MongoDB Foreign Data Wrapper Guide

Version 5.2.8

Built at 2021-01-19T10:46:26

1 What's New

The following features are added to create MongoDB Foreign Data Wrapper 5.2.8:

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

2 Requirements Overview

Supported Versions

The MongoDB Foreign Data Wrapper is certified with EDB Postgres Advanced Server 9.5 and above.

Supported Platforms

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

Linux x86-64

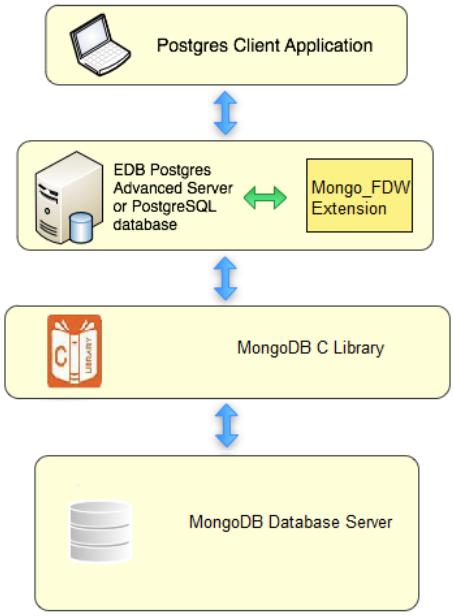
- RHEL 8.x/7.x
- CentOS 8.x/7.x
- OEL 8.x/7.x
- Ubuntu 20.04/18.04 LTS
- Debian 10.x/9.x

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.

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 7
- RHEL 8
- CentOS 7
- CentOS 8

On RHEL 7

Before installing the MongoDB Foreign Data Wrapper, you must install the following prerequisite packages, and request credentials from EDB:

Install the `epel-release` package:

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

Enable the optional, extras, and HA repositories:

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

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

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

After receiving your repository credentials:

1. Create the repository configuration file.
2. Modify the file, providing your user name and password.
3. Install `edb-as<xx>-mongo_fdw`.

Creating a Repository Configuration File

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

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

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

Modifying the file to provide your user name and password

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

```
[edb]
name=EnterpriseDB RPMs $releasever - $basearch
```

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

Installing the MongoDB Foreign Data Wrapper

After saving your changes to the configuration file, use the following command to install the MongoDB Foreign Data Wrapper:

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

where xx is the server version number.

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

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

On RHEL 8

Before installing the MongoDB Foreign Data Wrapper, you must install the following prerequisite packages, and request credentials from EDB:

Install the `epel-release` package:

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

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

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

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

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

After receiving your repository credentials:

1. Create the repository configuration file.
2. Modify the file, providing your user name and password.
3. Install `edb-as<xx>-mongo_fdw`.

Creating a Repository Configuration File

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

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

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

Modifying the file to provide your user name and password

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

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

Installing the MongoDB Foreign Data Wrapper

After saving your changes to the configuration file, use the following command to install the MongoDB Foreign Data Wrapper:

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

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

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

On CentOS 7

Before installing the MongoDB Foreign Data Wrapper, you must install the following prerequisite packages, and request credentials from EDB:

Install the `epel-release` package:

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

Note

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

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

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

After receiving your repository credentials you can:

1. Create the repository configuration file.
2. Modify the file, providing your user name and password.
3. Install `edb-as<xx>-mongo_fdw`.

Creating a Repository Configuration File

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

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

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

Modifying the file to provide your user name and password

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

```
[edb]
name=EnterpriseDB RPMs $releasever - $basearch
baseurl=https://<username>:
```

```
<password>@yum.enterprisedb.com/edb/redhat/rhel-$releasever-$basearch
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/ENTERPRISEDB-GPG-KEY
```

Installing the MongoDB Foreign Data Wrapper

After saving your changes to the configuration file, use the following command to install the MongoDB Foreign Data Wrapper:

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

where xx is the server version number.

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

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

On CentOS 8

Before installing the MongoDB Foreign Data Wrapper, you must install the following prerequisite packages, and request credentials from EDB:

Install the `epel-release` package:

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

Enable the `PowerTools` repository:

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

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

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

After receiving your repository credentials:

1. Create the repository configuration file.

2. Modify the file, providing your user name and password.
3. Install `edb-as<xx>-mongo_fdw`.

Creating a Repository Configuration File

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

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

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

Modifying the file to provide your user name and password

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

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

Installing the MongoDB Foreign Data Wrapper

After saving your changes to the configuration file, use the following command to install the MongoDB Foreign Data Wrapper:

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

where xx is the server version number.

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

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

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 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 <example_using_the_mongo_data_adapter>

Data Type Mappings

Where Clause Push-down

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

Connection Pooling

Mongo_FDW 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, as in the case of data migration. The syntax of a `DROP EXTENSION` command is:

```
DROP EXTENSION mongo_fdw CASCADE;
```

For more information, see DROP EXTENSION.

---

6 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.
replica_set	The replica set the server is member of. If it is set, the driver will auto-connect to correct primary in the replica set when writing.
read_preference	The order of read preference. Options available are: primary [default], secondary, primaryPreferred, secondaryPreferred, and nearest.
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`

> Specifies 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`

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

`data_type`

> Specifies 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

MongoDB (BSON Type)	Postgres
INT32 SMAL	LINT/INT2/INT/INTEGER/INT4
INT64 BIGI OID NAME	NT/INT8
UTF8 BPCH	AR/VARCHAR/CHARCTER VARYING/TEXT

7 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 postgres
        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;
```

8 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>
```