



EDB

Postgres extensions

リリース *1.0.0*

EnterpriseDB Corporation

2022 年 12 月 09 日

目次

第 1 章	オープンソース拡張機能	3
第 2 章	EDB 拡張	5
第 3 章	EDB がサポートするオープンソース拡張機能	7
3.1	EDB Advanced Storage Pack	7
3.1.1	自動クラスター	8
3.1.2	参照データ	8
3.2	EDB LDAP Sync	15
3.2.1	LDAP Sync release notes	16
3.2.2	Installing EDB LDAP Sync	16
3.2.3	Configuring EDB LDAP Sync	17
3.2.4	Using EDB LDAP Sync	26
3.3	EDB Postgres Tuner	27
3.3.1	Postgres Tuner release notes	28
3.3.2	Installing EDB Postgres Tuner	28
3.3.3	パラメーター	30
3.3.4	Using EDB Postgres Tuner	31

拡張機能のカテゴリ：

- [オープンソース拡張機能](#)
- [EDB 拡張](#)
- [EDB がサポートするオープンソース拡張機能](#)

注釈: BigAnimal の場合：

• PostgreSQL Compatible は EDB Postgres Extended Server に対応しています。-Oracle Compatible は EDB Postgres Advanced Server に対応しています。

第1章 オープンソース拡張機能

		Virtual Machines/Physical Servers			Kubernetes		BigAnimal		
Extension name	Requires superuser access	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		✓	✓	✓	✓	✓	✓	✓	✓
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_freezecheck	Yes	✓	✓	✓	✓	✓	✓	✓	✓
pg_prewarm	Yes	✓	✓	✓	✓	✓	✓	✓	✓
pg_stat_statements	Yes	✓	✓	✓	✓	✓	✓	✓	✓
pg_surgery	Yes	✓	✓	✓	✓	✓	✓	✓	✓
pg_trgm		✓	✓	✓	✓	✓	✓	✓	✓
pg_visibility	Yes	✓	✓	✓	✓	✓	✓	✓	✓
pgcrypto	Yes	✓	✓	✓	✓	✓	✓	✓	✓
pgrowlocks	Yes	✓	✓	✓	✓	✓	✓	✓	✓
pgstattuple	Yes	✓	✓	✓	✓	✓	✓	✓	✓
postgis_fdw	Yes	✓	✓	✓	✓	✓	✓	✓	✓
seg		✓	✓	✓	✓	✓	✓	✓	✓
sepgsql		✓	✓	✓	—	—	—	—	—
spi		✓	✓	✓	✓	✓	✓	✓	✓
autoinc	Yes	✓	✓	✓	✓	✓	✓	✓	✓
insert_username	Yes	✓	✓	✓	✓	✓	✓	✓	✓
moddatetime	Yes	✓	✓	✓	✓	✓	✓	✓	✓
refint	Yes	✓	✓	✓	✓	✓	✓	✓	✓
ssinfo	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		✓	—	—	—	—	—	—	—
pgsnmpd (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		✓	—	—	—	—	—	—	—

第2章 EDB 拡張

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		✓	✓	✓	—	—	—	—	—

図 1: EDB extensions

第3章 EDB がサポートするオープンソース拡張機能

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	✓	–	✓	–	–	–	–	✓

図 1: EDB supported open source extensions

3.1 EDB Advanced Storage Pack

EDB Advanced Storage Pack は、PostgreSQL データベースの高度なストレージオプションをテーブルアクセス方法（TAM）拡張機能の形式で提供します。これらのストレージオプションは、アプリケーションを変更することなく、データベースのパフォーマンスと信頼性を向上させます。

アクセスパターンが事前にわかっているテーブルの場合、さまざまなトレードオフを行うターゲット TAM を好む場合があります。たとえば、アプリケーションの特定のテーブルが INSERT 専用で、行が更新をまったく受け取らない場合、このテーブルに INSERT 固有の最適化を持つ専用の TAM の使用を検討できます。

EnterpriseDB は、Advanced Storage Pack で 2 つの TAM を提供します。

3.1.1 自動クラスター

Autocluster TAM は、サイドテーブルの値に最後に挿入された行を追跡することにより、クラスター化されたデータへのより高速なアクセスを提供します。次に、データのクラスター化を維持しながら、以前の行と同じデータブロックに新しい行を追加できます。このアプローチにより、関連データへのアクセス時間が短縮されます。この機能を使用するには、特定のキーのインデックススキャンで近くにあるすべての行を検出できるように、同じキー値を持つ行をクラスター化したままにします。スキャンでは、クエリを満たすために多くのテーブルページを取得する必要はありません。

3.1.2 参照データ

Refdata TAM は、ほとんどの場合静的データ用に最適化されています。これには、時折の INSERT と非常に少数の DELETE および UPDATE コマンドが含まれます。外部キーを使用してデータを参照するデータベーススキーマの場合、この TAM により、5~10%のパフォーマンスが向上し、スケラビリティが向上します。この機能を使用するには、参照テーブルを変更するたびに排他ロックを取得し、他のセッションによる同時変更と、そのテーブルを参照するテーブルへの変更をブロックします。例：

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

employee テーブルは標準のヒープテーブルです。department テーブルのみが refdata TAM を使用します。従業員テーブルの挿入と更新では、部門テーブルの行レベルのロックが取得されないため、クエリ時間が節約されます。このアプローチにより、部門テーブルの行を更新する必要がなくなり、参照される部門テーブルの行をディスクおよびログ先行書き込みに書き込む必要もなくなります。

department テーブルへの更新が頻繁に行われる場合、それと employee テーブルへの同時変更がブロックされるため、Refdata TAM を使用しないことをお勧めします。department テーブルに頻繁に変更を加えない場合、従業員テーブルへの頻繁な変更を高速化し、ログ先行書き込みトラフィックを削減することはこのコストに見合うかもしれません。

Advanced Storage Pack release notes

Advanced Storage Pack のドキュメントには、マイナーリリースとパッチを含む Advanced Storage Pack の最新バージョンが説明されています。これらのリリースノートでは、各リリースの新機能について説明します。マイナーリリースまたはパッチリリースで導入された新機能については、その機能を導入したリリースに関するコンテンツにもインジケータがあります。

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

Advanced Storage Pack のこのリリースには以下が含まれます。

Type	Description
Feature	これは初期リリースであり、Refdata と Autocluster TAM が含まれています。

Installing Advanced Storage Pack

Advanced Storage Pack は、使用している Postgres ディストリビューションと同じプラットフォームでサポートされています。Advanced Storage Pack のサポートは Postgres 11 から始まります。詳細については、次を参照してください。

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

前提条件

インストールプロセスを開始する前に：

- Postgres をインストールします。参照： - [Installing EDB Postgres Advanced Server](#)
- [Installing PostgreSQL](#)
- [Installing EDB Postgres Distributed \(includes EDB Postgres Extended\)](#)
- リポジトリをセットアップします。

リポジトリの設定は 1 回限りのタスクです。リポジトリを既に設定している場合、この手順を実行する必要はありません。

リポジトリをセットアップするには、

[EDB repositories](#) に移動し、そこに表示される指示に従います。

パッケージをインストールする

RPM パッケージインストールコマンドの構文は次のとおりです。

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

Debian パッケージのインストールコマンドの構文は次のとおりです。

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

そこで：

- <package-manager> は、オペレーティングシステムで使用するパッケージマネージャーです。

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

- <postgres> は、使用している Postgres のディストリビューションです。

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

- <postgres_version> は、使用している Postgres のバージョンです。
- <major_version> は、インストールする Advanced Storage Pack のメジャーバージョンです。

たとえば、RHEL 8 プラットフォームに EDB Postgres Advanced Server 14 の Advanced Storage Pack 1.0.0 をインストールするには：

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

Debian 11 プラットフォームに EDB Postgres Advanced Server 14 用の Advanced Storage Pack 1.0.0 をインストールするには：

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

Configuring Advanced Storage Pack

Postgres の起動時に早くロードされるように、カスタム TAM を実装する拡張モジュールを `shared_preload_libraries` に配置します。この手順は、指定された TAM に基づいたテーブルに初めてアクセスする前に拡張機能を使用できるようにするために必要です。たとえば、`postgresql.conf` のパラメーターを `autocluster` または `refdata` で更新します。

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

サーバー再起動後、SQL コマンドを実行して拡張機能を作成します。このコマンドは、SQL が実行される接続されたデータベースにのみ拡張機能を作成します。拡張機能が使用される各データベースで再実行する必要があります。

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

拡張機能が作成されたデータベースでは、拡張機能が提供する TAM を使用するテーブルを作成できます。

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

Using EDB Advanced Storage Pack

以下は、EDB Advances Storage Pack TAM が役立つシナリオです。

Refdata の例

Refdata が役立つシナリオは、ニューヨーク証券取引所 (NYSE) のすべての銘柄記号とその企業名の参照テーブルを作成する場合です。このデータはめったに変更されないことが予想され、市場全体のすべての株式取引を追跡するテーブルから頻繁に参照されます (高度なオートクラスターの例 など)。ヒープの代わりに Refdata

を使用して、パフォーマンスを向上させることができます。

```
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;
`` # 自動クラスターの例
```

自動クラスターが役立つシナリオは、モノのインターネット (IoT) データの場合です。自動クラスターの代わりにヒープを使用する場合、Postgres はこれらの関連する行をクラスター化できないため、行のセットへのアクセスは多くのデータブロックに触れ、非常に遅くなる可能性があります、入出力が重いです。

この例は、60 秒ごとに家の温度と温度設定を報告する IoT サーモスタット用です。

(次のページに続く)

(前のページからの続き)

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

Autocluster を使用すると、同じ `thermostat_id` を持つ行がクラスター化され、アクセスが簡単になります。

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

**注釈:** `cols` パラメーターは、`USING btree()` で指定された列数と一致する必要があります。この場合、`thermostat_id` のみがリストされているため、値は `{1}` です。!!!

テーブルに `thermostat_id` および `recordtime` データを入力します。

```
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);
```

IoT テーブルからデータを選択すると、`ctid` の場所から、同じ `thermostat_id` を持つデータがクラスター化されていることがわかります。

```
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)
```

## 高度なオートクラスターの例

これは、前の例よりも高度な Autocluster の使用方法です。これには、

*Refdata の例* から NYSE テーブルを参照し、株式記号に基づいて行をクラスター化することが含まれます。このアプローチにより、最新の取引数を簡単に見つけることができます。

Refdata の例の NYSE テーブルから始めます。

```
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;
```

Refdata テーブル内のほとんど静的な株式銘柄を参照し、NYSE の取引を含む高度に更新されたテーブルを作成します。銘柄記号の行をクラスター化して、特定の株の最後の x 回の取引を簡単に検索できるようにします。

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

ID ではなくシンボル名で挿入しやすいビューを作成します。

```
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
```

(次のページに続く)

(前のページからの続き)

```
WHERE ns.symbol = NEW.symbol
);
```

ビューの作成の詳細については、[PostgreSQL documentation](#) を参照してください。

静的データを事前に入力します（簡潔にするために短縮されています）。

```
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;
```

2022 年 11 月 18 日金曜日の指定された時間範囲の株式取引を挿入します（短く省略されています）。

```
\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
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
```

(次のページに続く)

(前のページからの続き)

```

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

```

特定の銘柄記号のデータから ctid を選択して、出力でどのようにクラスター化されたかを確認します。

```

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 は、Postgres を OpenLDAP や Active Directory などの LDAP プロトコルをサポートするディレクトリサーバーと同期するためのユーティリティと構成のコレクションです。データベース管理者は、Postgres のユーザーおよびグループを LDAP ユーザーおよびグループと同期するように ldap2pg をスケジュールできます。EDB LDAP Sync は、同様の構文を使用しながら、cron などの外部スケジューラを置き換えます。

### 3.2.1 LDAP Sync release notes

LDAP Sync のドキュメントには、マイナーリリースとパッチを含む LDAP Sync の最新バージョンが記載されています。これらのリリースノートでは、各リリースの新機能について説明します。マイナーリリースまたはパッチリリースで導入された新機能については、その機能を導入したリリースに関するコンテンツにもインジケータがあります。

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

このリリースの LDAP Sync には以下が含まれます。

Type	Description
Feature	これは、最初のリリースです。

### 3.2.2 Installing EDB LDAP Sync

EDB LDAP Sync は、使用している Postgres ディストリビューションと同じプラットフォームでサポートされています。EDB LDAP Sync のサポートは Postgres 11 から始まります。参照：

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

#### 前提条件

インストールプロセスを開始する前に：

- Postgres をインストールします。参照： - [Installing EDB Postgres Advanced Server](#)
- [Installing PostgreSQL](#)
- [Installing EDB Postgres Distributed \(includes EDB Postgres Extended\)](#)
- [ldap2pg の設定](#) をインストールします。
- リポジトリをセットアップします。

リポジトリの設定は 1 回限りのタスクです。リポジトリを既に設定している場合、この手順を実行する必要はありません。

リポジトリをセットアップするには、

EDB repositories に移動し、そこに表示される指示に従います。

### パッケージをインストールする

RPM パッケージインストールコマンドの構文は次のとおりです。

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

Debian パッケージのインストールコマンドの構文は次のとおりです。

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

そこで：

- <package-manager> は、オペレーティングシステムで使用するパッケージマネージャーです。

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> は、インストールする EDB LDAP Sync のメジャーバージョンです。

たとえば、RHEL 8 プラットフォームに EDB LDAP Sync バージョン 1.0.0 をインストールするには：

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

Debian 11 プラットフォームに EDB LDAP Sync バージョン 1.0.0 をインストールするには：

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

### 3.2.3 Configuring EDB LDAP Sync

EDB LDAP Sync は、LDAP または Active Directory サーバーから EDB Postgres Advanced Server に資格情報を同期するツールのコレクションです。次の方法でツールを構成することをお勧めします。

/usr/lib/python3.6/site-packages/ldap\_sync/samples/ でサンプルの EDB LDAP Sync 構成ファイルを見つけます。

## pgAgent の設定

pgAgent は定期的な同期タスクを設定します。Postgres での pgAgent セットアップのデフォルト構成として `edb-pgagent.conf_tmpl` を使用します。Postgres で pgAgent の構成ディレクトリにコピーします。

詳細については、[pgAgent documentation](#) を参照してください。

## Postgres データベースの権限の設定

Postgres が Postgres 認証中に LDAP ホストを照会できるように、データベースサーバーの権限設定を変更する必要があります。これを行うには、データベースサーバーの構成フォルダーにある `pg_hba.conf` を更新します。EDB Postgres Advanced Server 構成の場合、`pg_hba.conf` は `/opt/postgres/data/pg_hba.conf` にあります。

### pg\_hba.conf 設定

**注釈:** ユーザーはデータベースサービスによって異なります。PostgreSQL には `postgres` ユーザーがいます。EDB Postgres Advanced Server には ``enterprisedb`` ユーザーがいます。!!!

`pg_hba.conf` を構成する場合、LDAP 認証文字列は `enterprisedb` ユーザー認証文字列の後に置く必要があります。そうしないと、評価の順序のために、`enterprisedb` ユーザーはシステムにログインできません。たとえば、ローカル認証の代わりに LDAP サーバーが照会され、LDAP サーバーが認証の失敗を返し、認証プロセスが終了します。このため、`ldap2pg` セットアップの一部としてホスト認証を使用する場合は、`enterprisedb` ユーザーのホスト認証を追加する必要があります。

次の認証方法を使用できます。ピア、`dsn` オプションを使用したパスワード文字列を持つホスト、および `pgpass` を読み取るホスト。詳細については、[ldap2pg の設定](#) を参照してください。

Postgres データベースの権限は、データベース要件に従って設定されます。以下は、LDAP 認証クエリーのサンプル `pg_hba.conf` スタンザです。{} の値は、LDAP サーバーのホスト名と basen LDAP クエリー文字列です。

#	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				
					.
					.

(次のページに続く)

(前のページからの続き)

```
.
End of the file / add ldap SSL connection here

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

以下は、変更していない EDB Postgres Advanced Server のサンプル pg\_hba.conf ファイルです。

```
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
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
```

以下は、初期構成と LDAP 認証を使用したサンプル pg\_hba.conf です。

```
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"
ldappedn="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
```

(次のページに続く)

(前のページからの続き)

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

pg\_hab.conf を設定した後、構成をリロードして、Postgres シェルまたは systemctl で変更を適用します。

```
SELECT pg_reload_conf();
```

```
systemctl restart postgres
```

### ldap2pg の設定

ldap2pg は、PostgreSQL のロールとアクセス制御リスト (ACL) の管理ツールです。/etc/ フォルダのデフォルトの場所で ldap2pg.yml を見つけます。ldap2pg の詳細については、[ldpa2pg documentation](#) を参照してください。

次のコマンドを使用して ldap2pg ツールをインストールします。

```
yum install python3-ldap2pg
```

同期を機能させるには、構成ファイルに次のセクションを設定する必要があります。

- ldap
- postgres
- sync\_map

### LDAP サーバー接続を確認する

ldap2pg を構成する前に、接続文字列が機能することを確認します。LDAP サーバーの場合、openldap-clients パッケージをインストールして、データベースサーバーが LDAP サーバーと通信できるかどうかを確認できます。

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

ldapsearch コマンドを使用して、データベースサーバーから LDAP サーバーへの接続をテストします。

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

サンプルの検索文字列は次のようになります。

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

要求が成功すると、次のようなプレーンテキストの応答として LDAP オブジェクト定義が返されます。

```
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

LDAP 構成には、サーバーポートと url が含まれています。サーバーの種類とその構成によっては、資格情報の提供が必要になる場合があります。

## OpenLDAP の設定

匿名接続を許可するサーバーの場合、LDAP 構成には LDAP サーバーの URL とポートが必要です。認証が必要なサーバーの場合、password および binddn クエリを提供する必要があります。これを行うには、binddn にユーザー検索を実行する権限が必要です。

LDAP のデフォルトポートは 389 です。

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

### Active Directory 構成

Active Directory のセットアップには、ユーザー情報の読み取り権限を持つ `binddn`、`password`、および `uri` が必要です。

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

### postgres

Postgres 構成には、認証、スキーマとクエリの詳細など、ユーザー情報を入力する PostgreSQL データベース設定が含まれています。

`libpq` で使用可能な認証方法をサポートする `dsn` 文字列を使用して、データベース認証を実行できます。

### ピア認証

EDB Postgres Advanced Server のピア認証を使用するには、`dbanme` の下にデータベース名のみをリストする必要があります。次の文字列を入力します。

```
dsn: dbname=postgres
```

EDB Postgres Advanced Server のデフォルトのピアユーザーとして `ldap2pg` の同期を実行するには、`/etc/ldap2pg.yml` のファイル権限をローカルデータベースユーザーに設定します。例：

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

`-v` パラメーターを渡して、データベースサーバーで `ldap2pg` のユーザー同期プロセスが実行されていることを確認します。

```
$ ldap2pg -v
```

### Pgpass の使用

EDB Postgres Advanced Server は、デフォルトユーザー `enterprisedb` で `/var/lib/edb/` に `.pgpass` 認証ファイルを作成します。`enterprisedb` ユーザーとして同期ジョブを実行する予定がある場合は、ピア認証を使用する方が簡単です。

このケースでは、EDB Postgres Advanced Server 認証は `enterprisedb` ユーザーで実行されますが、`ldap2pg` の同期ジョブは別のユーザーとして実行されます。

`enterprisedb` ユーザーのホームディレクトリから `.pgpass` をコピーします。

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

/etc/ldap2pg.yaml の postgres セクションの dsn 変数を変更します。

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

## OpenLDAP の設定

これは、PostgreSQL ユーザーアカウントを OpenLDAP と同期するためのサンプル構成です。

dsn — username:password 認証方法を使用する Postgres URI 文字列。

databases\_query — ユーザーレコードを含む Postgres データベースを指定します。

schemas\_query — ldap2pg で使用できるように pg\_catalog 内のすべてのスキーマを返します。

roles\_blacklist\_query — 同期操作からテーブル、特に EDB Postgres Advanced Server または内部 Postgres テーブルに関連するものを除外します。

```
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 構成

これは、PostgreSQL ユーザーアカウントを Active Directory と同期するためのサンプル構成です。

dsn — username:password 認証方法を使用する Postgres URI 文字列。managed\_roles\_query — ロールの削除と特権の編集を特定のロールに制限するためのクエリ。owners\_query — すべてのデータベースとスキーマに共通の所有者のグローバルリストを指定します。schemas\_query — ldap2pg が各データベースのスキーマを認識できるように、スキーマの所有者を指定します。

```
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
 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
```

(次のページに続く)

(前のページからの続き)

```
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

各 sync\_map アイテムはマッピングと呼ばれます。マッピングは、description フィールドと、ldapsearch、role、および grant subsection のいずれかを持つ YAML 辞書です。

### ロール

PostgreSQL クラスターをターゲットとする 1 つ以上のロールを定義します。インポートされる選択した LDAP アカウントの下のロールと、それらが使用する権限を指定します。

値は単一のロールまたはリストです。この例は、sync\_map の下のロールの定義のサンプルです。

```
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
```

この例は、LDAP で認証されるすべてのユーザーに適用される LOGIN および SUPERUSER 特権を持つ単一のロールの単純な定義です。

```
role:
 name: {uid}
```

(次のページに続く)

(前のページからの続き)

**options:** LOGIN SUPERUSER

## 付与

**grant** セクションは、対応するパラメーターを使用してロールへの付与特権を定義します。言い換えれば、このセクションは、**roles** セクションで定義されたロールに付与される特権を決定します。

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

## ldapsearch クエリとフィルター

**ldapsearch** セクションは、PostgreSQL と同期できる LDAP アカウントを識別するために必要な検索クエリを指定します。例：

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

## ロールステートメント

**role** セクションは、PostgreSQL データベースにマッピングできるユーザーアカウントとオプションを定義します。各ユーザーレコードのデフォルトの **name** パラメーターは、アカウントのフルネームです。代わりに、短いユーザー名である **uid** 名が指定されます。

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

### 同期スケジュールを設定する

`schedule_pgagent` は、`ldap2pg` の同期スケジュールを設定します。このユーティリティは、`cron` に似た構文を使用して、データベースにロードされる SQL コマンド `pgagent.sql` を生成します。

この例では、毎時実行されるスケジュールを作成します。

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

このコマンドは、`pgagent.sql` ファイルを生成します。これは、`enterprisedb` ユーザー資格情報でデータベースにロードされます。

```
psql < pgagent.sql
```

## 3.2.4 Using EDB LDAP Sync

EDB LDAP Sync を初めて使用する前に、`ldap2pg` コマンドで `ldap2pg` の構文と構成を確認することをお勧めします。パラメーターを指定しないと、ユーティリティは予行演習を開始します。以下は、構文と構成のチェックが成功した例です。

```
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=planetexp... (objectClass...
Would create amy.
Would create bender.
Would create fry.
Would create hermes.
Would create leela.
Would create professor.
Would create zoidberg.
Comparison complete.
```

資格情報を手動で同期するには、次のコマンドを実行します。

```
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=planetexp... (objectClass...
```

(次のページに続く)

(前のページからの続き)

```

Create amy.
Create bender.
Create fry.
Create hermes.
Create leela.
Create professor.
Create zoidberg.
Synchronization complete.

```

EDB LDAP Sync が同期を完了したら、`pg_user` テーブルを照会して、成功したかどうかを確認できます。テーブルには、LDAP サーバーからインポートされたすべてのユーザーが含まれている必要があります。

```
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	**** *
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 は、15 年以上の EDB Postgres チューニングの経験を自動化する PostgreSQL 拡張機能です。

Postgres は、いくつかの保守的な設定を使用して、さまざまなホストサイズに対応します。Postgres が提供する設定の一部は、利用可能なリソースを活用しないため、不適切です。`initdb` によって設定される構成パラメーターは、メモリ量、CPU コアの数、およびパラメーターに適切な値を設定するために使用可能な

ストレージデバイスの種類を考慮していません。一部のパラメーターはワークロードによって異なります。ワークロードは、一部のパラメーターを動的に微調整するために使用するメトリックを提供します。

この拡張機能は、使用可能なリソースを最大限に活用する安全な推奨事項を提供します。変更を適用するかどうかと、いつ適用するかを制御できます。EDB Postgres Tuner を使用すると、チューニングの推奨事項を自動的に適用したり、チューニングの推奨事項を表示して選択的に適用したりできます。チューニングの専門知識がなくても、要求の高い Postgres データベースを正常に管理できるようになりました。

### 3.3.1 Postgres Tuner release notes

Postgres Tuner のドキュメントには、マイナーリリースとパッチを含む Postgres Tuner の最新バージョンが記載されています。これらのリリースノートでは、各リリースの新機能について説明します。マイナーリリースまたはパッチリリースで導入された新機能については、その機能を導入したリリースに関するコンテンツにもインジケータがあります。

Version	Release Date
<a href="#">Release notes for Advanced Storage Pack version 1.0.0</a>	2022 Nov 30

#### Release notes for Postgres Tuner version 1.0.0

Postgres Tuner のこのリリースには以下が含まれます。

Type	Description
Feature	これは、最初のリリースです。

### 3.3.2 Installing EDB Postgres Tuner

EDB Postgres Tuner は、使用している Postgres ディストリビューションと同じプラットフォームでサポートされています。EDB Postgres Tuner のサポートは Postgres 11 から始まります。詳細については、以下を参照してください。

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

## 前提条件

インストールプロセスを開始する前に：

- Postgres をインストールします。参照： - Installing EDB Postgres Advanced Server
- [Installing PostgreSQL](#)
- Installing EDB Postgres Distributed (includes EDB Postgres Extended)
- リポジトリをセットアップします。

リポジトリの設定は 1 回限りのタスクです。リポジトリを既に設定している場合、この手順を実行する必要はありません。

リポジトリをセットアップするには、

[EDB repositories](#) に移動し、そこに表示される指示に従います。

## パッケージをインストールする

RPM パッケージインストールコマンドの構文は次のとおりです。

```
sudo <package-manager> -y install edb-<postgres><postgres_version>-postgres-tuner
→<major_version>
```

Debian パッケージのインストールコマンドの構文は次のとおりです。

```
sudo <package-manager> -y install edb-<postgres><postgres_version>-postgres-tuner-
→<major_version>
```

そこで：

- <package-manager> は、オペレーティングシステムで使用するパッケージマネージャーです。

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

- <postgres> は、使用している Postgres のディストリビューションです。

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

- <postgres\_version> は、使用している Postgres のバージョンです。

- <major\_version> は、インストールする EDB Postgres Tuner のメジャーバージョンです。

たとえば、RHEL 8 プラットフォームに EDB Postgres Advanced Server 14 用の EDB EDB Postgres Tuner 1.0.0 をインストールするには：

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

Debian 11 プラットフォームに EDB Postgres Advanced Server 14 用の EDB EDB Postgres Tuner 1.0.0 をインストールするには：

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

- – title: EDB Postgres Tuner の構成 navTitle: 構成 –

EDB Postgres Tuner を実行するには、それを `shared_preload_libraries` に追加し、データベースに拡張機能を作成する必要があります。EDB Postgres Tuner は、メトリックを収集してチューニングの推奨事項を作成できます。

1. `postgresql.conf` ファイルで、`edb_pg_tuner` を `shared_preload_libraries` パラメーターに追加します。

```
shared_preload_libraries = edb_pg_tuner
```

---

**注釈:** `shared_preload_libraries` に他の拡張子がある場合、`edb_pg_tuner` をリストに追加できます。順番は関係ありません。!!!

---

1. Postgres を再起動します。

1. 次のコマンドを使用して、データベースに EDB Postgres Tuner 拡張機能を作成します。

```
CREATE EXTENSION edb_pg_tuner;
```

### 3.3.3 パラメーター

次のカスタム GUC は、EDB Postgres Tuner 拡張機能の動作を制御します。これらのパラメーターを変更する場合は、Postgres をリロードして変更を適用します。

- `edb_pg_tuner.autotune` —チューニングの推奨事項を自動的に適用します。デフォルトは `false` です。
- `edb_pg_tuner.naptime` —各チェックの間隔を秒単位で設定します。デフォルトは 600 秒（10 分）です。
- `edb_pg_tuner.max_wal_size_limit` — `max_wal_size` 推奨の最大値を設定します。デフォルトは 0 で、制限は設定されません。

### 3.3.4 Using EDB Postgres Tuner

EDB Postgres Tuner は 2 つの方法で使用できます。すべてのチューニングの推奨事項を自動的に適用するか、選択したチューニングの推奨事項を手動で適用します。

#### 自動チューニング

すべてのチューニングの推奨事項を自動的に適用するには、`edb_pg_tuner.autotune` パラメーターを `true` に設定します。Postgres を再起動して変更を適用します。EDB Postgres Tuner は、チューニングの推奨事項の適用を開始します。

#### マニュアルチューニング

選択したチューニングの推奨事項を手動で適用するには、`edb_pg_tuner.autotune` パラメーターがデフォルトの `false` に設定されていることを確認します。次に、`edb_pg_tuner_recommendations` 関数を使用して、出力からチューニングの推奨事項を検査します。

チューニングの推奨事項の形式を `conf` または `sql` に設定できます。例：

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

そこで：

- `conf` は、`parameter = value` のような出力を提供し、構成ファイルにコピーして貼り付けることができます。これがデフォルトです。
- `sql` は、好みの Postgres クライアントツールで実行できる `ALTER SYSTEM SET parameter = value` のような出力を提供します。

#### 例

この例では、チューニングの推奨事項を `postgresql.conf` ファイルに手動で適用する方法を示します。

`postgresql.conf` ファイルの設定例を次に示します。

```
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
```

次のコマンドは、デフォルトの `conf` 形式で `postgresql.conf` ファイルのチューニングの推奨事項を提供します。

```
postgres=# SELECT * FROM edb_pg_tuner_recommendations();
__OUTPUT__
 recommendation

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
(8 rows)
```

次のコマンドは、sql 形式で出力を提供します。

```
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)
```

conf 形式のチューニングの推奨事項をコピーして、postgresql.conf ファイルに貼り付けることができます。

```
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
```

好みの Postgres クライアントツールで sql 形式でチューニングの推奨事項を実行できます。

```
ALTER SYSTEM SET checkpoint_completion_target = 0.9;
ALTER SYSTEM SET effective_cache_size = 4653 MB;
```

(次のページに続く)

(前のページからの続き)

```

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;

```

## 推奨 GUC

EDB Postgres Tuner は、次の GUC を推奨できます。 **static** カテゴリは、固定の推奨設定を提供します。 **dynamic** カテゴリは、特定のアルゴリズムを使用して、ワークロードまたはハードウェアリソースに従ってより良い設定を提案します。

GUC	Category	Recommendation	Version
autovacuum	static	on	
check-point_completion_target	static	0.9	
effective_cache_size	dynamic		
enable_async_append	static	on	
enable_bitmapscan	static	on	
enable_gathermerge	static	on	
enable_group_by_reordering	static	on	
enable_hashagg	static	on	
enable_hashjoin	static	on	
enable_incremental_sort	static	on	13+
enable_indexonlyscan	static	on	
enable_indexscan	static	on	
enable_material	static	on	
enable_memoize	static	on	14+
enable_mergejoin	static	on	
enable_nestloop	static	on	
enable_parallel_append	static	on	11+
enable_parallel_hash	static	on	11+
enable_partition_pruning	static	on	11+
enable_partitionwise_aggregate	static	on	
enable_partitionwise_join	static	on	
enable_seqscan	static	on	
enable_sort	static	on	
enable_tidscan	static	on	

次のページに続く

表 12 – 前のページからの続き

GUC	Category	Recommendation	Version
fsync	static	on	
full_page_writes	static	on	
log_checkpoints	static	on	
max_wal_size	dynamic		
maintenance_work_mem	dynamic		
parallel_leader_participation	static	on	
seq_page_cost	static	1.0	
shared_buffers	dynamic		
track_activities	static	on	
track_counts	static	on	
zero_damaged_pages	static	on	