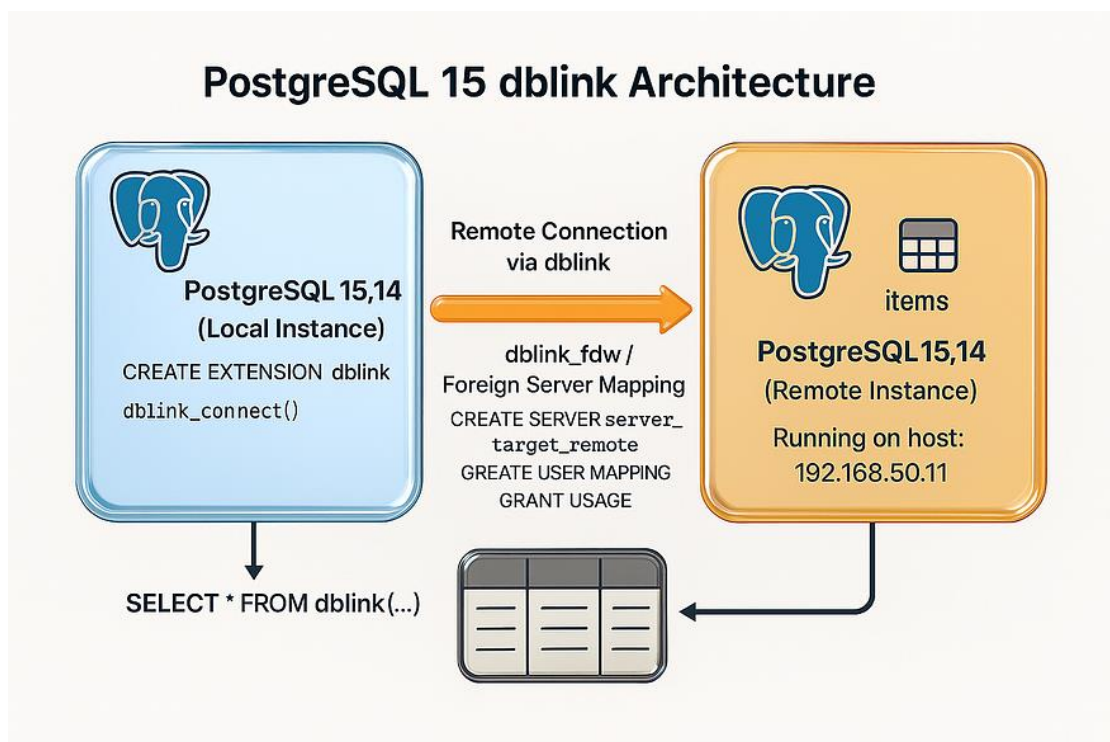


Step-by-Step Guide: Installing and Using the dblink Extension in PostgreSQL 15



PostgreSQL is widely recognized for its robustness and flexibility as an open-source relational database system. One of the features that makes it highly versatile is the support for **extensions**, which add functionality beyond the core database.

Among these, the **dblink extension** is especially powerful. It allows you to connect one PostgreSQL database to another and query remote data directly — without replication, backup restores, or complex ETL (Extract, Transform, Load) jobs.

In this guide, we'll walk through how to:

- Install the `dblink` extension on **PostgreSQL 15.14**
- Enable it inside your database
- Configure a remote connection
- Query data seamlessly from another PostgreSQL instance

Let's get started.

1. Confirming PostgreSQL Version

The first step is to make sure you are running **PostgreSQL 15.14**. This matters because extensions and contrib packages are version-specific.

Connect to PostgreSQL and check the server version:

```
SHOW server_version;

postgres=# SHOW server_version;
 server version 
-----
 15.14
(1 row)
```

If the version matches `15.14`, you're good to go. If not, ensure that you upgrade or install the correct version before proceeding.

2. Installing the Contrib Package

The `dblink` extension is part of the **contrib package** (`postgresql-contrib`), which contains optional but widely used modules. On a Linux system that uses YUM (such as Oracle Linux or CentOS 7), you can install it with:

```
[root@ggnode1 ~]# sudo yum install postgresql15-contrib
```

During installation, the package manager resolves dependencies. You might notice it also installs supporting Python libraries:

```
[root@ggnode1 dbs]# sudo yum install postgresql15-contrib # or: sudo yum install
postgresql15-contrib
ol7_UEKR6
| 3.0 kB 00:00:00
ol7_addons
| 3.0 kB 00:00:00
ol7_developer
| 3.0 kB 00:00:00
ol7_developer_EPEL
| 3.6 kB 00:00:00
ol7_latest
| 3.6 kB 00:00:00
ol7_optional_latest
| 3.0 kB 00:00:00
ol7_software_collections
| 3.0 kB 00:00:00
pgdg-common/7Server/x86_64/signature
| 665 B 00:00:00
pgdg-common/7Server/x86_64/signature
| 2.9 kB 00:00:00 !!!
pgdg12/7Server/x86_64/signature
| 665 B 00:00:00
pgdg12/7Server/x86_64/signature
| 3.6 kB 00:00:00 !!!
pgdg13/7Server/x86_64/signature
| 665 B 00:00:00
pgdg13/7Server/x86_64/signature
| 3.6 kB 00:00:00 !!!
pgdg14/7Server/x86_64/signature
| 665 B 00:00:00
pgdg14/7Server/x86_64/signature
| 3.6 kB 00:00:00 !!!
pgdg15/7Server/x86_64/signature
| 665 B 00:00:00
pgdg15/7Server/x86_64/signature
| 3.6 kB 00:00:00 !!!
Resolving Dependencies
--> Running transaction check
--> Package postgresql15-contrib.x86_64 0:15.14-1PGDG.rhel7 will be installed
--> Processing Dependency: libpython3.6m.so.1.0() (64bit) for package: postgresql15-
contrib-15.14-1PGDG.rhel7.x86_64
--> Running transaction check
--> Package python3-libs.x86_64 0:3.6.8-21.0.1.el7_9 will be installed
--> Processing Dependency: python(abn) = 3.6 for package: python3-libs-3.6.8-
21.0.1.el7_9.x86_64
--> Running transaction check
--> Package python3.x86_64 0:3.6.8-21.0.1.el7_9 will be installed
--> Processing Dependency: python3-pip for package: python3-3.6.8-
21.0.1.el7_9.x86_64
--> Processing Dependency: python3-setuptools for package: python3-3.6.8-
21.0.1.el7_9.x86_64
--> Running transaction check
--> Package python3-pip.noarch 0:9.0.3-8.0.3.el7 will be installed
--> Package python3-setuptools.noarch 0:39.2.0-10.0.3.el7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
=====
=====
Package                                     Arch
Version                                     Repository
Size
```

```

=====
Installing:
 postgresql15-contrib                x86_64
15.14-1PGDG.rhel7                    pgdg15
722 k
Installing for dependencies:
 python3                             x86_64
3.6.8-21.0.1.el7_9                   ol7_latest
70 k
 python3-libs                         x86_64
3.6.8-21.0.1.el7_9                   ol7_latest
7.0 M
 python3-pip                          noarch
9.0.3-8.0.3.el7                      ol7_latest
1.6 M
 python3-setuptools                   noarch
39.2.0-10.0.3.el7                    ol7_latest
629 k

```

Transaction Summary

```

=====
Install 1 Package (+4 Dependent packages)

```

```

Total download size: 10 M
Installed size: 50 M
Is this ok [y/d/N]: y
Downloading packages:
(1/5): python3-3.6.8-21.0.1.el7_9.x86_64.rpm
| 70 kB 00:00:00
(2/5): postgresql15-contrib-15.14-1PGDG.rhel7.x86_64.rpm
| 722 kB 00:00:00
(3/5): python3-pip-9.0.3-8.0.3.el7.noarch.rpm
| 1.6 MB 00:00:00
(4/5): python3-setuptools-39.2.0-10.0.3.el7.noarch.rpm
| 629 kB 00:00:00
(5/5): python3-libs-3.6.8-21.0.1.el7_9.x86_64.rpm
| 7.0 MB 00:00:01

```

```

-----
Total
7.7 MB/s | 10 MB 00:00:01

```

When prompted, type **y** to proceed. After downloading and installing the package, `dblink` becomes available for use.

3. Verifying dblink Files

Once installation is complete, confirm that the necessary files exist on your system. These files represent the extension definition and its library:

```
ls /usr/pgsql-15/share/extension/dblink.control
```

```
[root@ggnode1 ~]# ls /usr/pgsql-15/share/extension/dblink.control
/usr/pgsql-15/share/extension/dblink.control
[root@ggnode1 ~]#
```

```
ls /usr/pgsql-15/lib/dblink.so
```

```
[root@ggnode1 ~]# ls /usr/pgsql-15/lib/dblink.so
/usr/pgsql-15/lib/dblink.so
[root@ggnode1 ~]#
```

- **dblink.control**: Metadata file that describes the extension.
- **dblink.so**: Shared library providing the actual functionality.

If both files are present, installation was successful.

4. Enabling the Extension in PostgreSQL

With the contrib package installed, you can enable `dblink` inside your PostgreSQL database.

Run the following command:

```
CREATE EXTENSION dblink;
```

```
postgres=# CREATE EXTENSION dblink;
CREATE EXTENSION
```

You can verify that it has been enabled by listing installed extensions:

```
\dx dblink
```

```
postgres=# \dx dblink
```

List of installed extensions

Name	Version	Schema	Description
dblink	1.2	public	connect to other PostgreSQL databases from within a database

(1 row)

To double-check, you can query available extensions and their versions:

```
SELECT name, default_version, installed_version
FROM pg_available_extensions
WHERE name = 'dblink';
```

```
postgres=# SELECT name, default_version, installed_version
postgres=# FROM pg_available_extensions
postgres=# WHERE name = 'dblink';
 name | default_version | installed_version
-----+-----+-----
 dblink | 1.2             | 1.2
(1 row)
```

At this point, the extension is fully active and ready for configuration.

5. Configuring a Remote Connection

The real power of `dblink` lies in its ability to connect to remote PostgreSQL instances. To achieve this, you must first define a **foreign server** and map it to a user.

Step 1: Create a Foreign Server

```
CREATE SERVER server_target_remote FOREIGN DATA WRAPPER dblink fdw
OPTIONS (host '192.168.50.11', dbname 'postgres', port '5432');
```

```
postgres=# CREATE SERVER server_target_remote FOREIGN DATA WRAPPER dblink_fdw
OPTIONS (host '192.168.50.11', dbname 'postgres', port '5432');
CREATE SERVER
```

Here's what the options mean:

- `host`: IP address of the remote server
- `dbname`: Name of the remote database
- `port`: Port on which PostgreSQL is running (default is 5432)

Step 2: Create User Mapping

Now map your local user to a remote user, including credentials:

```
CREATE USER MAPPING FOR current_user SERVER server_target_remote
OPTIONS (user 'postgres', password 'oracle123');

postgres=# CREATE USER MAPPING FOR current_user SERVER server_target_remote
OPTIONS (user 'postgres', password 'oracle123');
CREATE USER MAPPING
```

This ensures that whenever the current user connects via `dblink`, PostgreSQL knows which credentials to use on the remote server.

Step 3: Grant Usage

Finally, grant permission to the local user for using the foreign server:

```
GRANT USAGE ON FOREIGN SERVER server_target_remote TO current_user;

postgres=# GRANT USAGE ON FOREIGN SERVER server_target_remote TO current_user;
GRANT
```

6. Connecting to the Remote Database

With the server and mapping in place, establish a connection:

```
SELECT dblink_connect('conn_db_link', 'server_target_remote');
```

```
postgres=# SELECT dblink connect('conn db link', 'server target remote');
dblink connect
-----
OK
(1 row)
```

- `conn_db_link`: The connection name you assign locally.
- `server_target_remote`: The foreign server created earlier.

A response of `OK` means the connection was successful.

7. Querying Remote Data

Once connected, you can query remote tables as if they were part of your local database. For example:

```
SELECT *
FROM dblink('conn db link', 'SELECT item_id, quantity, price FROM items')
AS result(item_id INT, quantity INT, price NUMERIC(10,2));
```

This query returns results directly from the remote `items` table:

```
postgres=# SELECT *
FROM dblink('conn db link', 'SELECT item id, quantity, price FROM items')
AS result(item id INT, quantity INT, price NUMERIC(10,2));
 item id | quantity | price
-----+-----+-----
 9093121 |      297 | 395.80
 9093122 |      640 | 226.10
 9093123 |      238 | 173.32
 9093124 |      200 |  80.91
 9093125 |       18 | 222.13
 9093126 |      945 | 101.12
 9093127 |      419 | 192.06
 9093128 |      739 | 493.85
 9093129 |      755 | 163.90
 9093130 |      374 |  69.76
 9093131 |      613 | 339.56
 9093132 |      923 |  75.71
 9093133 |      917 | 461.97
 9093134 |      916 | 156.12
 9093135 |      719 | 433.66
```


This is incredibly useful for scenarios where you need to:

- Consolidate reporting across multiple databases
- Migrate data incrementally
- Validate remote datasets without exporting them

8. Wrapping Up

The `dblink` extension is one of the simplest yet most effective ways to query across multiple PostgreSQL databases.

In this walkthrough, you learned how to:

1. Verify your PostgreSQL version (15.14)
2. Install the `postgresql15-contrib` package
3. Check that the necessary extension files exist
4. Enable the extension with `CREATE EXTENSION`
5. Configure a foreign server and user mapping
6. Establish a connection with `dblink_connect`
7. Query remote data directly from within your local session

By following these steps, you can extend PostgreSQL's capabilities beyond a single instance and unlock cross-database functionality for analytics, troubleshooting, or even lightweight data integration.

✦ Tip: While `dblink` is very useful, remember to use it wisely. For complex or frequent cross-database operations, consider **Foreign Data Wrappers (FDW)** like `postgres_fdw`, which offer better integration and performance tuning.