/1	Disables the genetic query optimizer.
Allow Keyset	UpdatableCursors	Yes/1	Allow Keyset-driven cursors
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` include only 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. To open the ODBC Data Source Administrator, in the Control Panel, open the **Administrative Tools** menu and double-click the appropriate ODBC Data Sources icon (32-bit or 64-bit).

Select **Add** to open the Create New Data Source dialog box. Select **EnterpriseDB (ANSI)** or **EnterpriseDB (UNICODE)** from the list of drivers and select **Finish**.

Use the fields on the EnterpriseDB ODBC Driver dialog box to define the named data source:

- Enter the database name in the **Database** field.
- Enter the host name or IP address of EDB Postgres 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 list next to the **SSL Mode** label is active, allowing you to use SSL and

other EDB Postgres 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.

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

Select **Global** to open a dialog where you can specify logging options for the EDB-ODBC driver (not the data source, but the driver).

- Select **Disable Genetic Optimizer** to disable the genetic query optimizer. By default, the query optimizer is on.
- Select **KSQO (Keyset Query Optimization)** to enable server-side support for keyset queries. By default, **Keyset Query Optimization** is on.
- Select **Recognize Unique Indexes** to force the `SQLStatistics()` function to report unique indexes. If the option is not selected, the `SQLStatistics()` function reports that all indexes allow duplicate values. By default, **Recognize Unique Indexes** is on.
- Select **Use Declare/Fetch** to specify for the driver to use server-side cursors whenever your application executes a `SELECT` command. By default, **Use Declare/Fetch** is off.
- Select **CommLog (C:\psqlodbc_xxxx.log)** to record all client/server traffic in a log file. By default, logging is off.
- Select **Parse Statements** to specify for the driver (rather than the server) to attempt to parse simple `SELECT` statements when you call the `SQLNumResultCols()`, `SQLDescribeCol()`, or `SQLColAttributes()` function. By default, this option is off.
- Select **Cancel as FreeStmt (Exp)** to specify for the `SQLCancel()` function to call `SQLFreeStmt(SQLClose)` on your behalf. By default, this option is off.
- Select **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.

- Select **Maximum** to specify for the driver to 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 cleared, the driver returns the size specified in the **Max VarChar** field in the **Miscellaneous** box.
- Select **Don't know** to specify for the driver to report a length of **unknown**.
- Select **Longest** to specify for the driver to search the result set and report the longest value found. Don't specify **Longest** if **UseDeclareFetch** is enabled.)

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

- Select **Text as LongVarChar** to treat TEXT values as if they are of type `SQL_LONGVARCHAR`. If cleared, the driver treats TEXT values as `SQL_VARCHAR` values. By default, TEXT values are treated as `SQL_LONGVARCHAR` values.
- Select **Unknowns as LongVarChar** to specify for the driver to treat values of unknown type as `SQL_LONGVARCHAR` values. If cleared, the driver treats values of unknown type as `SQL_VARCHAR` values. By default, values of unknown type are treated as `SQL_VARCHAR` values.
- Select **Bools as Char** to specify for the driver to treat BOOL values as `SQL_CHAR` values. If cleared, 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 semicolon-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 box to their default settings by selecting **Defaults**.

Select **Apply** button to apply any changes to the data source properties. Select **OK** to apply any changes and exit.

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

- Select **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.

- Select **Show System Tables** to include system tables in the result set of the `SQLTables()` function. If the option is enabled, the driver includes 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 box. By default, this option is off.
- Select **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 on.
- Select **Show sys/SQL Tables** to enable accessing database tables in a linked server with MS SQL.
- Select **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 cleared, 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.
- Select **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 them back to line-feed characters when sending character values to the server. By default, this option is on.
- Select **Updatable Cursors** to specify for the driver to permit positioned `UPDATE` and `DELETE` operations with the `SQLSetPos()` or `SQLBulkOperations()` functions. By default, this option is on.
- Select **bytea as LO** to specify for the driver to treat `BYTEA` values as if they're `SQL_LONGVARBINARY` values. If cleared, EDB-ODBC treats `BYTEA` values as if they are `SQL_VARBINARY` values. By default, `BYTEA` values are treated as `SQL_VARBINARY` values.
- Select **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, this option is off.
- Select **Disallow Premature** to specify for the driver to retrieve metadata about a query (the number of columns in a result set or the column types) without actually executing the query. If this option is cleared, the driver executes the query when you request metadata about the query. By default, this option is off.
- Select **True is -1** to tell the driver to return BOOL values of `True` as a `-1`. If this option is cleared, the driver returns BOOL values of `True` as `1`. The driver always returns BOOL values of `False` as `0`.
- Select **Server side prepare** to tell the driver to use the `PREPARE` and `EXECUTE` commands to implement the `Prepare/Execute` model. By default, this option is on.
- Select **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 options in the **Int8 As** box to specify how the driver returns `BIGINT` values to the client. Select **default** to specify the default type of **NUMERIC** if the client is MS Jet. Select **BIGINT** if the client is any other ODBC client. You can optionally specify for the driver to 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`. For **Extra Opts**, you can specify the options shown in the table.

Option	Specifies
0x1	Forces the output of short-length formatted connection string. Select this option when you're using the MFC CDatabase class.
0x2	Allows MS Access to recognize PostgreSQL's serial type as AutoNumber type.
0x4	Returns 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, information is returned only 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 options 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 options that specify how the driver handles error handling.

Option	Specifies
Transaction	If the driver encounters an error, it rolls back the current transaction.
Statement	If the driver encounters an error, it rolls back the current statement.
Nop	If the driver encounters an error, you must manually roll back 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:

- Select **Show Column** to include the `OID` column in the result set of the `SQLColumns()` function. If cleared, the `OID` column is hidden from `SQLColumns()`.
- Select **Fake Columns** to specify for the `SQLStatistics()` function to report that a unique index exists on each `OID` column.

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

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

After you define the connection properties for the named data source, select **Apply** to apply the options. Select **OK** to save the options and exit.

Select **Global** (on the EnterpriseDB ODBC Driver dialog box) to open the Global Settings dialog box. The options on this dialog box control logging options for the EDB-ODBC driver. Use this dialog box 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.

- Select **CommLog** to record all client/server traffic in a log file. The log file is named `C:\psqlodbc_<process-id>`, where `<process-id>` is the name of the process in use.
- Select **Mylog** to keep a log file of the driver's activity. The log file is named `c:\mylog_<process-id>`, where `<process-id>` is the name of the process in use.
- Specify a location for the log files in the **Folder for logging** field. After you entered the connection information for the named data source, select **Test** to verify that the driver manager can connect to the defined data source.

Select **OK** to exit the Connection Test dialog box. If the connection is successful, select **Save** 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

You can define named data sources on Linux in a text file that the driver manager reads to determine how to connect to the database. The driver manager usually looks for named data sources in two places:

- `/user/.odbc.ini` — Named data source that is available only to the current user.
- `/etc/odbc.ini` — Named data source that is available to all users.

There is no difference in the structure of these files and the same connection properties can be used in both files, only the location of the two files is different. If both files are available on a system, the user data source `/user/.odbc.ini` overrides the system data source `/etc/odbc.ini`.

A data source definition contains a data source name enclosed in square brackets and a list of driver and connection properties in the form of a property name followed by equal sign (=) followed by property value.

A typical user or system data source definition for the EDB-ODBC driver:

```
[EnterpriseDB]
```

```
Description = EnterpriseDB
DSN
Driver = /usr/edb/odbc/lib/edb-odbc.so
Trace = yes
TraceFile = /tmp/odbc.log
Database =
edb
Servername = localhost
UserName = enterprisedb
Password = manager
Port = 5444
```

Possible properties include:

Property	Description
[EnterpriseDB]	Name of your data source. You can use any name.
Description	Description of the named data source.
Driver	Path of the EDB-ODBC driver library file (edb-odbc.so).
Trace	`Yes` turns on the unixODBC driver's trace utility that records the sequence of calls made from an ODBC application to a log file. Using the trace utility can slow down an application.
TraceFile	File to receive information returned by the trace utility.
Database	EDB Postgres Advanced Server database.
Servername	Host name or IP address of the EDB Postgres Advanced Server.
Username	Name of a user.
Password	Password for the user.
Port	Port number.
Protocol	Front-end/back-end protocol version. The default value is 7.4. You can optionally specify protocol versions 7.4, 6.4, 6.3, or 6.2.
ReadOnly	`Yes` prevents the driver from executing the following commands: INSERT, UPDATE, DELETE, CREATE, ALTER, DROP, GRANT, REVOKE, or LOCK. Also prevents any calls that use the ODBC procedure call escape syntax (`call=procedure-name?`). The default value is `No` .
RowVersioning	`Yes` includes 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` . The default value is `No` .
ShowSystemTables	`Yes` includes system tables in the result set of the `SQLTables()` function. The default value is `No` .
ShowOidColumn	`Yes` includes the `OID` column in the result set of the `SQLColumns()` function. If `ShowOidColumn` is set to `No` , the `OID` column is hidden from `SQLColumns()` . The default value is `No` .
FakeOidIndex	`Yes` specifies the `SQLStatistics()` function to 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` .
ConnSettings	List of parameter assignments for the driver to use when opening this connection.

7 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 EDB Postgres 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 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 EDB Postgres 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 to 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` — 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 used only if the requested value is returned in the form of a character string.
- `string_length_pointer` — 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 isn't provided by `SQLGetInfo()`.

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

- `SQLUINTEGER` bitmask
- `SQLUINTEGER` flag
- `SQLUINTEGER` binary value
- `SQLUSMALLINT` value
- `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 EDB Postgres Advanced Server connection.

SQL info_type argument and description**	EDB_ODBC/EDB Postgres Advanced Server returns
SQL_ACCESSIBLE_PROCEDURES: Indicates if procedures returned by `SQLProcedures()` can be executed by the application.	Returns `N`. Some procedures executed by the `SQLProcedures()` function might be executed by the application.
SQL_ACCESSIBLE_TABLES: Indicates if the user has SELECT privileges on all table names returned by `SQLTables()`.	Returns `N`. The user might not have select privileges on one or more tables returned by the `SQLTables()` function.
SQL_ACTIVE_CONNECTIONS prev. SQL_MAX_DRIVER_CONNECTIONS: Indicates the maximum number of connections EDB-ODBC can support.	Returns `0`. There's no specified limit to the number of connections allowed.
SQL_ACTIVE_ENVIRONMENTS: The number of active environments EDB-ODBC can support.	Returns `0`. There's 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's no specified limit to the number of active statements allowed.
SQL_AGGREGATE_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 aren't supported.
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 isn't 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 can 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 info_type argument and description**	EDB_ODBC/EDB Postgres Advanced Server returns
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_CONCAT_NULL_BEHAVIOR: Indicates how the server handles concatenation of NULL values.	Returns `SQL_CB_NON_NULL`. Concatenating a NULL value and a non-NULL value results in a NULL value.
SQL_CONVERT_BIGINT: Indicates conversion support from the `BIGINT` type using the `CONVERT` function.	Returns `0`. The server doesn't support conversion.
SQL_CONVERT_BINARY: Indicates conversion support from the BINARY type using the `CONVERT` function.	Returns `0`. The server doesn't 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 doesn't support conversion.
SQL_CONVERT_DATE: Indicates conversion support from the DATE type using the `CONVERT` function.	Returns `0`. The server doesn't support conversion.
SQL_CONVERT_DECIMAL: Indicates conversion support from the DECIMAL type using the `CONVERT` function.	Returns `0`. The server doesn't support conversion.
SQL_CONVERT_DOUBLE: Indicates conversion support from the DOUBLE type using the `CONVERT` function.	Returns `0`. The server doesn't support conversion.
SQL_CONVERT_FLOAT: Indicates conversion support from the FLOAT type using the `CONVERT` function.	Returns `0`. The server doesn't 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 isn't currently supported.
SQL_CONVERT_INTERVAL_YEAR_MONTH: Indicates conversion support from the INTERVAL_YEAR_MONTH type using the `CONVERT` function.	This info_type isn't currently supported.
SQL_CONVERT_LONGVARBINARY: Indicates conversion support for the LONG_VARBINARY type using the `CONVERT` function.	Returns `0`. The server doesn't support conversion.
SQL_CONVERT_LONGVARCHAR: Indicates conversion support for the LONGVARCHAR type using the `CONVERT` function.	Returns `0`. The server doesn't support conversion.
SQL_CONVERT_NUMERIC: Indicates conversion support for the NUMERIC type using the `CONVERT` function.	Returns 0. The server doesn't support conversion.
SQL_CONVERT_REAL: Indicates conversion support for the REAL type using the `CONVERT` function	Returns `0`. The server doesn't support conversion.
SQL_CONVERT_SMALLINT: Indicates conversion support for the SMALLINT type using the `CONVERT` function.	Returns `SQL_CVT_INTEGER`, `SQL_CVT_BIT`.

SQL info_type argument and description**	EDB_ODBC/EDB Postgres Advanced Server returns
SQL_CONVERT_TIME: Indicates conversion support for TIME type using the `CONVERT` function.	Returns `0`. The server doesn't support conversion.
SQL_CVT_TIMESTAMP: Indicates conversion support for TIMESTAMP type using the `CONVERT` function.	Returns `0`. The server doesn't 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 doesn't 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 doesn't 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 doesn't 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 doesn't 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_CREATE_ASSERTION: Indicates support for the `CREATE ASSERTION` statement.	Returns `0`. The `CREATE ASSERTION` statement isn't supported.
SQL_CREATE_CHARACTER_SET: Indicates support for the `CREATE CHARACTER` statement.	Returns `0`. The `CREATE CHARACTER` statement isn't supported.
SQL_CREATE_COLLATION: Indicates support for the `CREATE COLLATION` statement.	Returns `0`. The `CREATE COLLATION` statement isn't supported.
SQL_CREATE_DOMAIN: Indicates support for the `CREATE DOMAIN` statement.	Returns `0`. The `CREATE DOMAIN` statement isn't 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 isn't 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 isn't currently supported.
SQL_DATA_SOURCE_NAME: Returns the server name used during connection.	The value returned is determined by the connection properties.

SQL info_type argument and description**	EDB_ODBC/EDB Postgres Advanced Server returns
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_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`. Nonrepeatable or phantom reads are possible. Dirty reads aren't.
SQL_DESCRIBE_PARAMETER: Indicates support for the `DESCRIBE INPUT` statement.	Returns `N`. The `DESCRIBE INPUT` statement isn't 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_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 info_type argument and description**	EDB_ODBC/EDB Postgres Advanced Server returns
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 isn't 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_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_INTO`.
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`. EDB Postgres Advanced Server doesn't 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 info_type argument and description**	EDB_ODBC/EDB Postgres Advanced Server returns
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_MAX_COLUMN_NAME_LEN: The maximum length of a column name.	Returns `64`. Column names can't 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 can't 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 can't 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_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 info_type argument and description**	EDB_ODBC/EDB Postgres Advanced Server returns
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`. EDB Postgres Advanced Server supports multiple result sets.
SQL_MULTIPLE_ACTIVE_TXN: Indicates if the server supports multiple active transactions.	Returns `Y`. EDB Postgres 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`. EDB Postgres Advanced Server doesn't 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`. EDB Postgres 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_ODBC_SAG_CLI_CONFORMANCE: Indicates the SQL Access Group compliance level that the driver adheres to.	Returns `SQL_OSCC_NOT_COMPLIANT`. The driver isn't 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 doesn't 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 don't 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 returns 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. Returns `SQL_PARC_NO_BATCH` if a single, cumulative row count is available for the entire array.

SQL info_type argument and description**	EDB_ODBC/EDB Postgres Advanced Server returns
SQL_PARAM_ARRAY_SELECTS: Indicates if the server returns one result set or a separate result set for each element in an array (or if the driver doesn't 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_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 might 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 CATALOG_NAME_SEPARATOR prev. SQL_QUALIFIER_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_SCHEMA_USAGE: Indicates the SQL statements that might 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 info_type argument and description**	EDB_ODBC/EDB Postgres Advanced Server returns
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_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/EDB Postgres 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_TIMEDATE_ADD_INTERVALS: Indicates the timestamp intervals supported by the server for the `TIMESTAMPADD` scalar function.	Returns `0`.
SQL_TIMEDATE_DIFF_INTERVALS: Indicates the timestamp intervals supported by the server for the `TIMESTAMPDIFF` scalar function.	Returns `0`.
SQL_TIMEDATE_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_USER_NAME: Identifies the name of the user connected to a database. Can 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 want to retrieve.
- `value_pointer` — A pointer to the location in memory to 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` isn't 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	

Attribute	Supported?	Notes
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` — Identifies the attribute whose value you want to set.
- `value_pointer` — A pointer to the value that the attribute assumes.
- `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 shown in the table.

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 timeframe in which they can be set. Some attributes must be set before the connection is established, while others can be set only 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` — Identifies the attribute whose value you want to retrieve.
- `value_pointer` — A pointer to the location in memory to 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 isn't 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` isn't 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 want to set.
- `value_pointer` — A pointer to the value assigned to the attribute. The value is 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` isn't 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 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 a 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` — The attribute value.
- `value_pointer` — A pointer to the location in memory to 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 `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 an `SQLINTEGER` that receives the number of bytes required to hold the requested value. If `value_pointer` is NULL, `string_length_pointer` isn't returned.

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

Attribute	Supported?	Restrictions?
SQL_ATTR_APP_PARAM_DESC	Yes	
SQL_ATTR_APP_ROW_DESC	Yes	
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

Attribute	Supported?	Restrictions?
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` — The environment handle.
- `attribute` — Identifies the statement attribute whose value you want to set.

- `value_pointer` — A pointer to the location in memory that holds the value to 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 `SQLUINTEGER` 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_UINTEGER`
Any other type	`SQL_IS_POINTER`

Error handling

You can retrieve diagnostic information for these ODBC functions mentioned by using 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` is 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?
SQLAllocConnect()	Yes
SQLAllocEnv()	Yes
SQLAllocStmt()	Yes
SQLBindCol()	Yes
SQLCancel()	Yes
SQLColAttributes()	Yes
SQLConnect()	Yes
SQLDescribeCol()	Yes
SQLDisconnect()	Yes
SQLError()	Yes
SQLExecDirect()	Yes
SQLExecute()	Yes

ODBC API function name	Supported by EDB-ODBC?
SQLFetch()	Yes
SQLFreeConnect()	Yes
SQLFreeEnv()	Yes
SQLFreeStmt()	Yes
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

ODBC API function name	Supported by EDB-ODBC?
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 EDB Postgres 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

ODBC data type	Corresponding EDB Postgres Advanced Server data type
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

Prerequisite for ADO users

You must execute `Command.Prepared = True` before executing `Command.Execute`.

Thread safety

EDB-ODBC is thread safe.

8 Scram compatibility

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