MongoDB Foreign Data Wrapper Guide

Version 5.2.9

Built at 2021-07-20T15:05:03

1	What's New	3
2	Requirements Overview	3
3	Architecture Overview	3
4	Installing the MongoDB Foreign Data Wrapper	4
5	Updating the MongoDB Foreign Data Wrapper	8
6	Features of the MongoDB Foreign Data Wrapper	9
7	Configuring the MongoDB Foreign Data Wrapper	11
8	Example: Using the MongoDB Foreign Data Wrapper	19
9	Identifying the MongoDB Foreign Data Wrapper Version	21
10	Limitations	21
11	Uninstalling the MongoDB Foreign Data Wrapper	22

1 What's New

The following features are added to create MongoDB Foreign Data Wrapper `5.2.9`:

- Updated mongo-c-driver to 1.17.3
- Updated json-c to 0.15

2 Requirements Overview

Supported Versions

The MongoDB Foreign Data Wrapper is certified with EDB Postgres Advanced Server 9.6 and above.

Supported Platforms

The MongoDB Foreign Data Wrapper is supported on the following platforms:

Linux x86-64

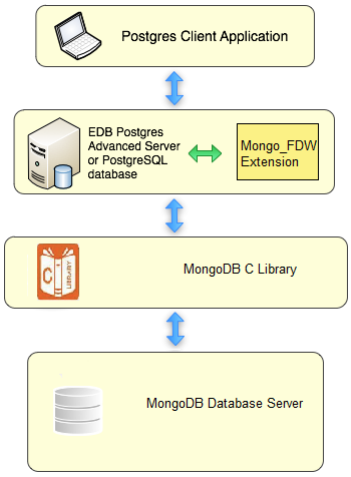
- RHEL 8.x/7.x
- CentOS 8.x/7.x
- OL 8.x/7.x
- Ubuntu 20.04/18.04 LTS
- Debian 10.x/9.x

Linux on IBM Power8/9 (LE)

- RHEL 7.x

3 Architecture Overview

The MongoDB data wrapper provides an interface between a MongoDB server and a Postgres database. It transforms a Postgres statement (`SELECT`/`INSERT`/`DELETE`/`UPDATE`) into a query that is understood by the MongoDB database.



4 Installing the MongoDB Foreign Data Wrapper

The MongoDB Foreign Data Wrapper can be installed with an RPM package. During the installation process, the installer will satisfy software prerequisites. If yum encounters a dependency that it cannot resolve, it will provide a list of the required dependencies that you must manually resolve.

Installing the MongoDB Foreign Data Wrapper using an RPM Package

You can install the MongoDB Foreign Data Wrapper using an RPM package on the following platforms:

- RHEL or CentOS 7 PPCLE
- RHEL 7
- RHEL 8
- CentOS 7
- CentOS 8

On RHEL or CentOS 7 PPCLE

1. Use the following command to create a configuration file and install Advance Toolchain:

```
rpm --import
https://public.dhe.ibm.com/software/server/POWER/Linux/toolchain/at/redhat/RHEL7/gp
-pubkey-6976a827-5164221b

cat > /etc/yum.repos.d/advance-toolchain.repo <<EOF
# Begin of configuration file
[advance-toolchain]
name=Advance Toolchain IBM FTP
baseurl=https://public.dhe.ibm.com/software/server/POWER/Linux/toolchain/at/redhat/
7
failovermethod=priority
enabled=1
```

```
gpgcheck=1
gpgkey=ftp://public.dhe.ibm.com/software/server/POWER/Linux/toolchain/at/redhat/RHI
-pubkey-6976a827-5164221b
# End of configuration file
EOF
```

2. Install the EDB repository:

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

3. Replace 'USERNAME:PASSWORD' below with your username and password for the EDB repositories:

```
sed -i "s@<username>:<password>@USERNAME:PASSWORD@" /etc/yum.repos.d/edb.repo
```

4. Install the EPEL repository:

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

5. On RHEL 7 PPCLE, enable the additional repositories to resolve EPEL dependencies:

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

6. Install the selected package:

```
dnf install edb-as<xx>-mongo_fdw
```

where `xx` is the server version number.

On RHEL 7

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

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

2. Replace 'USERNAME:PASSWORD' below with your username and password for the EDB repositories:

```
sed -i "s@<username>:<password>@USERNAME:PASSWORD@" /etc/yum.repos.d/edb.repo
```

3. Install the EPEL repository:

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

4. Enable the additional repositories to resolve dependencies:

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

5. Install the selected package:

```
dnf install edb-as<xx>-mongo_fdw
```

where `xx` is the server version number.

On RHEL 8

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

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

2. Replace 'USERNAME:PASSWORD' below with your username and password for the EDB repositories:

```
sed -i "s@<username>:<password>@USERNAME:PASSWORD@" /etc/yum.repos.d/edb.repo
```

3. Install the EPEL repository:

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

4. Enable the additional repositories to resolve dependencies:

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

5. Disable the built-in PostgreSQL module:

```
dnf -qy module disable postgresql
```

6. Install the selected package:

```
dnf install edb-as<xx>-mongo_fdw
```

where `xx` is the server version number.

On CentOS 7

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

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

2. Replace 'USERNAME:PASSWORD' below with your username and password for the EDB repositories:

```
sed -i "s@<username>:<password>@USERNAME:PASSWORD@" /etc/yum.repos.d/edb.repo
```

3. Install the EPEL repository:

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

4. Install the selected package:

```
dnf install edb-as<xx>-mongo_fdw
```

where `xx` is the server version number.

On CentOS 8

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

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

2. Replace 'USERNAME:PASSWORD' below with your username and password for the EDB repositories:

```
sed -i "s@<username>:<password>@USERNAME:PASSWORD@" /etc/yum.repos.d/edb.repo
```

3. Install the EPEL repository:

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

4. Enable the additional repositories to resolve dependencies:

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

5. Disable the built-in PostgreSQL module:

```
dnf -qy module disable postgresql
```

6. Install the selected package:

```
dnf install edb-as<xx>-mongo_fdw
```

where `xx` is the server version number.

Installing the MongoDB Foreign Data Wrapper on a Debian or Ubuntu Host

To install the MongoDB Foreign Data Wrapper on a Debian or Ubuntu host, you must have credentials that allow access to the EDB repository. To request credentials for the repository, visit the EDB website.

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

1. Assume superuser privileges:

```
sudo su -
```

2. Configure the EDB repository:

   On Debian 9 and Ubuntu:

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

   On Debian 10:

   1. Set up the EDB repository:

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

```
cs).list'
```

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

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

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

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

4. Add the EDB signing key:

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

5. Update the repository metadata:

```
apt-get update
```

6. Install the Debian package:

```
apt-get install edb-as<xx>-mongo-fdw
```

where `xx` is the server version number.

5 Updating the MongoDB Foreign Data Wrapper

Updating an RPM Installation

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

- On RHEL or CentOS 7:

```
yum upgrade edb-repo
```

- On RHEL or CentOS 7 on PPCLE:

```
yum upgrade edb-repo
```

- On RHEL or CentOS 8:

```
dnf upgrade edb-repo
```

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

- On RHEL or CentOS 7:

```
yum upgrade edb-as<xx>-mongo_fdw edb-libmongoc-libs
```

- On RHEL or CentOS 7 on PPCLE:

  ```
  yum upgrade edb-as<xx>-mongo_fdw edb-libmongoc-libs
  ```

- On RHEL or CentOS 8:

  ```
  dnf upgrade edb-as<xx>-mongo_fdw
  ```

  where `xx` is the server version number.

Updating MongoDB Foreign Data Wrapper on a Debian or Ubuntu Host

To update MongoDB Foreign Data Wrapper on a Debian or Ubuntu Host, use the following command:

```
apt-get --only-upgrade install edb-as<xx>-mongo-fdw edb-libmongoc
```

where `xx` is the server version number.

---

6 Features of the MongoDB Foreign Data Wrapper

The key features of the MongoDB Foreign Data Wrapper are listed below:

Writable FDW

The MongoDB Foreign Data Wrapper allows you to modify data on a MongoDB server. Users can `INSERT`, `UPDATE` and `DELETE` data in the remote MongoDB collections by inserting, updating and deleting data locally in foreign tables.

See also:

- Example: Using the MongoDB Foreign Data Wrapper
- Data Type Mappings

WHERE Clause Push-down

MongoDB Foreign Data Wrapper allows the push-down of the `WHERE` clause only when clauses include the comparison expressions that have a column and a constant as arguments. `WHERE` clause push-down is not supported where the constant is an array.

Connection Pooling

The MongoDB Foreign Data Wrapper establishes a connection to a foreign server during the first query that uses a

foreign table associated with the foreign server. This connection is kept and reused for subsequent queries in the same session.

Automated Cleanup

The MongoDB Foreign Data Wrapper allows the cleanup of foreign tables in a single operation using the `DROP EXTENSION` command. This feature is especially useful when a foreign table has been created for a temporary purpose. The syntax of a `DROP EXTENSION` command is:

```
DROP EXTENSION mongo_fdw CASCADE;
```

For more information, see DROP EXTENSION.

Full Document Retrieval

This feature allows you to retrieve documents along with all their fields from collection without any knowledge of the fields in the BSON document available in MongoDB's collection. Those retrieved documents are in JSON format.

You can retrieve all available fields in a collection residing in MongoDB Foreign Data Wrapper as explained in the following example:

Example:

```
> db.warehouse.find();
{ "_id" : ObjectId("58a1ebbaf543ec0b90545859"), "warehouse_id" : 1,
"warehouse_name" : "UPS", "warehouse_created" : ISODate("2014-12-12T07:12:10Z") }
{ "_id" : ObjectId("58a1ebbaf543ec0b9054585a"), "warehouse_id" : 2,
"warehouse_name" : "Laptop", "warehouse_created" : ISODate("2015-11-11T08:13:10Z")
}
```

Steps for retrieving the document:

1. Create foreign table with a column name `__doc`. The type of the column could be json, jsonb, text, or varchar.

```
CREATE FOREIGN TABLE test_json(__doc json) SERVER mongo_server OPTIONS (database
'testdb', collection 'warehouse');
```

2. Retrieve the document.

```
SELECT * FROM test_json ORDER BY __doc::text COLLATE "C";
```

The output:

```
edb=#SELECT * FROM test_json ORDER BY __doc::text COLLATE "C";
                                                                        __doc
------------------------------------------------------------------------------
------------------------------------------------------------------------
{ "_id" : { "$oid" : "58a1ebbaf543ec0b90545859" }, "warehouse_id" : 1,
"warehouse_name" : "UPS", "warehouse_created" : { "$date" : 1418368330000 } }
{ "_id" : { "$oid" : "58a1ebbaf543ec0b9054585a" }, "warehouse_id" : 2,
"warehouse_name" : "Laptop", "warehouse_created" : { "$date" : 1447229590000 } }
(2 rows)
```

7 Configuring the MongoDB Foreign Data Wrapper

Before using the MongoDB Foreign Data Wrapper, you must:

1. Use the CREATE EXTENSION command to create the MongoDB Foreign Data Wrapper extension on the Postgres host.
2. Use the CREATE SERVER command to define a connection to the MongoDB server.
3. Use the CREATE USER MAPPING command to define a mapping that associates a Postgres role with the server.
4. Use the CREATE FOREIGN TABLE command to define a table in the Postgres database that corresponds to a database that resides on the MongoDB cluster.

CREATE EXTENSION

Use the `CREATE EXTENSION` command to create the `mongo_fdw` extension. To invoke the command, use your client of choice (for example, psql) to connect to the Postgres database from which you will be querying the MongoDB server, and invoke the command:

```
CREATE EXTENSION [IF NOT EXISTS] mongo_fdw [WITH] [SCHEMA schema_name];
```

Parameters

`IF NOT EXISTS`

Include the `IF NOT EXISTS` clause to instruct the server to issue a notice instead of throwing an error if an extension with the same name already exists.

`schema_name`

Optionally specify the name of the schema in which to install the extension's objects.

Example

The following command installs the MongoDB foreign data wrapper:

```
CREATE EXTENSION mongo_fdw;
```

For more information about using the foreign data wrapper `CREATE EXTENSION` command, see:

https://www.postgresql.org/docs/current/static/sql-createextension.html.

CREATE SERVER

Use the `CREATE SERVER` command to define a connection to a foreign server. The syntax is:

```
CREATE SERVER server_name FOREIGN DATA WRAPPER mongo_fdw
    [OPTIONS (option 'value' [, ...])]
```

The role that defines the server is the owner of the server; use the `ALTER SERVER` command to reassign ownership of a foreign server. To create a foreign server, you must have `USAGE` privilege on the foreign-data wrapper specified in the `CREATE SERVER` command.

Parameters

`server_name`

Use `server_name` to specify a name for the foreign server. The server name must be unique within the database.

`FOREIGN_DATA_WRAPPER`

Include the `FOREIGN_DATA_WRAPPER` clause to specify that the server should use the `mongo_fdw` foreign data wrapper when connecting to the cluster.

`OPTIONS`

Use the `OPTIONS` clause of the `CREATE SERVER` command to specify connection information for the foreign server object. You can include:

Option	Description
address	The address or hostname of the Mongo server. The default value is `127.0.0.1`.
port	The port number of the Mongo Server. Valid range is 0 to 65535. The default value is `27017`.
authentication_database	The database against which user will be authenticated. This option is only valid with password based authentication.
ssl	Requests an authenticated, encrypted SSL connection. By default, the value is set to `false`. Set the value to `true` to enable ssl. See http://mongoc.org/libmongoc/current/mongoc_ssl_opt_t.html to understand the options.
pem_file	SSL option.
pem_pwd	SSL option.
ca_file	SSL option.
ca_dir	SSL option.
crl_file	SSL option.
weak_cert_validation	SSL option.

Example

The following command creates a foreign server named `mongo_server` that uses the `mongo_fdw` foreign data wrapper to connect to a host with an IP address of `127.0.0.1`:

```
CREATE SERVER mongo_server FOREIGN DATA WRAPPER mongo_fdw OPTIONS (host
'127.0.0.1', port '27017');
```

The foreign server uses the default port (`27017`) for the connection to the client on the MongoDB cluster.

For more information about using the `CREATE SERVER` command, see:

https://www.postgresql.org/docs/current/static/sql-createserver.html

CREATE USER MAPPING

Use the `CREATE USER MAPPING` command to define a mapping that associates a Postgres role with a foreign server:

```
CREATE USER MAPPING FOR role_name SERVER server_name
```

```
       [OPTIONS (option 'value' [, ...])];
```

You must be the owner of the foreign server to create a user mapping for that server.

Parameters

`role_name`

Use `role_name` to specify the role that will be associated with the foreign server.

`server_name`

Use `server_name` to specify the name of the server that defines a connection to the MongoDB cluster.

`OPTIONS`

Use the `OPTIONS` clause to specify connection information for the foreign server.

`username`: the name of the user on the MongoDB server.

`password`: the password associated with the username.

Example

The following command creates a user mapping for a role named `enterprisedb`; the mapping is associated with a server named `mongo_server`:

```
CREATE USER MAPPING FOR enterprisedb SERVER mongo_server;
```

If the database host uses secure authentication, provide connection credentials when creating the user mapping:

```
CREATE USER MAPPING FOR enterprisedb SERVER mongo_server OPTIONS (username
'mongo_user', password 'mongo_pass');
```

The command creates a user mapping for a role named `enterprisedb` that is associated with a server named `mongo_server`. When connecting to the MongoDB server, the server will authenticate as `mongo_user`, and provide a password of `mongo_pass`.

For detailed information about the `CREATE USER MAPPING` command, see:

https://www.postgresql.org/docs/current/static/sql-createusermapping.html

CREATE FOREIGN TABLE

A foreign table is a pointer to a table that resides on the MongoDB host. Before creating a foreign table definition on the Postgres server, connect to the MongoDB server and create a collection; the columns in the table will map to columns in a table on the Postgres server. Then, use the `CREATE FOREIGN TABLE` command to define a table on the Postgres server with columns that correspond to the collection that resides on the MongoDB host. The syntax is:

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name ( [
  { column_name data_type [ OPTIONS ( option 'value' [, ... ] ) ] [ COLLATE
collation ] [ column_constraint [ ... ] ]
    | table_constraint }
    [, ... ]
] )
```

```
[ INHERITS ( parent_table [, ... ] ) ]
  SERVER server_name [ OPTIONS ( option 'value' [, ... ] ) ]
```

where `column_constraint` is:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL | NULL | CHECK (expr) [ NO INHERIT ] | DEFAULT default_expr }
```

and `table_constraint` is:

```
[ CONSTRAINT constraint_name ] CHECK (expr) [ NO INHERIT ]
```

Parameters

`table_name`

Specify the name of the foreign table; include a schema name to specify the schema in which the foreign table should reside.

`IF NOT EXISTS`

Include the `IF NOT EXISTS` clause to instruct the server to not throw an error if a table with the same name already exists; if a table with the same name exists, the server will issue a notice.

`column_name`

Specify the name of a column in the new table; each column should correspond to a column described on the MongoDB server.

`data_type`

Specify the data type of the column; when possible, specify the same data type for each column on the Postgres server and the MongoDB server. If a data type with the same name is not available, the Postgres server will attempt to cast the data type to a type compatible with the MongoDB server. If the server cannot identify a compatible data type, it will return an error.

`COLLATE collation`

Include the `COLLATE` clause to assign a collation to the column; if not specified, the column data type's default collation is used.

`INHERITS (parent_table [, ... ])`

Include the `INHERITS` clause to specify a list of tables from which the new foreign table automatically inherits all columns. Parent tables can be plain tables or foreign tables.

`CONSTRAINT constraint_name`

Specify an optional name for a column or table constraint; if not specified, the server will generate a constraint name.

`NOT NULL`

Include the `NOT NULL` keywords to indicate that the column is not allowed to contain null values.

`NULL`

Include the `NULL` keywords to indicate that the column is allowed to contain null values. This is the default.

`CHECK (expr) [NO INHERIT]`

Use the `CHECK` clause to specify an expression that produces a Boolean result that each row in the table must satisfy. A check constraint specified as a column constraint should reference that column's value only, while an expression appearing in a table constraint can reference multiple columns.

A `CHECK` expression cannot contain subqueries or refer to variables other than columns of the current row.

Include the `NO INHERIT` keywords to specify that a constraint should not propagate to child tables.

`DEFAULT default_expr`

Include the `DEFAULT` clause to specify a default data value for the column whose column definition it appears within. The data type of the default expression must match the data type of the column.

`SERVER server_name [OPTIONS (option 'value' [, ... ] ) ]`

To create a foreign table that will allow you to query a table that resides on a MongoDB file system, include the `SERVER` clause and specify the `server_name` of the foreign server that uses the MongoDB data adapter.

Use the `OPTIONS` clause to specify the following `options` and their corresponding values:

option	value
database	The name of the database to query. The default value is `test`.
collection	The name of the collection to query. The default value is the foreign table name.

Example

To use data that is stored on MongoDB server, you must create a table on the Postgres host that maps the columns of a MongoDB collection to the columns of a Postgres table. For example, for a MongoDB collection with the following definition:

```
db.warehouse.find
(
        {
                "warehouse_id" : 1
        }
).pretty()
{
        "_id" : ObjectId("53720b1904864dc1f5a571a0"),
        "warehouse_id" : 1,
        "warehouse_name" : "UPS",
        "warehouse_created" : ISODate("2014-12-12T07:12:10Z")
}
```

You should execute a command on the Postgres server that creates a comparable table on the Postgres server:

```
CREATE FOREIGN TABLE warehouse
(
 _id               NAME,
 warehouse_id      INT,
 warehouse_name    TEXT,
 warehouse_created TIMESTAMPZ
)
SERVER mongo_server
```

```
OPTIONS (database 'db', collection 'warehouse');
```

The first column of the table must be `_id` of the type `name`.

Include the `SERVER` clause to specify the name of the database stored on the MongoDB server and the name of the table (`warehouse`) that corresponds to the table on the Postgres server.

For more information about using the `CREATE FOREIGN TABLE` command, see:

https://www.postgresql.org/docs/current/static/sql-createforeigntable.html

!!! Note MongoDB foreign data wrapper supports the write capability feature.

Data Type Mappings

When using the foreign data wrapper, you must create a table on the Postgres server that mirrors the table that resides on the MongoDB server. The MongoDB data wrapper will automatically convert the following MongoDB data types to the target Postgres type:

MongoDB (BSON Type)	Postgres
ARRAY JSON BOOL BOOL	
BINARY BYTE	A
DATE_TIME DATE DOCUMENT JSON	/TIMESTAMP/TIMESTAMPTZ
DOUBLE FLOA	T/FLOAT4/FLOAT8/DOUBLE PRECISION/NUMERIC
INT32 SMAL	LINT/INT2/INT/INTEGER/INT4
INT64 BIGI OID NAME	NT/INT8
UTF8 BPCH	AR/VARCHAR/CHARCTER VARYING/TEXT

DROP EXTENSION

Use the `DROP EXTENSION` command to remove an extension. To invoke the command, use your client of choice (for example, psql) to connect to the Postgres database from which you will be dropping the MongoDB server, and run the command:

```
DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ];
```

Parameters

`IF EXISTS`

Include the `IF EXISTS` clause to instruct the server to issue a notice instead of throwing an error if an extension with the specified name doesn't exists.

`name`

Specify the name of the installed extension. It is optional.

`CASCADE`

Automatically drop objects that depend on the extension. It drops all the other dependent objects too.

`RESTRICT`

Do not allow to drop extension if any objects, other than its member objects and extensions listed in the same DROP command are dependent on it.

Example

The following command removes the extension from the existing database:

```
DROP EXTENSION mongo_fdw;
```

For more information about using the foreign data wrapper `DROP EXTENSION` command, see:

https://www.postgresql.org/docs/current/sql-dropextension.html.

DROP SERVER

Use the `DROP SERVER` command to remove a connection to a foreign server. The syntax is:

```
DROP SERVER [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

The role that drops the server is the owner of the server; use the `ALTER SERVER` command to reassign ownership of a foreign server. To drop a foreign server, you must have `USAGE` privilege on the foreign-data wrapper specified in the `DROP SERVER` command.

Parameters

`IF EXISTS`

Include the `IF EXISTS` clause to instruct the server to issue a notice instead of throwing an error if a server with the specified name doesn't exists.

`name`

Specify the name of the installed server. It is optional.

`CASCADE`

Automatically drop objects that depend on the server. It should drop all the other dependent objects too.

`RESTRICT`

Do not allow to drop the server if any objects are dependent on it.

Example

The following command removes a foreign server named `mongo_server`:

```
DROP SERVER mongo_server;
```

For more information about using the `DROP SERVER` command, see:

https://www.postgresql.org/docs/current/sql-dropserver.html

DROP USER MAPPING

Use the `DROP USER MAPPING` command to remove a mapping that associates a Postgres role with a foreign server. You must be the owner of the foreign server to remove a user mapping for that server.

```
DROP USER MAPPING [ IF EXISTS ] FOR { user_name | USER | CURRENT_USER | PUBLIC }
SERVER server_name;
```

Parameters

`IF EXISTS`

Include the `IF EXISTS` clause to instruct the server to issue a notice instead of throwing an error if the user mapping doesn't exist.

`user_name`

Specify the user name of the mapping.

`server_name`

Specify the name of the server that defines a connection to the MongoDB cluster.

Example

The following command drops a user mapping for a role named `enterprisedb`; the mapping is associated with a server named `mongo_server`:

```
DROP USER MAPPING FOR enterprisedb SERVER mongo_server;
```

For detailed information about the `DROP USER MAPPING` command, see:

https://www.postgresql.org/docs/current/static/sql-dropusermapping.html

DROP FOREIGN TABLE

A foreign table is a pointer to a table that resides on the MongoDB host. Use the `DROP FOREIGN TABLE` command to remove a foreign table. Only the owner of the foreign table can drop it.

```
DROP FOREIGN TABLE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

Parameters

`IF EXISTS`

Include the `IF EXISTS` clause to instruct the server to issue a notice instead of throwing an error if the foreign table with the specified name doesn't exists.

`name`

Specify the name of the foreign table.

`CASCADE`

Automatically drop objects that depend on the foreign table. It should drop all the other dependent objects too.

`RESTRICT`

Do not allow to drop foreign table if any objects are dependent on it.

Example

```
DROP FOREIGN TABLE warehouse;
```

For more information about using the `DROP FOREIGN TABLE` command, see:

https://www.postgresql.org/docs/current/sql-dropforeigntable.html

8 Example: Using the MongoDB Foreign Data Wrapper

Before using the MongoDB foreign data wrapper, you must connect to your database with a client application. The following examples demonstrate using the wrapper with the psql client. After connecting to psql, you can follow the steps in the example below:

```
-- load extension first time after install
CREATE EXTENSION mongo_fdw;

-- create server object
CREATE SERVER mongo_server
          FOREIGN DATA WRAPPER mongo_fdw
          OPTIONS (address '127.0.0.1', port '27017');

-- create user mapping
CREATE USER MAPPING FOR enterprisedb
          SERVER mongo_server
          OPTIONS (username 'mongo_user', password 'mongo_pass');

-- create foreign table
CREATE FOREIGN TABLE warehouse
        (
            _id name,
            warehouse_id int,
            warehouse_name text,
            warehouse_created timestamptz
        )
          SERVER mongo_server
          OPTIONS (database 'db', collection 'warehouse');

-- Note: first column of the table must be "_id" of type "name".

-- select from table
SELECT * FROM warehouse WHERE warehouse_id = 1;
           _id            | warehouse_id | warehouse_name |     warehouse_created
--------------------------+--------------+----------------+-----------------------
----
 53720b1904864dc1f5a571a0 |            1 | UPS            | 2014-12-12
```

```
12:42:10+05:30
(1 row)

db.warehouse.find
(
        {
            "warehouse_id" : 1
        }
).pretty()
{
        "_id" : ObjectId("53720b1904864dc1f5a571a0"),
        "warehouse_id" : 1,
        "warehouse_name" : "UPS",
        "warehouse_created" : ISODate("2014-12-12T07:12:10Z")
}

-- insert row in table
INSERT INTO warehouse VALUES (0, 2, 'Laptop', '2015-11-11T08:13:10Z');

db.warehouse.insert
(
        {
            "warehouse_id" : NumberInt(2),
            "warehouse_name" : "Laptop",
            "warehouse_created" : ISODate("2015-11-11T08:13:10Z")
        }
)

-- delete row from table
DELETE FROM warehouse WHERE warehouse_id = 2;

db.warehouse.remove
(
        {
            "warehouse_id" : 2
        }
)

-- update a row of table
UPDATE warehouse SET warehouse_name = 'UPS_NEW' WHERE warehouse_id = 1;

db.warehouse.update
(
        {
            "warehouse_id" : 1
        },
        {
            "warehouse_id" : 1,
            "warehouse_name" : "UPS_NEW",
            "warehouse_created" : ISODate("2014-12-12T07:12:10Z")
        }
)

-- explain a table
EXPLAIN SELECT * FROM warehouse WHERE warehouse_id = 1;
```

```
                              QUERY PLAN
-----------------------------------------------------------------------
 Foreign Scan on warehouse  (cost=0.00..0.00 rows=1000 width=84)
   Filter: (warehouse_id = 1)
   Foreign Namespace: db.warehouse
(3 rows)

-- collect data distribution statistics
ANALYZE warehouse;

-- drop foreign table
DROP FOREIGN TABLE warehouse;

-- drop user mapping
DROP USER MAPPING FOR enterprisedb SERVER mongo_server;

-- drop server
DROP SERVER mongo_server;
```

9 Identifying the MongoDB Foreign Data Wrapper Version

The MongoDB Foreign Data Wrapper includes a function that you can use to identify the currently installed version of the `.so` file for the data wrapper. To use the function, connect to the Postgres server, and enter:

```
SELECT mongo_fdw_version();
```

The function returns the version number:

```
mongo_fdw_version
-----------------
<xxxxx>
```

10 Limitations

The following limitations apply to MongoDB Foreign Data Wrapper:

- If the BSON document key contains uppercase letters or occurs within a nested document, MongoDB Foreign Data Wrapper requires the corresponding column names to be declared in double quotes.
- PostgreSQL limits column names to 63 characters by default. You can increase the `NAMEDATALEN` constant in `src/include/pg_config_manual.h`, compile, and re-install when column names extend beyond 63 characters.
- MongoDB Foreign Data Wrapper errors out on BSON field which is not listed in the known types (For example: byte, arrays). It throws an error: `Cannot convert BSON type to column type`.

11 Uninstalling the MongoDB Foreign Data Wrapper

Uninstalling an RPM Package

You can use the `yum remove` or `dnf remove` command to remove a package installed by `yum` or `dnf`. To remove a package, open a terminal window, assume superuser privileges, and enter the command:

- On RHEL or CentOS 7:

  ```
  yum remove edb-as<xx>-mongo_fdw
  ```

- On RHEL or CentOS 8:

  ```
  dnf remove edb-as<xx>-mongo_fdw
  ```

Where `xx` is the server version number.

Uninstalling MongoDB Foreign Data Wrapper on a Debian or Ubuntu Host

- To uninstall MongoDB Foreign Data Wrapper on a Debian or Ubuntu host, invoke the following command.

  ```
  apt-get remove edb-as<xx>-mongo-fdw
  ```

Where `xx` is the server version number.