



EDB

Postgres extensions

Release 1.0.0

EnterpriseDB Corporation

Dec 09, 2022

CONTENTS

- 1 Open source extensions 3**
- 2 EDB extensions 5**
- 3 EDB supported open source extensions 7**
 - 3.1 EDB Advanced Storage Pack 7
 - 3.1.1 Autocluster 8
 - 3.1.2 Refdata 8
 - 3.2 EDB LDAP Sync 14
 - 3.2.1 LDAP Sync release notes 15
 - 3.2.2 Installing EDB LDAP Sync 15
 - 3.2.3 Configuring EDB LDAP Sync 16
 - 3.2.4 Using EDB LDAP Sync 24
 - 3.3 EDB Postgres Tuner 25
 - 3.3.1 Postgres Tuner release notes 25
 - 3.3.2 Installing EDB Postgres Tuner 26
 - 3.3.3 Parameters 27
 - 3.3.4 Using EDB Postgres Tuner 28

Categories of extensions:

- *Open source extensions*
- *EDB extensions*
- *EDB supported open source extensions*

Note: For BigAnimal: - PostgreSQL Compatible corresponds to EDB Postgres Extended Server. - Oracle Compatible corresponds to EDB Postgres Advanced Server.

OPEN SOURCE EXTENSIONS

Extension name	Requires superuser access	Virtual Machines/Physical Servers			Kubernetes		BigAnimal		
		PostgreSQL	EDB Postgres Extended Server	EDB Postgres Advanced Server	PostgreSQL	EDB Postgres Advanced Server	PostgreSQL	PostgreSQL Compatible	Oracle Compatible
PostgreSQL Contrib Extensions/Modules									
adminpack	Yes	✓	✓	✓	–	–	–	–	–
amcheck	Yes	✓	✓	✓	✓	✓	✓	✓	✓
auth_delay	Yes	✓	✓	✓	✓	✓	✓	✓	✓
auto_explain	Yes	✓	✓	✓	✓	✓	✓	✓	✓
bloom	Yes	✓	✓	✓	✓	✓	✓	✓	✓
btree_gin		✓	✓	✓	✓	✓	✓	✓	✓
btree_gist		✓	✓	✓	✓	✓	✓	✓	✓
citext		✓	✓	✓	✓	✓	✓	✓	✓
cube		✓	✓	✓	✓	✓	✓	✓	✓
dblink	Yes	✓	✓	✓	✓	✓	✓	✓	✓
dict_int		✓	✓	✓	✓	✓	✓	✓	✓
dict_xsyn	Yes	✓	✓	✓	✓	✓	✓	✓	✓
earthdistance	Yes	✓	✓	✓	✓	✓	✓	✓	✓
file_fdw	Yes	✓	✓	✓	–	–	–	–	–
fuzzystrmatch		✓	✓	✓	✓	✓	✓	✓	✓
hstore		✓	✓	✓	✓	✓	✓	✓	✓
hstore_plperl	Yes	✓	✓	✓	–	–	✓	✓	✓
hstore_plperl_u	Yes	✓	✓	✓	–	–	✓	✓	✓
intagg	Yes	✓	✓	✓	✓	✓	✓	✓	✓
intarray		✓	✓	✓	✓	✓	✓	✓	✓
isn		✓	✓	✓	✓	✓	✓	✓	✓
jsonb_plperl		✓	✓	✓	–	–	✓	✓	✓
jsonb_plperl_u	Yes	✓	✓	✓	–	–	✓	✓	✓
lo		✓	✓	✓	✓	✓	✓	✓	✓
ltree		✓	✓	✓	✓	✓	✓	✓	✓
old_snapshot	Yes	✓	✓	✓	✓	✓	✓	✓	✓
pageinspect	Yes	✓	✓	✓	✓	✓	✓	✓	✓
passwordcheck		✓	✓	✓	–	–	–	–	–
plpgsql		✓	✓	✓	✓	✓	✓	✓	✓
pg_buffercache	Yes	✓	✓	✓	✓	✓	✓	✓	✓
pg_freeze	Yes	✓	✓	✓	✓	✓	✓	✓	✓
pg_prewarm	Yes	✓	✓	✓	✓	✓	✓	✓	✓
pg_stat_statements	Yes	✓	✓	✓	✓	✓	✓	✓	✓
pg_surgery	Yes	✓	✓	✓	✓	✓	✓	✓	✓
pg_trgm		✓	✓	✓	✓	✓	✓	✓	✓
pg_visibility	Yes	✓	✓	✓	✓	✓	✓	✓	✓
pgcrypto	Yes	✓	✓	✓	✓	✓	✓	✓	✓
pgrowlocks	Yes	✓	✓	✓	✓	✓	✓	✓	✓
pgstattuple	Yes	✓	✓	✓	✓	✓	✓	✓	✓
postgres_fdw	Yes	✓	✓	✓	✓	✓	✓	✓	✓
seg		✓	✓	✓	✓	✓	✓	✓	✓
sepgsql		✓	✓	✓	–	–	–	–	–
spi		✓	✓	✓	✓	✓	✓	✓	✓
autoinc	Yes	✓	✓	✓	✓	✓	✓	✓	✓
insert_username	Yes	✓	✓	✓	✓	✓	✓	✓	✓
moddatetime	Yes	✓	✓	✓	✓	✓	✓	✓	✓
refint	Yes	✓	✓	✓	✓	✓	✓	✓	✓
sslinfo	Yes	✓	✓	✓	✓	✓	✓	✓	✓
tablefunc		✓	✓	✓	✓	✓	✓	✓	✓
tcn		✓	✓	✓	✓	✓	✓	✓	✓
test_decoding	Yes	✓	✓	✓	✓	✓	✓	✓	✓
tsm_system_rows		✓	✓	✓	✓	✓	✓	✓	✓
tsm_system_time		✓	✓	✓	✓	✓	✓	✓	✓
unaccent		✓	✓	✓	✓	✓	✓	✓	✓
uuid-oss	Yes	✓	✓	✓	✓	✓	✓	✓	✓
xml2	Yes	✓	✓	✓	✓	✓	✓	✓	✓
PostgreSQL Non-Contrib Extensions/Modules									
ip4r		✓	–	–	–	–	–	–	–
pg_permissions		✓	–	–	–	–	–	–	–
pg_store_plans		✓	–	–	–	–	–	–	–
pgaudit	Yes	✓	✓	–	✓	–	✓	–	–
pglogical 2.x		✓	✓	–	–	–	–	–	–
pgrouting		✓	–	–	–	–	–	–	–
pgsmpd (11-13)		✓	–	✓	–	–	–	–	–
pljava (11)		✓	–	✓	–	–	–	–	–
plperl		✓	✓	✓	–	–	✓	✓	✓
plperl_u	Yes	✓	✓	✓	–	–	✓	✓	✓
bool_plperl		✓	✓	✓	–	–	✓	✓	✓
bool_plperl_u	Yes	✓	✓	✓	–	–	✓	✓	✓
plpythonu (default by version)	Yes	✓	✓	✓	–	–	–	–	–
hstore_plpythonu	Yes	✓	✓	✓	–	–	–	–	–
jsonb_plpythonu	Yes	✓	✓	✓	–	–	–	–	–
ltree_plpythonu	Yes	✓	✓	✓	–	–	–	–	–
plpython2u (default 11-14)		✓	✓	✓	–	–	✓	✓	✓
hstore_plpython2u		✓	✓	✓	–	–	✓	✓	✓
jsonb_plpython2u		✓	✓	✓	–	–	✓	✓	✓
ltree_plpython2u		✓	✓	✓	–	–	✓	✓	✓
plpython3u (default 15)	Yes	✓	✓	✓	–	–	✓	✓	✓
hstore_plpython3u	Yes	✓	✓	✓	–	–	✓	✓	✓
jsonb_plpython3u	Yes	✓	✓	✓	–	–	✓	✓	✓
ltree_plpython3u	Yes	✓	✓	✓	–	–	✓	✓	✓
pltcl		✓	✓	✓	–	–	✓	✓	✓
pltclu	Yes	✓	✓	✓	–	–	✓	✓	✓
set_user		✓	–	–	–	–	–	–	–
sslutils		✓	–	–	–	–	–	–	–
timescaledb		✓	–	–	–	–	–	–	–

EDB EXTENSIONS

Extension name	Requires superuser access	Virtual Machines/Physical Servers			Kubernetes		BigAnimal		
		PostgreSQL	EDB Postgres Extended Server	EDB Postgres Advanced Server	PostgreSQL	EDB Postgres Advanced Server	PostgreSQL	PostgreSQL Compatible	Oracle Compatible
autocluster		✓	✓	✓	—	—	—	—	—
bdr 3.x		✓	✓	✓	—	—	—	—	—
bdr 4.x		✓	✓	✓	—	—	—	—	—
dbms_job		—	—	✓	—	—	—	—	—
dbms_scheduler		—	—	✓	—	—	—	—	—
edb_cloneschema		—	—	✓	—	—	—	—	—
edb_dblink_libpq		—	—	✓	—	✓	—	—	✓
edb_dblink_oci	Yes	—	—	✓	—	✓	—	—	✓
edb_dbo	Yes	—	—	✓	—	✓	—	—	✓
edb_pg_tuner		✓	✓	✓	✓	✓	—	—	—
edb_wait_states		✓	✓	✓	—	—	—	—	—
edbspl	Yes	—	—	✓	—	✓	—	—	✓
parallel_clone		—	—	✓	—	—	—	—	—
pglogical 3.x	Yes	✓	✓	✓	—	—	—	—	—
refdata		✓	✓	✓	—	—	—	—	—

Fig. 1: EDB extensions

EDB SUPPORTED OPEN SOURCE EXTENSIONS

Extension name	Requires superuser access	Virtual Machines/Physical Servers			Kubernetes		BigAnimal		
		PostgreSQL	EDB Postgres Extended Server	EDB Postgres Advanced Server	PostgreSQL	EDB Postgres Advanced Server	PostgreSQL	PostgreSQL Compatible	Oracle Compatible
PostGIS									
postgis		✓	–	✓	✓	✓	–	–	–
postgis_raster		✓	–	✓	✓	✓	–	–	–
postgis_sfcgal		✓	–	✓	✓	✓	–	–	–
postgis_tiger_geocoder		✓	–	✓	✓	✓	–	–	–
postgis_topology		✓	–	✓	✓	✓	–	–	–
address_standardizer		✓	–	✓	✓	✓	–	–	–
address_standardizer_data_us		✓	–	✓	✓	✓	–	–	–
Pgpool									
pgpool_adm		✓	–	✓	–	–	–	–	–
pgpool_recovery		✓	–	✓	–	–	–	–	–
PLdebugger									
pldbgapi		✓	–	✓	–	–	–	–	–
Foreign Data Wrappers									
hdfs_fdw	Yes	✓	–	✓	–	–	–	–	–
mongo_fdw	Yes	✓	–	✓	–	–	–	–	✓
mysql_fdw	Yes	✓	–	✓	–	–	–	–	✓

Fig. 1: EDB supported open source extensions

3.1 EDB Advanced Storage Pack

EDB Advanced Storage Pack provides advanced storage options for PostgreSQL databases in the form of table access method (TAM) extensions. These storage options can enhance the performance and reliability of databases without requiring application changes.

For tables whose access patterns you know in advance, you might prefer a targeted TAM that makes different tradeoffs. For instance, if a given table in an application is INSERT-only and the rows never receive any updates, you can might consider using a specialized TAM for this table that has INSERT-specific optimizations.

EnterpriseDB offers two TAMs in the Advanced Storage Pack.

3.1.1 Autocluster

The Autocluster TAM provides faster access to clustered data by keeping track of the last inserted row for any value in a side table. You can then add new rows to the same data blocks as previous rows, keeping the data clustered. This approach reduces access time to related data. To use this feature, keep rows with the same key values clustered together so that an index scan for a specific key can find all the rows close together. The scan doesn't need to retrieve as many table pages to satisfy the query.

3.1.2 Refdata

The Refdata TAM is optimized for mostly static data, which contains an occasional INSERT and very few DELETE and UPDATE commands. For database schemas that use foreign keys to reference data, this TAM can provide performance gains of 5-10% and increased scalability. To use this feature, take an exclusive lock on the reference table whenever you modify it, blocking out concurrent modifications by any other session as well as modifications to tables that reference the table. For example:

```
CREATE TABLE department (  
    department_id SERIAL PRIMARY KEY,  
    department_name TEXT  
) USING refdata;  
  
CREATE TABLE employee (  
    ...  
    department_id NOT NULL REFERENCES department(department_id)  
);
```

The employee table is a standard heap table. Only the department table uses the refdata TAM. Inserts and updates of the employee table don't take out row-level locks on the department table, which saves query time. This approach also avoids the need to update the rows in the department table and avoids the need to write the referenced department table rows to disk and to the write-ahead log.

If updates to the department table are frequent, we recommend that you don't use the Refdata TAM because concurrent modifications to it and to the employee table are then blocked. If you make infrequent changes to the department table, speeding up frequent changes to the employee table and reducing write-ahead log traffic might be worth this cost.

Advanced Storage Pack release notes

The Advanced Storage Pack documentation describes the latest version of Advanced Storage Pack, including minor releases and patches. These release notes cover what was new in each release. For new functionality introduced in a minor or patch release, there are also indicators in the content about the release that introduced the feature.

Version	Release Date
<i>Release notes for Advanced Storage Pack version 1.0.0</i>	2022 Nov 30

Release notes for Advanced Storage Pack version 1.0.0

This release of Advanced Storage Pack includes:

Type	Description
Feature	This is the initial release and includes the Refdata and Autocluster TAMs.

Installing Advanced Storage Pack

The Advanced Storage Pack is supported on the same platforms as the Postgres distribution you're using. Support for Advanced Storage Pack starts with Postgres 11. For details, see:

- [EDB Postgres Advanced Server Product Compatibility](#)
- [PostgreSQL Product Compatibility](#)
- [EDB Postgres Distributed \(includes EDB Postgres Extended\)](#)

Prerequisites

Before you begin the installation process:

- Install Postgres. See: - [Installing EDB Postgres Advanced Server](#)
- [Installing PostgreSQL](#)
- [Installing EDB Postgres Distributed \(includes EDB Postgres Extended\)](#)
- Set up the repository.

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

To set up the repository, go to [EDB repositories](#) and follow the instructions provided there.

Install the package

The syntax for the RPM package install command is:

```
sudo <package-manager> -y install edb-<postgres><postgres_version>-advanced-storage-pack-  
↪<major_version>
```

The syntax for the Debian package install command is:

```
sudo <package-manager> -y install edb-<postgres><postgres_version>-advanced-storage-pack-  
↪<major_version>
```

Where:

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

Package manager	Operating system
dnf	RHEL 8 and derivatives
yum	RHEL 7 and derivatives, CentOS 7
zypper	SLES
apt-get	Debian and derivatives

- `<postgres>` is the distribution of Postgres you're using:

Postgres distribution	Value
PostgreSQL	pg
EDB Postgres Advanced Server	as
EDB Postgres Extended	pgextended

- `<postgres_version>` is the version of Postgres you're using.
- `<major_version>` is the major version of Advanced Storage Pack you're installing.

For example, to install Advanced Storage Pack 1.0.0 for EDB Postgres Advanced Server 14 on a RHEL 8 platform:

```
sudo dnf -y install edb-as14-advanced-storage-pack1
```

To install Advanced Storage Pack 1.0.0 for EDB Postgres Advanced Server 14 on a Debian 11 platform:

```
sudo apt-get -y install edb-pg15-advanced-storage-pack-1
```

Configuring Advanced Storage Pack

Place the extension module implementing the custom TAM in `shared_preload_libraries` so that it loads early during Postgres startup. This step is needed so that the extension is available before the table based on the given TAM is accessed for the first time. For example, update the parameter in `postgresql.conf` with `autocluster` or `refadata` :

```
shared_preload_libraries = $libdir/<extension_name>
```

After restarting the server, execute the SQL command to create the extension. This command creates the extension only in the connected database where the SQL is executed. You must rerun it in each database where the extension is used:

```
CREATE EXTENSION <extension_name>;
```

In databases where the extension was created, you can create tables to use the TAM, which the extension provides:

```
CREATE TABLE mytable USING <extension_name>;
```

Using EDB Advanced Storage Pack

The following are scenarios where the EDB Advances Storage Pack TAMs are useful.

Refdata example

A scenario where Refdata is useful is when creating a reference table of all the New York Stock Exchange (NYSE) stock symbols and their corporate names. This data is expected to change very rarely and be referenced frequently from a table tracking all stock trades for the entire market (like in the [Advanced Autocluster example](#)). You can use Refdata instead of heap to increase performance.

```
CREATE SEQUENCE nyse_symbol_id_seq;
CREATE TABLE nyse_symbol (
    nyse_symbol_id INTEGER NOT NULL PRIMARY KEY DEFAULT NEXTVAL(nyse_symbol_id_seq),
    symbol TEXT NOT NULL,
    name TEXT NOT NULL
) USING refdata;
```
 ## Autocluster example

```

A scenario where Autocluster is useful is with Internet of Things (IoT) data, which are usually inserted with many rows that relate to each other and often use append-only data. When using heap instead of Autocluster, Postgres can't cluster together these related rows, so access to the set of rows touches many data blocks, can be very slow, and is input/output heavy.

This example is for an IoT thermostat that reports house temperatures and temperature settings every 60 seconds:

```
```sql
CREATE TABLE iot (
    thermostat_id      bigint,
    recordtime          timestamp,
    measured_temperature float 4,
    temperature_setting float 4,
) USING autocluster;
```

Using Autocluster, rows with the same thermostat_id are clustered together and are easier to access:

```
CREATE INDEX ON iot USING btree(thermostat_id);
SELECT autocluster.autocluster(
    rel := iot::regclass,
    cols := {1},
    max_objects := 10000
);
```

Note: The *cols* parameter must match the number of columns specified in *USING btree()*. In this case, only *thermostat_id* is listed so the value is *{1}*. !!!

Populate the table with the thermostat_id and recordtime data:

```

INSERT INTO iot (thermostat_id, recordtime) VALUES (456, 12:01);
INSERT INTO iot (thermostat_id, recordtime) VALUES (8945, 04:55);
INSERT INTO iot (thermostat_id, recordtime) VALUES (456, 15:32);
INSERT INTO iot (thermostat_id, recordtime) VALUES (6785, 01:36);
INSERT INTO iot (thermostat_id, recordtime) VALUES (456, 19:25);
INSERT INTO iot (thermostat_id, recordtime) VALUES (5678, 03:44);

```

When you select the data from the IoT table, you can see from the ctid location that the data with the same thermostat_id was clustered together:

```
SELECT ctid, thermostat_id, recordtime FROM iot;
```

__OUTPUT__

ctid	thermostat_id	recordtime
(0,1)	456	12:01
(2,2)	8945	04:55
(0,2)	456	15:32
(3,2)	6785	01:36
(0,3)	456	19:25
(2,5)	5678	03:44

(6 rows)

Advanced Autocluster example

This is a more advanced way of using Autocluster than the previous example. It involves referencing the NYSE table from the [Refdata example](#) and clustering together the rows based on the stock symbol. This approach makes it easier to find the latest number of trades.

Start with the NYSE table from the Refdata example:

```

CREATE SEQUENCE nyse_symbol_id_seq;
CREATE TABLE nyse_symbol (
    nyse_symbol_id INTEGER NOT NULL PRIMARY KEY DEFAULT NEXTVAL(nyse_symbol_id_seq),
    symbol TEXT NOT NULL,
    name TEXT NOT NULL
) USING refdata;

```

Create a highly updated table containing NYSE trades, referencing the mostly static stock symbols in the Refdata table. Cluster the rows on the stock symbol to make it easier to look up the last x trades for a given stock:

```

CREATE TABLE nyse_trade (
    nyse_symbol_id INTEGER NOT NULL REFERENCES nyse_symbol(nyse_symbol_id),
    trade_time TIMESTAMP NOT NULL DEFAULT NOW(),
    trade_price FLOAT8 NOT NULL CHECK(trade_price >= 0.0),
    trade_volume BIGINT NOT NULL CHECK(trade_volume >= 1)
); -- USING autocluster;
CREATE INDEX ON nyse_trade USING BTREE(nyse_symbol_id);
SELECT autocluster.autocluster(
    rel := nyse_trade::regclass,
    cols := {1},
    max_objects := 3000
);

```

(continues on next page)

(continued from previous page)

```
autocluster
- -----
(1 row)
```

Create a view to facilitate inserting by symbol name rather than id:

```
CREATE VIEW nyse_trade_symbol AS
  SELECT ns.symbol, nt.trade_time, nt.trade_price, nt.trade_volume
    FROM nyse_symbol ns
   JOIN nyse_trade nt
     ON ns.nyse_symbol_id = nt.nyse_symbol_id;
CREATE RULE stock_insert AS ON INSERT TO nyse_trade_symbol
  DO INSTEAD INSERT INTO nyse_trade
    (SELECT ns.nyse_symbol_id, NEW.trade_time, NEW.trade_price, NEW.trade_volume
      FROM nyse_symbol ns
     WHERE ns.symbol = NEW.symbol
    );
```

For more information on creating a view, see the [PostgreSQL documentation](#).

Prepopulate the static data (shortened for brevity):

```
INSERT INTO nyse_symbol (symbol, name) VALUES
(A, Agilent Technologies),
(AA, Alcoa Corp),
(AAC, Ares Acquisition Corp Cl A),
(AAIC, Arlington Asset Investment Corp),
(AAIN, Arlington Asset Investment Corp 6.000%),
(AAN, Aarons Holdings Company),
(AAP, Advance Auto Parts Inc),
(AAQC, Accelerate Acquisition Corp Cl A),
(ZTR, Virtus Total Return Fund Inc),
(ZTS, Zoetis Inc Cl A),
(ZUO, Zuora Inc),
(ZVIA, Zevia Pbc Cl A),
(ZWS, Zurn Elkay Water Solutions Corp),
(ZYME, Zymeworks Inc);
ANALYZE nyse_symbol;
```

Insert stock trades over a given time range on Friday, November 18, 2022 (shortened for brevity):

```
\timing
INSERT INTO nyse_trade_symbol VALUES (NSC, Fri Nov 18 09:51:32 2022, 248.100000, 98778);
Time: 32.349 ms
INSERT INTO nyse_trade_symbol VALUES (BOE, Fri Nov 18 09:51:32 2022, 9.640000, 72973);
Time: 1.055 ms
INSERT INTO nyse_trade_symbol VALUES (LOMA, Fri Nov 18 09:51:32 2022, 6.180000, 41632);
Time: 0.927 ms
INSERT INTO nyse_trade_symbol VALUES (LXP, Fri Nov 18 09:51:32 2022, 10.670000, 85768);
Time: 0.941 ms
INSERT INTO nyse_trade_symbol VALUES (ABBV, Fri Nov 18 09:51:32 2022, 155.000000, 46842);
Time: 0.916 ms
```

(continues on next page)

(continued from previous page)

```

INSERT INTO nyse_trade_symbol VALUES (AGD, Fri Nov 18 09:51:32 2022, 9.360000, 90684);
Time: 0.669 ms
INSERT INTO nyse_trade_symbol VALUES (PAGS, Fri Nov 18 11:14:31 2022, 12.985270, 34734);
Time: 0.849 ms
INSERT INTO nyse_trade_symbol VALUES (KTF, Fri Nov 18 11:14:31 2022, 8.435753, 73719);
Time: 0.679 ms
INSERT INTO nyse_trade_symbol VALUES (AES, Fri Nov 18 11:14:31 2022, 28.072732, 549);
Time: 0.667 ms
INSERT INTO nyse_trade_symbol VALUES (LIN, Fri Nov 18 11:14:31 2022, 334.617829, 39838);
Time: 0.665 ms
INSERT INTO nyse_trade_symbol VALUES (DTB, Fri Nov 18 11:14:31 2022, 18.679245, 55863);
Time: 0.680 ms
ANALYZE nyse_trade;
Time: 73.832 ms

```

Select the ctid from the data for a given stock symbol to see in the output how it was clustered together:

```

SELECT ctid, * FROM nyse_trade WHERE nyse_symbol_id = 1000 ORDER BY trade_time DESC
LIMIT 10;
--OUTPUT--

```

ctid	nyse_symbol_id	trade_time	trade_price	trade_volume
(729,71)	1000	Fri Nov 18 11:13:51 2022	11.265938	72662
(729,22)	1000	Fri Nov 18 11:08:39 2022	11.262747	50897
(729,20)	1000	Fri Nov 18 11:08:30 2022	11.267203	37120
(729,9)	1000	Fri Nov 18 11:07:21 2022	11.269852	792
(729,6)	1000	Fri Nov 18 11:07:02 2022	11.268067	46221
(632,123)	1000	Fri Nov 18 11:04:46 2022	11.272623	97874
(632,118)	1000	Fri Nov 18 11:04:28 2022	11.271794	65579
(632,14)	1000	Fri Nov 18 10:55:45 2022	11.268543	8557
(632,2)	1000	Fri Nov 18 10:54:45 2022	11.26414	94078
(506,126)	1000	Fri Nov 18 10:54:01 2022	11.264657	89641

(10 rows)

3.2 EDB LDAP Sync

EDB LDAP Sync is a collection of utilities and configurations for synchronizing Postgres with directory servers that support the LDAP protocol, such as OpenLDAP and Active Directory. Database administrators can schedule ldap2pg to synchronize Postgres users and groups with LDAP users and groups. EDB LDAP Sync replaces external schedulers, like cron, while using a similar syntax.

3.2.1 LDAP Sync release notes

The LDAP Sync documentation describes the latest version of LDAP Sync, including minor releases and patches. These release notes cover what was new in each release. For new functionality introduced in a minor or patch release, there are also indicators in the content about the release that introduced the feature.

Version	Release Date
<i>Release notes for Advanced Storage Pack version 1.0.0</i>	2022 Nov 30

Release notes for LDAP Sync version 1.0.0

This release of LDAP Sync includes:

Type	Description
Feature	This is the initial release.

3.2.2 Installing EDB LDAP Sync

EDB LDAP Sync is supported on the same platforms as the Postgres distribution you're using. Support for EDB LDAP Sync starts with Postgres 11. See:

- [EDB Postgres Advanced Server Product Compatibility](#)
- [PostgreSQL Product Compatibility](#)
- [EDB Postgres Distributed \(includes EDB Postgres Extended\)](#)

Prerequisites

Before you begin the installation process:

- Install Postgres. See: [- Installing EDB Postgres Advanced Server](#)
- [Installing PostgreSQL](#)
- Installing EDB Postgres Distributed (includes EDB Postgres Extended)
- Install *Configure ldap2pg* .
- Set up the repository.

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

To set up the repository, go to [EDB repositories](#) and follow the instructions provided there.

Install the package

The syntax for the RPM package install command is:

```
sudo <package-manager> -y install edb-ldap-sync<major_version>
```

The syntax for the Debian package install command is:

```
sudo <package-manager> -y install edb-ldap-sync-<major_version>
```

Where:

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

Package manager	Operating system
dnf	RHEL 8 and derivatives
yum	RHEL 7 and derivatives, CentOS 7
zypper	SLES
apt-get	Debian and derivatives

- `<major_version>` is the major version of EDB LDAP Sync you're installing.

For example, to install EDB LDAP Sync version 1.0.0 on a RHEL 8 platform:

```
sudo dnf -y install edb-ldap-sync1
```

To install EDB LDAP Sync version 1.0.0 on a Debian 11 platform:

```
sudo apt-get -y install edb-ldap-sync-1
```

3.2.3 Configuring EDB LDAP Sync

EDB LDAP Sync is a collection of tools that synchronize credentials from an LDAP or Active Directory server to EDB Postgres Advanced Server. We recommend that you configure its tools in the following ways.

Find the sample EDB LDAP Sync configuration files at `/usr/lib/python3.6/site-packages/ldap_sync/samples/`.

Configure pgAgent

pgAgent sets up periodic synchronization tasks. Use `edb-pgagent.conf_tmpl` as the default configuration for pgAgent setup with Postgres. Copy it into the configuration directory for pgAgent with Postgres.

For more information, see the pgAgent documentation .

Setting Postgres database permissions

You need to modify the permission settings for the database server to allow Postgres to query the LDAP host during Postgres authentication. You can do this by updating `pg_hba.conf` in the configuration folder of the database server. For EDB Postgres Advanced Server configuration, `pg_hba.conf` is located in `/opt/postgres/data/pg_hba.conf`.

pg_hba.conf settings

Note: The user varies depending on your database service. PostgreSQL has a *postgres* user. EDB Postgres Advanced Server has an *enterprisedb* user. !!!

When configuring `pg_hba.conf`, the LDAP authentication string must come after the *enterprisedb* user authentication string. Otherwise, due to the order of evaluation, the *enterprisedb* user can't log into the system. For example, LDAP server is queried instead of local authentication, LDAP server returns an authentication failure, and the authentication process terminates. For this reason, you need to add host authentication for the *enterprisedb* user if you want to use host authentication as a part of the `ldap2pg` setup.

The following authentication methods are available: peer, host with a password string by way of `dsn` option, and host by way of reading `.pgpass`. See [Configure ldap2pg](#) for more information.

The Postgres database permissions are set according to your database requirements. The following are sample `pg_hba.conf` stanzas for LDAP authentication queries, where the values in `{}` are the LDAP server hostname and the basedn LDAP query string:

```
# TYPE DATABASE USER ADDRESS METHOD
local all enterprisedb peer
host all enterprisedb 127.0.0.1/32 scram-sha-256
host all enterprisedb ::1/128 scram-sha-256
host all all 0.0.0.0/0 ldap ldapserver="{{ldap_
↪host/ip}}" ldapbasedn="{{basedn}}" ldapsearchattribute=uid ldaptls=0 ldapport=389
# Other authentication strings
.
.
.
# End of the file / add ldap SSL connection here

hostssl all all 0.0.0.0/0 ldap ldapserver="{{ldap host/ip}}"
↪ldapbasedn="dc={{basedn}}" ldapsearchattribute=uid ldaptls=0 ldapport=389
```

The following is a sample `pg_hba.conf` file for EDB Postgres Advanced Server without any modifications:

```
# TYPE DATABASE USER ADDRESS METHOD
local all enterprisedb peer
local replication all peer
local all all peer
host all all 127.0.0.1/32 scram-sha-256
host all all ::1/128 scram-sha-256
host replication all 127.0.0.1/32 scram-sha-256
host replication all ::1/128 scram-sha-256
```

(continues on next page)

(continued from previous page)

```
# node1
hostssl all all 10.10.0.5/32 scram-sha-256
hostssl replication all 10.10.0.5/32 scram-sha-256
# ldap

hostssl all all 0.0.0.0/0 scram-sha-256
```

The following is a sample `pg_hba.conf` with the initial configuration and LDAP authentication:

#	TYPE	DATABASE	USER	ADDRESS	METHOD
	local	all	enterprisedb		peer
	host	all	enterprisedb	127.0.0.1/32	scram-sha-256
	host	all	enterprisedb	:::1/128	scram-sha-256
	host	all	all	0.0.0.0/0	ldap ldapserver="ldap" ↵
			↵ldapbasedn="dc=planetexpress,dc=com" ldapsearchattribute=uid		ldaptls=0 ldapport=10389
	local	replication	all		peer
	local	all	all		peer
	host	all	all	127.0.0.1/32	scram-sha-256
	host	all	all	:::1/128	scram-sha-256
	host	replication	all	127.0.0.1/32	scram-sha-256
	host	replication	all	:::1/128	scram-sha-256

```
# node1

hostssl all all 172.17.0.2/32 scram-sha-256
hostssl replication all 172.17.0.2/32 scram-sha-256
# ldap

hostssl all all 0.0.0.0/0 ldap ldapserver="ldap" ldapbasedn="dc=planetexpress,dc=com" ↵
↵ldapsearchattribute=uid ldaptls=0 ldapport=10389
```

After setting `pg_hab.conf`, reload the configuration to apply the changes with the Postgres shell or the `systemctl`:

```
SELECT pg_reload_conf();
```

```
systemctl restart postgres
```

Configure ldap2pg

ldap2pg is a PostgreSQL role and access control list (ACL) management tool. Find the `ldap2pg.yml` in its default location in the `/etc/` folder. For more information on ldap2pg, see the [ldap2pg documentation](#).

Install the ldap2pg tool with the following command:

```
yum install python3-ldap2pg
```

For synchronization to work, you need to set up the following sections in the configuration file:

- `ldap`
- `postgres`
- `sync_map`

Verify LDAP server connection

Before configuring ldap2pg, check that the connection string works. For LDAP servers, you can check if the database server can communicate with the LDAP server by installing the `openldap-clients` package:

```
$ yum install -y openldap-clients
```

Test the connection from the database server to the LDAP server with the `ldapsearch` command:

```
$ ldapsearch -H ldap://ldap_server:389 -x -b "<ldap query string>" \
  -D "<admin query string>" -w <admin password> \
  uid=alphanum_user_id
```

A sample search string looks like this:

```
$ ldapsearch -H ldap://ldap:10389 -x -b "ou=people,dc=planetexpress,dc=com" \
  -D "cn=admin,dc=planetexpress,dc=com" -w GoodNewsEveryone \
  uid=professor
```

A successful request returns an LDAP object definition as a plain-text response like the following:

```
# extended LDIF
#
# LDAPv3
# base <ou=people,dc=planetexpress,dc=com> with scope subtree
# filter: uid=professor
# requesting: ALL
#
# Hubert J. Farnsworth, people, planetexpress.com

dn: cn=Hubert J. Farnsworth,ou=people,dc=planetexpress,dc=com
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
objectClass: top
cn: Hubert J. Farnsworth
```

ldap

The LDAP configuration contains the server port and url. Depending on the type of server and its configuration, you might also need to provide credentials.

OpenLDAP configuration

For servers that allow anonymous connections, the LDAP configuration requires a LDAP server url and port. For servers that need authentication, you must provide `password` and `binddn` queries. To do this, `binddn` must have permission to perform user lookups.

The default port for LDAP is 389:

```
ldap:  
  uri: ldap://ldap:389
```

Active Directory configuration

The Active Directory setup requires a `binddn`, `password`, and `uri` with permission to read user information.

```
ldap:  
  binddn: cn=ldap2pg,ou=postgresql,dc=edb,dc=com  
  password: EDBad32  
  uri: ldap://10.100.20.21:389
```

postgres

The Postgres configuration contains the PostgreSQL database settings, such as authentication and the specifics of schemas and queries, which populate user information.

You can perform database authentication with the `dsn` string, which supports any authentication methods available in `libpq`.

Peer authentication

To use peer authentication for EDB Postgres Advanced Server, you need to list only the database name under `dbanme`. Enter the following string:

```
dsn: dbname=postgres
```

To run the sync of `ldap2pg` as a default peer user for EDB Postgres Advanced Server, set the file permission for `/etc/ldap2pg.yml` to the local database user. For example:

```
$ chown enterprisedb:enterprisedb /etc/ldap2pg.yml
```

Pass a `-v` parameter to verify that the user sync process for `ldap2pg` on the database server runs:

```
$ ldap2pg -v
```

Using Pgpss

EDB Postgres Advanced Server creates the `.pgpass` authentication file in `/var/lib/edb/` with the default user `enterprisedb`. If you plan to run synchronization jobs as the `enterprisedb` user, it's easier to use peer authentication.

This case examines the scenario where EDB Postgres Advanced Server authentication is performed with the `enterprisedb` user but the sync job for `ldap2pg` is run as a different user.

Copy `.pgpass` from the `enterprisedb` user home directory:

```
$ sudo cp /var/lib/edb/.pgpass ~/
```

Modify the `dsn` variable in the `postgres` section in `/etc/ldap2pg.yaml` :

```
dsn: host=localhost user=enterprisedb dbname=postgres
```

OpenLDAP configuration

This is a sample configuration for synchronizing the PostgreSQL user accounts with OpenLDAP.

`dsn` — A Postgres URI string that uses the `username:password` authentication method.

`databases_query` — Specifies the Postgres database that contains the user records.

`schemas_query` — Returns all schemas in `pg_catalog` to be available to `ldap2pg`.

`roles_blacklist_query` — Excludes tables from synchronization operations, in particular anything related to EDB Postgres Advanced Server or internal Postgres tables.

```
postgres:
  dsn: postgres://enterprisedb:LOCALPASS@localhost:5444/postgres
  databases_query: [postgres]
  schemas_query: "SELECT nspname FROM pg_catalog"
  roles_blacklist_query: [aq_administrator_role, pg_*]
```

Active Directory configuration

This is a sample configuration for synchronizing the PostgreSQL user accounts with Active Directory.

`dsn` — A Postgres URI string that uses the `username:password` authentication method. `managed_roles_query` — Query to restrict role deletion and privilege edition to specific roles. `owners_query` — Specifies a global list of owners common to all databases and schemas. `schemas_query` — Specifies owners for schemas so that `ldap2pg` can be aware of the schemas in each database.

```
postgres:
  dsn: postgres://enterprisedb:LOCALPASS@localhost:5444/postgres
  managed_roles_query: |
    SELECT public
    UNION
    SELECT DISTINCT role.rolname
    FROM pg_roles AS role
    LEFT OUTER JOIN pg_auth_members AS ms ON ms.member = role.oid
    LEFT OUTER JOIN pg_roles AS ldap_roles
```

(continues on next page)

(continued from previous page)

```

    ON ldap_roles.rolname = ldap_roles AND ldap_roles.oid = ms.roleid
WHERE role.rolname IN (ldap_roles, readers, writers, owners)
    OR ldap_roles.oid IS NOT NULL
ORDER BY 1;
owners_query: |
SELECT DISTINCT role.rolname
FROM pg_catalog.pg_roles AS role
JOIN pg_catalog.pg_auth_members AS ms ON ms.member = role.oid
JOIN pg_catalog.pg_roles AS owners
    ON owners.rolname = owners AND owners.oid = ms.roleid
ORDER BY 1;
schemas_query: |
SELECT nspname FROM pg_catalog.pg_namespace
WHERE nspname NOT LIKE pg_% AND nspname <> information_schema
ORDER BY 1;

```

sync_map

Each sync_map item is called a *mapping*. A mapping is a YAML dictionary with a description field and any of ldapsearch, role, and grant subsection.

Roles

Define one or more roles that target the PostgreSQL cluster. Specify the roles under selected LDAP accounts that are imported and the permissions they use.

A value can be a single role or a list. This example is a sample definition of roles under sync_map :

```

roles:
- names:
  - ldap_roles
  - readers
  options: NOLOGIN
- name: writers
  # Grant reading to writers
  parent: readers
  options: NOLOGIN
- name: owners
  # Grant read/write to owners
  parent: writers
  options: NOLOGIN

```

This example is a simple definition of a single role with LOGIN and SUPERUSER privileges applied to every user authenticated with LDAP:

```

role:
  name: {uid}
  options: LOGIN SUPERUSER

```

Grant

The `grant` section defines grant privilege to a role with the corresponding parameters. In other words, this section determines the privileges granted for a role defined in the `roles` section.

```
grant:
- privilege: ro
  role: readers
  schemas: __all__
- privilege: rw
  role: writers
  schema: __all__
- privilege: ddl
  role: owners
  schema: __all__
```

ldapsearch query and filters

The `ldapsearch` section specifies the search queries needed to identify the LDAP accounts that you can synchronize with PostgreSQL. For example:

```
ldapsearch:
  base: ou=people,dc=planetexpress,dc=com
  binddn: ou=people,dc=planetexpress,dc=com
  filter: "(objectClass=inetOrgPerson)"
```

Role statement

The `role` section defines the user accounts and options that can be mapped in the PostgreSQL database. The default name parameter for each user record is `cn`, which is the account's full name. The `uid` name is instead specified, which is a short username.

```
role:
  name: {uid}
  options: LOGIN SUPERUSER
```

Set a synchronization schedule

`schedule_pgagent` sets up a synchronization schedule for `ldap2pg`. This utility uses syntax similar to `cron` and produces the SQL command `pgagent.sql`, which is loaded into the database.

This example creates a schedule that runs every hour:

```
schedule_pgagent "0 * * * *"
```

This command produces the `pgagent.sql` file, which is loaded into the database with `enterprisedb` user credentials:

```
psql < pgagent.sql
```

3.2.4 Using EDB LDAP Sync

Before using EDB LDAP Sync for the first time, we recommend that you verify the syntax and configuration of `ldap2pg` with the `ldap2pg` command. Without any parameters, the utility begins a dry run. The following is an example of a successful syntax and configuration check:

```
ldap2pg
__OUTPUT__
Starting ldap2pg 5.7.
Using /etc/ldap2pg.yml.
Running in dry mode. Postgres will be untouched.
Inspecting roles in Postgres cluster...
Query LDAP to create superusers.
Querying LDAP ou=people,dc=planetexpre... (objectClass...
Would create amy.
Would create bender.
Would create fry.
Would create hermes.
Would create leela.
Would create professor.
Would create zoidberg.
Comparison complete.
```

To manually synchronize credentials, you can run the following command:

```
ldap2pg -N
__OUTPUT__
Using /etc/ldap2pg.yml.
Running in real mode.
Inspecting roles in Postgres cluster...
Query LDAP to create superusers.
Querying LDAP ou=people,dc=planetexpre... (objectClass...
Create amy.
Create bender.
Create fry.
Create hermes.
Create leela.
Create professor.
Create zoidberg.
Synchronization complete.
```

Once EDB LDAP Sync completes synchronization, you can check if it was successful by querying the `pg_user` table. The table should contain all of the users imported from the LDAP server:

```
SELECT * FROM pg_catalog.pg_user;
__OUTPUT__
 username | usesysid | usecreatedb | usesuper | userepl | usebypassrls | passwd |
-----+-----+-----+-----+-----+-----+-----+
 valuntil | useconfig
-----+-----+-----+-----+-----+-----+-----+
 postgres |      10 | t          | t        | t        | t          | ****  **** |
 amy      |    16563 | f          | t        | f        | f          | ****  **** |
```

(continues on next page)

(continued from previous page)

bender	16564	f	t	f	f	****	****	└
↪ fry	16565	f	t	f	f	****	****	└
↪ hermes	16566	f	t	f	f	****	****	└
↪ professor	16567	f	t	f	f	****	****	└
↪ leela	16568	f	t	f	f	****	****	└
↪ zoidberg	16569	f	t	f	f	****	****	└
↪ (8 rows)								

3.3 EDB Postgres Tuner

EDB Postgres Tuner is a PostgreSQL extension that automates 15+ years of EDB Postgres tuning experience.

Postgres uses some conservative settings to cover different host sizes. Some of the settings provided by Postgres are unsuitable because they don't take advantage of the available resources. Configuration parameters set by `initdb` don't account for the amount of memory, the number of CPU cores, and the kind of storage devices available to set appropriate values for parameters. Some parameters depend on the workload. The workload provides metrics to use to fine-tune some parameters dynamically.

This extension provides safe recommendations that maximize the use of available resources. It also allows you to control if and when to apply the changes. EDB Postgres Tuner enables you to apply tuning recommendations automatically or view tuning recommendations and selectively apply them. It's now possible to successfully manage demanding Postgres databases without tuning expertise.

3.3.1 Postgres Tuner release notes

The Postgres Tuner documentation describes the latest version of Postgres Tuner, including minor releases and patches. These release notes cover what was new in each release. For new functionality introduced in a minor or patch release, there are also indicators in the content about the release that introduced the feature.

Version	Release Date
<i>Release notes for Advanced Storage Pack version 1.0.0</i>	2022 Nov 30

Release notes for Postgres Tuner version 1.0.0

This release of Postgres Tuner includes:

Type	Description
Feature	This is the initial release.

3.3.2 Installing EDB Postgres Tuner

EDB Postgres Tuner is supported on the same platforms as the Postgres distribution you're using. Support for EDB Postgres Tuner starts with Postgres 11. For details, see:

- [EDB Postgres Advanced Server Product Compatibility](#)
- [PostgreSQL Product Compatibility](#)
- [EDB Postgres Distributed \(includes EDB Postgres Extended\)](#)

Prerequisites

Before you begin the installation process:

- Install Postgres. See: - [Installing EDB Postgres Advanced Server](#)
- [Installing PostgreSQL](#)
- Installing EDB Postgres Distributed (includes EDB Postgres Extended)
- Set up the repository.

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

To set up the repository, go to [EDB repositories](#) and follow the instructions provided there.

Install the package

The syntax for the RPM package install command is:

```
sudo <package-manager> -y install edb-<postgres><postgres_version>-postgres-tuner<major_<br/>↪version>
```

The syntax for the Debian package install command is:

```
sudo <package-manager> -y install edb-<postgres><postgres_version>-postgres-tuner-<major_<br/>↪version>
```

Where:

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

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

continues on next page

Table 10 – continued from previous page

Package manager	Operating system
apt-get	Debian and derivatives

- `<postgres>` is the distribution of Postgres you're using:

Postgres distribution	Value
PostgreSQL	pg
EDB Postgres Advanced Server	as
EDB Postgres Extended	pgextended

- `<postgres_version>` is the version of Postgres you're using.
- `<major_version>` is the major version of EDB Postgres Tuner you're installing.

For example, to install EDB Postgres Tuner 1.0.0 for EDB Postgres Advanced Server 14 on a RHEL 8 platform:

```
sudo dnf -y install edb-as14-postgres-tuner1
```

To install EDB Postgres Tuner 1.0.0 for EDB Postgres Advanced Server 14 on a Debian 11 platform:

```
sudo apt-get -y install edb-as14-postgres-tuner-1
```

- – title: Configuring EDB Postgres Tuner navTitle: Configuring —

To run EDB Postgres Tuner, you need to add it to the `shared_preload_libraries` and create the extension in the database. EDB Postgres Tuner can then gather metrics to make tuning recommendations.

1. In the `postgresql.conf` file, add `edb_pg_tuner` to the `shared_preload_libraries` parameter:

```
shared_preload_libraries = edb_pg_tuner
```

Note: If `shared_preload_libraries` has other extensions, then you can add `edb_pg_tuner` to the list. The order doesn't matter. !!!

1. Restart Postgres.
2. Create the EDB Postgres Tuner extension in your database with the following command:

```
CREATE EXTENSION edb_pg_tuner;
```

3.3.3 Parameters

The following custom GUCs control the EDB Postgres Tuner extension behavior. If you modify these parameters, then reload Postgres to apply the changes.

- `edb_pg_tuner.autotune` — Applies tuning recommendations automatically. The default is `false`.
- `edb_pg_tuner.naptime` — Sets the interval between each check in seconds. The default is 600 seconds (10 minutes).
- `edb_pg_tuner.max_wal_size_limit` — Sets the maximum value for the `max_wal_size` recommendation. The default is 0, which sets no limit.

3.3.4 Using EDB Postgres Tuner

You can use EDB Postgres Tuner in two ways: to automatically apply all tuning recommendations or to manually apply selected tuning recommendations.

Automatic tuning

To automatically apply all tuning recommendations, set the `edb_pg_tuner.autotune` parameter to `true`. Restart Postgres to apply the change. EDB Postgres Tuner then begins applying tuning recommendations.

Manual tuning

To manually apply selected tuning recommendations, make sure the `edb_pg_tuner.autotune` parameter is set to `false`, which is the default. Then, use the `edb_pg_tuner_recommendations` function to inspect the tuning recommendations from the output.

You can set the format of the tuning recommendations to either `conf` or `sql`. For example:

```
SELECT edb_pg_tuner_recommendations(sql);
```

Where:

- `conf` provides an output like `parameter = value`, which you can copy and paste into your configuration file. This is the default.
- `sql` provides an output like `ALTER SYSTEM SET parameter = value`, which you can execute in your preferred Postgres client tool.

Example

This example shows how to manually apply tuning recommendations to a `postgresql.conf` file.

Here's an example of settings in a `postgresql.conf` file:

```
checkpoint_completion_target = 0.2
effective_cache_size = 4GB
enable_seqscan = off
fsync = off
full_page_writes = off
max_wal_size = 2MB
seq_page_cost = 1.3
shared_buffers = 128MB
```

The following command provides tuning recommendations for the `postgresql.conf` file in the default `conf` format:

```
postgres=# SELECT * FROM edb_pg_tuner_recommendations();
__OUTPUT__
      recommendation
-----
checkpoint_completion_target = 0.9
effective_cache_size = 4653 MB
enable_seqscan = on
fsync = on
```

(continues on next page)

(continued from previous page)

```

full_page_writes = on
max_wal_size = 84MB
maintenance_work_mem = 524 MB
shared_buffers = 1474 MB
(8 rows)

```

The following command provides an output in the sql format:

```

postgres=# SELECT edb_pg_tuner_recommendations(sql);
__OUTPUT__
          edb_pg_tuner_recommendations
-----
ALTER SYSTEM SET checkpoint_completion_target = 0.9;
ALTER SYSTEM SET effective_cache_size = 4653 MB;
ALTER SYSTEM SET enable_seqscan = on;
ALTER SYSTEM SET fsync = on;
ALTER SYSTEM SET full_page_writes = on;
ALTER SYSTEM SET max_wal_size = 84MB;
ALTER SYSTEM SET maintenance_work_mem = 524 MB;
ALTER SYSTEM SET shared_buffers = 1474 MB;
(8 rows)

```

You can copy and paste the tuning recommendations in the conf format into the postgresql.conf file:

```

checkpoint_completion_target = 0.9
effective_cache_size = 4653 MB
enable_seqscan = on
fsync = on
full_page_writes = on
max_wal_size = 84MB
maintenance_work_mem = 524 MB
shared_buffers = 1474 MB

```

You can execute the tuning recommendations in the sql format in your preferred Postgres client tool:

```

ALTER SYSTEM SET checkpoint_completion_target = 0.9;
ALTER SYSTEM SET effective_cache_size = 4653 MB;
ALTER SYSTEM SET enable_seqscan = on;
ALTER SYSTEM SET fsync = on;
ALTER SYSTEM SET full_page_writes = on;
ALTER SYSTEM SET max_wal_size = 84MB;
ALTER SYSTEM SET maintenance_work_mem = 524 MB;
ALTER SYSTEM SET shared_buffers = 1474 MB;

```

Recommended GUCs

EDB Postgres Tuner can recommend the following GUCs. The `static` category provides fixed recommendation settings. The `dynamic` category uses specific algorithms to suggest a better setting according to your workload or hardware resources.

GUC	Category	Recommendation	Version
<code>autovacuum</code>	static	on	
<code>check-point_completion_target</code>	static	0.9	
<code>effective_cache_size</code>	dynamic		
<code>enable_async_append</code>	static	on	
<code>enable_bitmapscan</code>	static	on	
<code>enable_gathermerge</code>	static	on	
<code>enable_group_by_reordering</code>	static	on	
<code>enable_hashagg</code>	static	on	
<code>enable_hashjoin</code>	static	on	
<code>enable_incremental_sort</code>	static	on	13+
<code>enable_indexonlyscan</code>	static	on	
<code>enable_indexscan</code>	static	on	
<code>enable_material</code>	static	on	
<code>enable_memoize</code>	static	on	14+
<code>enable_mergejoin</code>	static	on	
<code>enable_nestloop</code>	static	on	
<code>enable_parallel_append</code>	static	on	11+
<code>enable_parallel_hash</code>	static	on	11+
<code>enable_partition_pruning</code>	static	on	11+
<code>enable_partitionwise_aggregate</code>	static	on	
<code>enable_partitionwise_join</code>	static	on	
<code>enable_seqscan</code>	static	on	
<code>enable_sort</code>	static	on	
<code>enable_tidscan</code>	static	on	
<code>fsync</code>	static	on	
<code>full_page_writes</code>	static	on	
<code>log_checkpoints</code>	static	on	
<code>max_wal_size</code>	dynamic		
<code>maintenance_work_mem</code>	dynamic		
<code>parallel_leader_participation</code>	static	on	
<code>seq_page_cost</code>	static	1.0	
<code>shared_buffers</code>	dynamic		
<code>track_activities</code>	static	on	
<code>track_counts</code>	static	on	
<code>zero_damaged_pages</code>	static	on	