TDS Foreign data wrapper

Logo

About

This is a PostgreSQL foreign data wrapper that can connect to databases that use the Tabular Data Stream (TDS) protocol, such as Sybase databases and Microsoft SQL server.

This foreign data wrapper requires a library that implements the DB-Library interface, such as FreeTDS. This has been tested with FreeTDS, but not the proprietary implementations of DB-Library.

This should support PostgreSQL 9.2+.

The current version does not yet support JOIN push-down, or write operations.

It does support WHERE and column pushdowns when *match_column_names* is enabled.

CentOS 7	Rocky Linux 8	Ubuntu 20.04

Installing on RHEL and clones (CentOS, Rocky Linux, AlmaLinux, Oracle. . .)

See installing tds_fdw on CentOS.

Installing on Ubuntu

See installing tds_fdw on Ubuntu.

Installing on Debian

See installing tds_fdw on Debian.

Installing on openSUSE

See installing tds_fdw on openSUSE.

Installing on OSX

See installing tds_fdw on OSX.

Installing on Alpine (and Docker)

See installing tds_fdw on Alpine.

Usage

Foreign server

See creating a foreign server.

Foreign table

See creating a foreign table.

User mapping

See creating a user mapping.

Foreign schema

See importing a foreign schema.

Variables

See variables.

`EXPLAIN`

`EXPLAIN (VERBOSE)` will show the query issued on the remote system.

Notes about character sets/encoding

1. If you get an error like this with MS SQL Server when working with Unicode data:

   > NOTICE: DB-Library notice: Msg #: 4004, Msg state: 1, Msg: Unicode data in a Unicode-only collation or ntext data cannot be sent to clients using DB-Library (such as ISQL) or ODBC version 3.7 or earlier., Server: PILLIUM SQLEXPRESS, Process: , Line: 1, Level: 16
   > ERROR: DB-Library error: DB #: 4004, DB Msg: General SQL Server error: Check messages from the SQL Server, OS #: -1, OS Msg: (null), Level: 16

   and Choosing a TDS protocol version.

2. Although many newer versions of the TDS protocol will only use USC-2 to communicate with the server, FreeTDS converts the UCS-2 to the client character set of your choice. To set the client character set, you can set *client charset* in *freetds.conf*. See

Encrypted connections to MSSQL

Support

If you find any bugs, or you would like to request enhancements, please submit your comments on the project's GitHub Issues page.

Additionally, I do subscribe to several PostgreSQL mailing lists including *pgsql-general* and *pgsql-hackers*. If tds_fdw is mentioned in an email sent to one of those lists, I typically see it.

Debugging

See Debugging

TDS Foreign data wrapper

Importing a Foreign Schema

Options

Foreign schema parameters accepted:

- *import_default*

Required: No

Default: false

Controls whether column DEFAULT expressions are included in the definitions of foreign tables.

- *import_not_null*

Required: No

Default: true

Controls whether column NOT NULL constraints are included in the definitions of foreign tables.

Example

```
IMPORT FOREIGN SCHEMA dbo
    EXCEPT (mssql_table)
    FROM SERVER mssql_svr
    INTO public
    OPTIONS (import_default 'true');
```

TDS Foreign data wrapper

Creating a Foreign Server

Options

Foreign server parameters accepted:

- *servername*

Required: Yes

Default: 127.0.0.1

The servername, address or hostname of the foreign server server.

This can be a DSN, as specified in *freetds.conf*. See FreeTDS name lookup.

You can set this option to a comma separated list of server names, then each server is tried until the first connection succeeds.
This is useful for automatic fail-over to a secondary server.

- *port*

Required: No

The port of the foreign server. This is optional. Instead of providing a port here, it can be specified in *freetds.conf* (if *servername* is a DSN).

- *database*

Required: No

The database to connect to for this server.

- *dbuse*

Required: No

Default: 0

This option tells tds_fdw to connect directly to *database* if *dbuse* is 0. If *dbuse* is not 0, tds_fdw will connect to the server's default database, and then select *database* by calling DB-Library's dbuse() function.

For Azure, *dbuse* currently needs to be set to 0.

- *language*

Required: No

The language to use for messages and the locale to use for date formats. FreeTDS may default to *us_english* on most systems. You can probably also change this in *freetds.conf*.

For information related to this for MS SQL Server, see SET LANGUAGE in MS SQL Server.

For information related to Sybase ASE, see Sybase ASE login options and SET LANGUAGE in Sybase ASE.

- *character_set*

Required: No

The client character set to use for the connection, if you need to set this for some reason.

For TDS protocol versions 7.0+, the connection always uses UCS-2, so this parameter does nothing in those cases. See Localization and TDS 7.0.

- *tds_version*

Required: No

The version of the TDS protocol to use for this server. See Choosing a TDS protocol version and History of TDS Versions.

- *msg_handler*

Required: No

Default: blackhole

The function used for the TDS message handler. Options are "notice" and "blackhole." With the "notice" option, TDS messages are turned into PostgreSQL notices. With the "blackhole" option, TDS messages are ignored.

- *fdw_startup_cost*

Required: No

A cost that is used to represent the overhead of using this FDW used in query planning.

- *fdw_tuple_cost*

Required: No

A cost that is used to represent the overhead of fetching rows from this server used in query planning.

Foreign table parameters accepted in server definition: Some foreign table options can also be set at the server level. Those include:

- *use_remote_estimate*
- *row_estimate_method*

Example

```
CREATE SERVER mssql_svr
    FOREIGN DATA WRAPPER tds_fdw
    OPTIONS (servername '127.0.0.1', port '1433', database 'tds_fdw_test', tds_version '7.1'
```

TDS Foreign data wrapper

Creating a Foreign Table

Options

Foreign table parameters accepted:

- *query*

Required: Yes (mutually exclusive with *table*)

The query string to use to query the foreign table.

- *schema_name*

Required: No

The schema that the table is in. The schema name can also be included in *table_name*, so this is not required.

- *table_name*

Aliases: *table*

Required: Yes (mutually exclusive with *query*)

The table on the foreign server to query.

- *match_column_names*

Required: No

Whether to match local columns with remote columns by comparing their table names or whether to use the order that they appear in the result set.

- *use_remote_estimate*

Required: No

Whether we estimate the size of the table by performing some operation on the remote server (as defined by *row_estimate_method*), or whether we just use a local estimate, as defined by *local_tuple_estimate*.

- *local_tuple_estimate*

Required: No

A locally set estimate of the number of tuples that is used when *use_remote_estimate* is disabled.

- *row_estimate_method*

Required: No

Default: `execute`

This can be one of the following values:

- `execute`: Execute the query on the remote server, and get the actual number of rows in the query.
- `showplan_all`: This gets the estimated number of rows using MS SQL Server's SET SHOWPLAN_ALL.

Foreign table column parameters accepted:

- *column_name*

Required: No

The name of the column on the remote server. If this is not set, the column's remote name is assumed to be the same as the column's local name. If match_column_names is set to 0 for the table, then column names are not used at all, so this is ignored.

Example

Using a *table_name* definition:

```
CREATE FOREIGN TABLE mssql_table (
    id integer,
    data varchar)
    SERVER mssql_svr
    OPTIONS (table_name 'dbo.mytable', row_estimate_method 'showplan_all');
```

Or using a *schema_name* and *table_name* definition:

```
CREATE FOREIGN TABLE mssql_table (
    id integer,
    data varchar)
    SERVER mssql_svr
    OPTIONS (schema_name 'dbo', table_name 'mytable', row_estimate_method 'showplan_all');
```

Or using a *query* definition:

```
CREATE FOREIGN TABLE mssql_table (
    id integer,
    data varchar)
    SERVER mssql_svr
    OPTIONS (query 'SELECT * FROM dbo.mytable', row_estimate_method 'showplan_all');
```

Or setting a remote column name:

```
CREATE FOREIGN TABLE mssql_table (
    id integer,
    col2 varchar OPTIONS (column_name 'data'))
    SERVER mssql_svr
    OPTIONS (schema_name 'dbo', table_name 'mytable', row_estimate_method 'showplan_all');
```

TDS Foreign data wrapper

Logo

About

This is a PostgreSQL foreign data wrapper that can connect to databases that use the Tabular Data Stream (TDS) protocol, such as Sybase databases and Microsoft SQL server.

This foreign data wrapper requires a library that implements the DB-Library interface, such as FreeTDS. This has been tested with FreeTDS, but not the proprietary implementations of DB-Library.

This should support PostgreSQL 9.2+.

The current version does not yet support JOIN push-down, or write operations.

It does support WHERE and column pushdowns when *match_column_names* is enabled.

CentOS 7	Rocky Linux 8	Ubuntu 20.04

Installing on RHEL and clones (CentOS, Rocky Linux, AlmaLinux, Oracle. . .)

See installing tds_fdw on CentOS.

Installing on Ubuntu

See installing tds_fdw on Ubuntu.

Installing on Debian

See installing tds_fdw on Debian.

Installing on openSUSE

See installing tds_fdw on openSUSE.

Installing on OSX

See installing tds_fdw on OSX.

Installing on Alpine (and Docker)

See installing tds_fdw on Alpine.

Usage

Foreign server

See creating a foreign server.

Foreign table

See creating a foreign table.

User mapping

See creating a user mapping.

Foreign schema

See importing a foreign schema.

Variables

See variables.

`EXPLAIN`

`EXPLAIN (VERBOSE)` will show the query issued on the remote system.

Notes about character sets/encoding

1. If you get an error like this with MS SQL Server when working with Unicode data:

   > NOTICE: DB-Library notice: Msg #: 4004, Msg state: 1, Msg: Unicode data in a Unicode-only collation or ntext data cannot be sent to clients using DB-Library (such as ISQL) or ODBC version 3.7 or earlier., Server: PILLIUM SQLEXPRESS, Process: , Line: 1, Level: 16
   > ERROR: DB-Library error: DB #: 4004, DB Msg: General SQL Server error: Check messages from the SQL Server, OS #: -1, OS Msg: (null), Level: 16

   and Choosing a TDS protocol version.

2. Although many newer versions of the TDS protocol will only use USC-2 to communicate with the server, FreeTDS converts the UCS-2 to the client character set of your choice. To set the client character set, you can set *client charset* in *freetds.conf*. See

Encrypted connections to MSSQL

Support

If you find any bugs, or you would like to request enhancements, please submit your comments on the project's GitHub Issues page.

Additionally, I do subscribe to several PostgreSQL mailing lists including *pgsql-general* and *pgsql-hackers*. If tds_fdw is mentioned in an email sent to one of those lists, I typically see it.

Debugging

See Debugging

TDS Foreign data wrapper

Installing on Alpine Linux

This document will show how to install tds_fdw on Alpine Linux 3.10.3. Other Alpine Linux distributions should be similar.

Install FreeTDS and build dependencies

The TDS foreign data wrapper requires a library that implements the DB-Library interface, such as FreeTDS.

```
apk add --update freetds-dev
```

Some other dependencies are also needed to install PostgreSQL and then compile tds_fdw:

```
apk add gcc libc-dev make
```

In case you will get `fatal error: stdio.h: No such file or directory` later on (on `make USE_PGXS=1`) - installing `musl-dev` migth help (https://stackoverflow.com/questions/42366739/cant-find-stdio-h-in-alpine-linux):

```
apk add musl-dev
```

Install PostgreSQL

If you need to install PostgreSQL, do so by installing from APK. For example, to install PostgreSQL 11.6 on Alpine Linux:

```
apk add postgresql=11.6-r0 postgresql-client=11.6-r0 postgresql-dev=11.6-r0
```

In postgres-alpine docker image you will need only

```
apk add postgresql-dev
```

Install tds_fdw

Build from release package If you'd like to use one of the release packages, you can download and install them via something like the following:

```
export TDS_FDW_VERSION="2.0.3"
apk add wget
wget https://github.com/tds-fdw/tds_fdw/archive/v${TDS_FDW_VERSION}.tar.gz
tar -xvzf v${TDS_FDW_VERSION}.tar.gz
cd tds_fdw-${TDS_FDW_VERSION}/
make USE_PGXS=1
sudo make USE_PGXS=1 install
```

NOTE: If you have several PostgreSQL versions and you do not want to build for the default one, first locate where the binary for `pg_config` is, take note of the full path, and then append `PG_CONFIG=<PATH>` after `USE_PGXS=1` at the `make` commands.

Build from repository If you would rather use the current development version, you can clone and build the git repository via something like the following:

```
apk add git
git clone https://github.com/tds-fdw/tds_fdw.git
cd tds_fdw
make USE_PGXS=1
make USE_PGXS=1 install
```

NOTE: If you have several PostgreSQL versions and you do not want to build for the default one, first locate where the binary for `pg_config` is, take note of the full path, and then append `PG_CONFIG=<PATH>` after `USE_PGXS=1` at the `make` commands.

Start server If this is a fresh installation, then create the initial cluster and start the server:

```
mkdir /var/lib/postgresql/data
chmod 0700 /var/lib/postgresql/data
chown postgres. /var/lib/postgresql/data
su postgres -c 'initdb /var/lib/postgresql/data'
mkdir /run/postgresql/
chown postgres. /run/postgresql/
su postgres -c 'pg_ctl start -D /var/lib/postgresql/data "-o -c listen_addresses=\"\""'
```

Install extension

```
psql -U postgres
postgres=# CREATE EXTENSION tds_fdw;
```

Dockerfile Example This Dockerfile will build PostgreSQL 11 in Alpine Linux with tds_fdw from master branch

```
FROM library/postgres:11-alpine
RUN apk add --update freetds-dev && \
    apk add git gcc libc-dev make && \
    apk add postgresql-dev postgresql-contrib && \
    git clone https://github.com/tds-fdw/tds_fdw.git && \
    cd tds_fdw && \
    make USE_PGXS=1 && \
    make USE_PGXS=1 install && \
    apk del git gcc libc-dev make && \
    cd ..  && \
    rm -rf tds_fdw
```

You can easily adapt the Dockerfile if you want to use a release package.

This Dockerfile works just like to official PostgreSQL image, just with tds_fdw added. See Docker Hub library/postgres for details. # TDS Foreign data wrapper

Installing on Debian

This document will show how to install tds_fdw on Debian 10. Other Debian distributions should be similar.

Install FreeTDS and build dependencies

The TDS foreign data wrapper requires a library that implements the DB-Library interface, such as FreeTDS.

```
sudo apt-get update
sudo apt-get install libsybdb5 freetds-dev freetds-common
```

Some other dependencies are also needed to install PostgreSQL and then compile tds_fdw:

```
sudo apt-get install gnupg gcc make
```

Install PostgreSQL

If you need to install PostgreSQL, do so by following the apt installation directions. For example, to install PostgreSQL 11 on Debian:

```
sudo bash -c 'source /etc/os-release; echo "deb http://apt.postgresql.org/pub/repos/apt/ ${V
sudo apt-key adv --keyserver hkp://pool.sks-keyservers.net --recv-keys 0xACCC4CF8
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install postgresql-11 postgresql-client-11 postgresql-server-dev-11
```

NOTE: If you already have PostgreSQL installed on your system be sure that the package postgresql-server-dev-XX is installed too (where XX stands for your PostgreSQL version).

Install tds_fdw

Build from release package If you'd like to use one of the release packages, you can download and install them via something like the following:

```
export TDS_FDW_VERSION="2.0.3"
sudo apt-get install wget
wget https://github.com/tds-fdw/tds_fdw/archive/v${TDS_FDW_VERSION}.tar.gz
tar -xvzf v${TDS_FDW_VERSION}.tar.gz
cd tds_fdw-${TDS_FDW_VERSION}/
make USE_PGXS=1
sudo make USE_PGXS=1 install
```

NOTE: If you have several PostgreSQL versions and you do not want to build for the default one, first locate where the binary for `pg_config` is, take note of the full path, and then append `PG_CONFIG=<PATH>` after `USE_PGXS=1` at the `make` commands.

Build from repository If you would rather use the current development version, you can clone and build the git repository via something like the following:

```
sudo apt-get install git
git clone https://github.com/tds-fdw/tds_fdw.git
cd tds_fdw
make USE_PGXS=1
sudo make USE_PGXS=1 install
```

NOTE: If you have several PostgreSQL versions and you do not want to build for the default one, first locate where the binary for `pg_config` is, take note of the full path, and then append `PG_CONFIG=<PATH>` after `USE_PGXS=1` at the `make` commands.

Start server If this is a fresh installation, then start the server:

```
sudo service postgresql start
```

Install extension

```
psql -U postgres
postgres=# CREATE EXTENSION tds_fdw;
```

TDS Foreign data wrapper

Installing on openSUSE

This document will show how to install tds_fdw on openSUSE Leap 15.1. Other openSUSE and SUSE distributions should be similar.

Install FreeTDS and build dependencies

The TDS foreign data wrapper requires a library that implements the DB-Library interface, such as FreeTDS.

```
sudo zypper install freetds-devel
```

Some other dependencies are also needed to install PostgreSQL and then compile tds_fdw:

```
sudo zypper install gcc make
```

Install PostgreSQL

If you need to install PostgreSQL, for example, 10.X:

```
sudo zypper install postgresql10 postgresql10-server postgresql10-devel
```

NOTE: If you already have PostgreSQL installed on your system be sure that the package postgresqlXX-devel is installed too (where XX stands for your PostgreSQL version).

Install tds_fdw

Build from release package If you'd like to use one of the release packages, you can download and install them via something like the following:

```
export TDS_FDW_VERSION="2.0.3"
wget https://github.com/tds-fdw/tds_fdw/archive/v${TDS_FDW_VERSION}.tar.gz
tar -xvzf v${TDS_FDW_VERSION}.tar.gz
cd tds_fdw-${TDS_FDW_VERSION}/
make USE_PGXS=1
sudo make USE_PGXS=1 install
```

NOTE: If you have several PostgreSQL versions and you do not want to build for the default one, first locate where the binary for `pg_config` is, take note of the full path, and then append `PG_CONFIG=<PATH>` after `USE_PGXS=1` at the `make` commands.

Build from repository If you would rather use the current development version, you can clone and build the git repository via something like the following:

```
zypper in git
git clone https://github.com/tds-fdw/tds_fdw.git
cd tds_fdw
make USE_PGXS=1
sudo make USE_PGXS=1 install
```

NOTE: If you have several PostgreSQL versions and you do not want to build for the default one, first locate where the binary for `pg_config` is, take note of the full path, and then append `PG_CONFIG=<PATH>` after `USE_PGXS=1` at the `make` commands.

Start server If this is a fresh installation, then start the server:

```
sudo service postgresql start
```

Install extension

```
psql -U postgres
postgres=# CREATE EXTENSION tds_fdw;
```

TDS Foreign data wrapper

Installing on OSX

This document will show how to install tds_fdw on OSX using the Homebrew package manager for the required packages.

Install FreeTDS

The TDS foreign data wrapper requires a library that implements the DB-Library interface, such as FreeTDS.

```
brew install freetds
```

Note: If you install FreeTDS from another source, e.g. MacPorts, you might have to adjust the value for `TDS_INCLUDE` in the make calls below (e.g. `-I/opt/local/include/freetds` for MacPorts).

Install PostgreSQL

If you need to install PostgreSQL, do so by following the apt installation directions. For example, to install PostgreSQL 9.5 on Ubuntu:

```
brew install postgres
```

Or use Postgres.app: http://postgresapp.com/

Install tds_fdw

Build from release package If you'd like to use one of the release packages, you can download and install them via something like the following:

```
export TDS_FDW_VERSION="2.0.3"
wget https://github.com/tds-fdw/tds_fdw/archive/v${TDS_FDW_VERSION}.tar.gz
tar -xvzf v${TDS_FDW_VERSION}.tar.gz
cd tds_fdw-${TDS_FDW_VERSION}
make USE_PGXS=1 TDS_INCLUDE=-I/usr/local/include/
sudo make USE_PGXS=1 install
```

NOTE: If you have several PostgreSQL versions and you do not want to build for the default one, first locate where the binary for `pg_config` is, take note of the full path, and then append `PG_CONFIG=<PATH>` after `USE_PGXS=1` at the `make` commands.

Build from repository If you would rather use the current development version, you can clone and build the git repository via something like the following:

```
git clone https://github.com/tds-fdw/tds_fdw.git
cd tds_fdw
make USE_PGXS=1 TDS_INCLUDE=-I/usr/local/include/
sudo make USE_PGXS=1 install
```

NOTE: If you have several PostgreSQL versions and you do not want to build for the default one, first locate where the binary for `pg_config` is, take note of the full path, and then append `PG_CONFIG=<PATH>` after `USE_PGXS=1` at the `make` commands.

Start server If this is a fresh installation, then start the server:

```
brew services start postgresql
```

Or the equivalent command if you are not using Homebrew.

Install extension

```
psql -U postgres
postgres=# CREATE EXTENSION tds_fdw;
```

TDS Foreign data wrapper

Installing on RHEL and Clones such as CentOS, Rocky Linux, AlmaLinux or Oracle

This document will show how to install tds_fdw on Rocky Linux 8.5. RHEL distributions should be similar.

NOTE: For the sake of simplicity, we will use `yum` as it works as an alias for `dnf` on newer distributions.

Option A: yum/dnf (released versions)

PostgreSQL If you need to install PostgreSQL, do so by following the RHEL installation instructions.

Here is an extract of the instructions:

Only for RHEL 8 and clones such as Rocky Linux 8:

```
sudo sudo dnf -qy module disable postgresql # Not required for RHEL8 and clones
```

Install the PostgreSQL repository and packages:

```
sudo rpm -i https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-r
sudo yum install postgresql11 postgresql11-server postgresql11-libs postgresql11-devel
```

tds_fdw The PostgreSQL development team packages `tds_fdw`, but they do not provide FreeTDS.

First, install the EPEL repository:

```
sudo yum install epel-release
```

And then install `tds_fdw`:

```
sudo yum install tds_fdw11.x86_64
```

Option B: Compile tds_fdw

PostgreSQL If you need to install PostgreSQL, do so by following the RHEL installation instructions.

Here is an extract of the instructions:

Only for RHEL 8 and clones such as Rocky Linux 8:

```
sudo sudo dnf -qy module disable postgresql # Not required for RHEL8 and clones
```

Install the PostgreSQL repository and packages:

```
sudo rpm -i https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-r
sudo yum install postgresql11 postgresql11-server postgresql11-libs postgresql11-devel
```

Install FreeTDS devel and build dependencies The TDS foreign data wrapper requires a library that implements the DB-Library interface, such as FreeTDS.

NOTE: In CentOS, you need the EPEL repository installed to install FreeTDS

```
sudo yum install epel-release
sudo yum install freetds-devel
```

IMPORTANT: CentOS7/Oracle7 and PostgreSQL >= 11

When using the official PostgreSQL packages from postgresql.org, JIT with bitcode is enabled by default and will require llvm5 and `clang` from LLVM5 installed at `/opt/rh/llvm-toolset-7/root/usr/bin/clang` to be able to compile.

You have LLVM5 at the EPEL CentOS7 repository, but not LLVM7, so you will need install the CentOS Software collections.

You can easily do it with the following commands:

```
sudo yum install centos-release-scl
```

Some other dependencies are also needed to install PostgreSQL and then compile tds_fdw:

```
sudo yum install gcc make wget
```

Build from release package If you'd like to use one of the release packages, you can download and install them via something like the following:

```
export TDS_FDW_VERSION="2.0.3"
wget https://github.com/tds-fdw/tds_fdw/archive/v${TDS_FDW_VERSION}.tar.gz
tar -xvzf v${TDS_FDW_VERSION}.tar.gz
cd tds_fdw-${TDS_FDW_VERSION}
make USE_PGXS=1 PG_CONFIG=/usr/pgsql-11/bin/pg_config
sudo make USE_PGXS=1 PG_CONFIG=/usr/pgsql-11/bin/pg_config install
```

NOTE: If you have several PostgreSQL versions and you do not want to build for the default one, first locate where the binary for `pg_config` is, take note of the full path, then adjust `PG_CONFIG` accordingly.

Build from repository If you would rather use the current development version, you can clone and build the git repository via something like the following:

```
yum install git
git clone https://github.com/tds-fdw/tds_fdw.git
cd tds_fdw
make USE_PGXS=1 PG_CONFIG=/usr/pgsql-11/bin/pg_config
sudo make USE_PGXS=1 PG_CONFIG=/usr/pgsql-11/bin/pg_config install
```

NOTE: If you have several PostgreSQL versions and you do not want to build for the default one, first locate where the binary for `pg_config` is, take note of the full path, then adjust `PG_CONFIG` accordingly.

Final steps

Start server If this is a fresh installation, then initialize the data directory and start the server:

```
sudo /usr/pgsql-11/bin/postgresql11-setup initdb
sudo systemctl enable postgresql-11.service
sudo systemctl start postgresql-11.service
```

Install extension

```
/usr/pgsql-11/bin/psql -U postgres
postgres=# CREATE EXTENSION tds_fdw;
```

TDS Foreign data wrapper

Installing on Ubuntu

This document will show how to install tds_fdw on Ubuntu 18.04. Other Ubuntu distributions should be similar.

Install FreeTDS and build dependencies

The TDS foreign data wrapper requires a library that implements the DB-Library interface, such as FreeTDS.

```
sudo apt-get update
sudo apt-get install libsybdb5 freetds-dev freetds-common
```

Some other dependencies are also needed to install PostgreSQL and then compile tds_fdw:

```
sudo apt-get install gnupg gcc make
```

Install PostgreSQL

If you need to install PostgreSQL, do so by following the apt installation directions. For example, to install PostgreSQL 11 on Ubuntu:

```
sudo bash -c 'source /etc/os-release; echo "deb http://apt.postgresql.org/pub/repos/apt/ ${
sudo apt-key adv --keyserver hkp://pool.sks-keyservers.net --recv-keys 0xACCC4CF8
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install postgresql-11 postgresql-client-11 postgresql-server-dev-11
```

NOTE: If you already have PostgreSQL installed on your system be sure that the package postgresql-server-dev-XX is installed too (where XX stands for your PostgreSQL version).

Install tds_fdw

Build from release package If you'd like to use one of the release packages, you can download and install them via something like the following:

```
export TDS_FDW_VERSION="2.0.3"
sudo apt-get install wget
wget https://github.com/tds-fdw/tds_fdw/archive/v${TDS_FDW_VERSION}.tar.gz
tar -xvzf v${TDS_FDW_VERSION}.tar.gz
cd tds_fdw-${TDS_FDW_VERSION}/
make USE_PGXS=1
sudo make USE_PGXS=1 install
```

NOTE: If you have several PostgreSQL versions and you do not want to build for the default one, first locate where the binary for `pg_config` is, take note of the full path, and then append `PG_CONFIG=<PATH>` after `USE_PGXS=1` at the `make` commands.

Build from repository If you would rather use the current development version, you can clone and build the git repository via something like the following:

```
sudo apt-get install git
git clone https://github.com/tds-fdw/tds_fdw.git
cd tds_fdw
make USE_PGXS=1
sudo make USE_PGXS=1 install
```

NOTE: If you have several PostgreSQL versions and you do not want to build for the default one, first locate where the binary for `pg_config` is, take note of the full path, and then append `PG_CONFIG=<PATH>` after `USE_PGXS=1` at the `make` commands.

Start server If this is a fresh installation, then start the server:

```
sudo service postgresql start
```

Install extension

```
psql -U postgres
postgres=# CREATE EXTENSION tds_fdw;
```

TDS Foreign data wrapper

Creating a User Mapping

Options

User mapping parameters accepted:

- *username*

Required: Yes

The username of the account on the foreign server.

- *password*

Required: Yes

The password of the account on the foreign server.

Example

```
CREATE USER MAPPING FOR postgres
    SERVER mssql_svr
    OPTIONS (username 'sa', password '');
```
```# TDS Foreign data wrapper


## Variables

### Available Variables

* *tds_fdw.show_before_row_memory_stats* - print memory context stats to the PostgreSQL log

* *tds_fdw.show_after_row_memory_stats* - print memory context stats to the PostgreSQL log a

* *tds_fdw.show_finished_memory_stats* - print memory context stats to the PostgreSQL log wh

### Setting Variables

To set a variable, use the [SET command](https://www.postgresql.org/docs/12/sql-set.html). i

postgres=# SET tds_fdw.show_finished_memory_stats=1; SET “‘
```