



EDB

EDB Postgres Advanced Server Guide

リリース 15.2.0

EnterpriseDB Corporation

2023 年 02 月 17 日

目次

第 1 章 Introduction	3
1.1 ハードリミット	3
1.1.1 Conventions used	4
1.1.2 About the examples	5
第 2 章 Enhanced compatibility features	17
2.1 互換性機能を有効にする	17
2.2 ストアドプロシージャ言語	17
2.3 オプティマイザのヒント	18
2.4 データディクショナリビュー	18
2.5 dblink_ora	18
2.6 プロファイル管理	18
2.7 ビルトインパッケージ	19
2.8 Open Client ライブラリ	21
2.9 ユーティリティ	22
2.9.1 EDB*Plus	22
2.9.2 EDB*Loader	22
2.9.3 EDB*ラップ	23
2.9.4 動的ランタイム計測ツールアーキテクチャ (DRITA)	23
2.10 ECPGPlus	23
2.11 テーブルのパーティショニング	24
第 3 章 Database administration	25
3.1 Configuration parameters	25
3.1.1 Setting configuration parameters	25
3.1.2 Summary of configuration parameters	28
3.1.3 Configuration parameters by functionality	56
3.2 Index Advisor	101
3.2.1 Index Advisor components	102
3.2.2 Index Advisor configuration	103
3.2.3 Using Index Advisor	105
3.2.4 Reviewing the Index Advisor recommendations	109
3.2.5 Index Advisor limitations	113
3.3 SQL Profiler	114
3.3.1 トラブルシューティング	115
3.4 pgsnmpd	116
3.4.1 pgsnmpd の設定	116
3.4.2 リスナーアドレスの設定	117
3.4.3 pgsnmpd の呼び出し	117
3.4.4 pgsnmpd ヘルプの表示	118

3.4.5	pgsnmpd に情報を要求する	118
3.5	EDB audit logging	119
3.5.1	Audit logging configuration parameters	119
3.5.2	Selecting SQL statements to audit	122
3.5.3	Enabling audit logging	137
3.5.4	Audit log file	143
3.5.5	Using error codes to filter audit logs	150
3.5.6	Using command tags to filter audit logs	151
3.5.7	Redacting passwords in audit logs	152
3.5.8	Audit log archiving	154
3.5.9	Object auditing	157
3.6	Unicode collation algorithm	163
3.6.1	基本的な Unicode 照合アルゴリズムの概念	164
3.6.2	Unicode の国際コンポーネント	164
3.6.3	照合の使用	167
第 4 章	EDB Resource Manager	173
4.1	リソースグループの作成と管理	174
4.1.1	リソースグループを作成する	174
4.1.2	説明	174
4.1.3	パラメーター	174
4.1.4	例	174
4.1.5	ALTER リソースグループ	175
4.1.6	説明	175
4.1.7	パラメーター	176
4.1.8	例	176
4.1.9	リソースグループをドロップ	177
4.1.10	説明	177
4.1.11	パラメーター	177
4.1.12	例	177
4.1.13	リソースグループへのプロセスの割り当て	177
4.1.14	リソースグループからのプロセスの削除	178
4.1.15	リソースグループのプロセスの監視	179
4.2	CPU 使用率の調整	180
4.2.1	リソースグループの CPU レート制限の設定	181
4.2.2	例：単一グループ内の単一プロセス	182
4.2.3	例：単一グループ内の複数のプロセス	183
4.2.4	例：複数のグループの複数のプロセス	185
4.3	ダーティバッファスロットリング	188
4.3.1	リソースグループのダーティレート制限の設定	188
4.3.2	例：単一グループ内の単一プロセス	189
4.3.3	例：単一グループ内の複数のプロセス	192
4.3.4	例：複数のグループの複数のプロセス	194
4.4	システムカタログ	197
4.4.1	edb_all_resource_groups	197
4.4.2	edb_resource_group	198

第 5 章	libpq C library	199
5.1	EDB SPL で libpq を使用する	199
5.2	REFCURSOR のサポート	199
5.3	単一の REFCURSOR を返す	200
5.3.1	複数の REFCURSOR を返す	203
5.4	配列バインディング	206
5.4.1	PQBulkStart	206
5.4.2	API 定義	207
5.4.3	PQexecBulk	207
5.4.4	API 定義	207
5.4.5	PQBulkFinish	207
5.4.6	API 定義	207
5.4.7	PQexecBulk 準備済み	207
5.4.8	API 定義	208
5.4.9	コード例 (PQBulkStart、PQexecBulk、PQBulkFinish を使用)	208
5.4.10	コード例 (PQexecBulkPrepared を使用)	209
第 6 章	Debugger	211
6.1	デバッガーの構成	212
6.2	デバッガーの起動	212
6.3	デバッガーウィンドウ	213
6.4	メインデバッガーウィンドウ	215
6.4.1	プログラム本体パネル	215
6.4.2	タブパネル	216
6.4.3	スタックタブ	219
6.5	プログラムのデバッグ	221
6.5.1	コードのステップ実行	221
6.5.2	ブレークポイントの使用	221
6.5.3	ローカルブレークポイントの削除	223
6.5.4	インコンテキストデバッグのグローバルブレークポイントを設定する	223
6.5.5	デバッガーの終了	226
第 7 章	Performance analysis and tuning	229
7.1	ダイナチューン	229
7.1.1	edb_dynatune	229
7.1.2	edb_dynatune_profile	230
7.2	EDB の待機状態	230
7.2.1	EDB 待機状態バックグラウンドワーカー (EWSBW)	230
7.2.2	待機状態のワーカーを終了する	231
7.2.3	ウェイトステートインターフェイス	232
7.2.4	edb_wait_states_data	232
7.2.5	パラメーター	233
7.2.6	例	234
7.2.7	edb_wait_states_queries	235
7.2.8	パラメーター	235
7.2.9	例	235
7.2.10	edb_wait_states_sessions	236

7.2.11	パラメーター	237
7.2.12	例	237
7.2.13	edb_wait_states_samples	238
7.2.14	パラメーター	238
7.2.15	例	238
7.2.16	edb_wait_states_purge	239
7.2.17	例	240
第 8 章	EDB clone schema	241
8.1	セットアッププロセス	243
8.1.1	拡張機能と PL / Perl のインストール	243
8.1.2	構成パラメーターの設定	244
8.1.3	パフォーマンス構成パラメーター	245
8.1.4	ステータスログ	245
8.1.5	EDB クローンスキーマのインストール	246
8.1.6	外部サーバーとユーザーマッピングの作成	246
8.1.7	ローカルクローニング実行の外部サーバーとユーザーマッピング	247
8.1.8	リモートクローニング機能の外部サーバーとユーザーマッピング	248
8.2	EDB クローンスキーマ機能	251
8.2.1	ローカルコピースキーマ	251
8.2.2	パラメーター	252
8.2.3	例	253
8.2.4	localcopyschema_nb	256
8.2.5	パラメーター	256
8.2.6	例	257
8.2.7	リモートコピースキーマ	258
8.2.8	パラメーター	259
8.2.9	例	260
8.2.10	remotecopyschema_nb	263
8.2.11	パラメーター	264
8.2.12	例	265
8.2.13	process_status_from_log	266
8.2.14	パラメーター	266
8.2.15	例	267
8.2.16	remove_log_file_and_job	267
8.2.17	パラメーター	267
8.2.18	例	268
第 9 章	Enhanced SQL and other miscellaneous features	269
9.1	コメント	269
9.1.1	パラメーター	270
9.1.2	ノート	272
9.1.3	例	272
9.2	関数 version() の出力	273
9.3	スタンバイ時のロジカルデコード	274
第 10 章	System catalog tables	275
10.1	edb_dir	275

10.2	edb_all_resource_groups	275
10.3	edb_policy	276
10.4	edb_profile	277
10.5	edb_redaction_column	278
10.6	edb_redaction_policy	279
10.7	edb_resource_group	279
10.8	edb_variable	280
10.9	pg_synonym	280
10.10	product_component_version	281
第 11 章 EDB Postgres Advanced Server exceptions		283
11.1	EDB Postgres Advanced Server のキーワード	285

EDB Postgres Advanced Server は、オープンソースの PostgreSQL データベースに拡張機能を追加します。拡張機能は、データベース管理、拡張 SQL 機能、データベースとアプリケーションのセキュリティ、パフォーマンスの監視と分析、およびアプリケーション開発ユーティリティをサポートします。このガイドでは、EDB Postgres Advanced Server 専用の機能について説明します。

拡張された互換性機能。このセクションでは、EDB Postgres Advanced Server でサポートされている互換性機能の概要を提供します。

データベース管理。このセクションには、データベース管理者に役立つ機能とツールに関する情報が含まれています。

- **Index Advisor** は、アプリケーションのパフォーマンスを向上させるためにテーブルに必要な追加のインデックスを判断するのに役立ちます。
- **SQL Profiler** は、アプリケーションで実行されていない SQL クエリを見つけて診断します。
- **pgsnmpd** は、EDB Postgres Advanced Server の現在の状態に関する階層的な監視情報を返す SNMP エージェントです。

EDB リソースマネージャー。このセクションには、EDB Postgres Advanced Server プロセスによるシステムリソースの使用を制御する機能を提供する EDB Resource Manager 機能に関する情報が含まれています。

- **Resource Groups** は、リソース制限を定義できるグループを作成および維持する方法を示しています。
- **CPU Usage Throttling** は、EDB Postgres Advanced Server プロセスによる CPU 使用率を制御する方法を提供します。
- **Dirty Buffer Throttling** は、EDB Postgres Advanced Server プロセスによる共有バッファのダーティレートを制御するメソッドを提供します。

libpq C ライブラリ。libpq C ライブラリは、EDB Postgres Advanced Server の C アプリケーションプログラミングインターフェイス (API) 言語です。

PL デバッガー。PL デバッガーは、PL/pgSQL 用のグラフィカル指向のデバッグツールです。

パフォーマンス分析とチューニング。このセクションには、アプリケーションとデータベースサーバーのパフォーマンスを分析および向上させるためのさまざまなツールが含まれています。

- **Dynatune** は、アプリケーションの使用の種類に応じて EDB Postgres Advanced Server を構成するための迅速かつ簡単な手段を提供します。
- **EDB 待機状態**は、パフォーマンス診断のために待機イベントやその他のデータをキャプチャする方法を提供します。

EDB クローンスキーマ。このセクションには、単一のデータベース内またはあるデータベースから別のデータベースにスキーマとそのデータベースオブジェクトをコピーする機能を提供する EDB クローンスキーマ機能に関する情報が含まれています。

拡張された SQL およびその他の機能。このセクションには、拡張された SQL 機能と、柔軟性と利便性を高めるその他の機能に関する情報が含まれています。

システムカタログテーブル。このセクションには、EDB Postgres Advanced Server 固有のデータベースオブジェクト用に追加された追加の_システムカタログテーブル_が含まれています。

EDB Postgres Advanced Server の例外。このセクションには、事前定義された例外、SQLstate 値、関連する redwood エラーコード、および例外の説明に関する情報が含まれています。

EDB Postgres Advanced Server のキーワード。このセクションには、EDB Postgres Advanced Server がキーワードとして認識する単語に関する情報が含まれています。

EDB Postgres Advanced Server と PostgreSQL で共有される機能については、

[PostgreSQL core documentation](#) を参照してください。

第1章 Introduction

EDB Postgres Advanced Server 14 で追加された機能については、
release notes を参照してください。

1.1 ハードリミット

次の表では、PostgreSQL のさまざまなハード制限について説明します。ただし、絶対的なハード制限に達する前に、パフォーマンスの制限や使用可能なディスク領域などの実制的な制限が適用される場合があります。

Item	Upper limit	Comment
<i>database size</i>	unlimited	該当なし
<i>number of databases</i>	4,294,950,911	該当なし
<i>relations per database</i>	1,431,650,303	該当なし
<i>relation size</i>	32 TB	8192 バイトのデフォルト BLCKSZ
<i>rows per table</i>	limited by the number of tuples that can fit onto 4,294,967,295 pages	該当なし
<i>columns per table</i>	1600	1 ページに収まるタプルサ イズによってさらに制限さ れます。以下の注を参照
<i>field size</i>	1 GB	該当なし
<i>identifier length</i>	63 bytes	PostgreSQL を再コンパイル することで増やすことがで きます
<i>indexes per table</i>	unlimited	データベースごとの最大リ レーションによる制約
<i>columns per index</i>	32	PostgreSQL を再コンパイル することで増やすことがで きます
<i>partition keys</i>	32	PostgreSQL を再コンパイル することで増やすことがで きます

注釈:

- 格納されるタプルは単一の 8192 バイトのヒープページに収まる必要があるため、テーブルの列の最大数がさらに削減されます。例えば、タプルヘッダーを除くと、1600 個の `int` カラムで構成されるタプルは 6400 バイトを消費し、ヒープページに格納できます。ただし、1600 *bigint* 列のタプルは 12800 バイトを消費するため、ヒープページに収まりません。 *text*、*varchar*、*char* などの型の可変長フィールドは、値が必要なだけ大きい場合、テーブルの `TOAST` テーブルに値を格納できます。テーブルのヒープのタプル内に 18 バイトのポインターのみを残す必要があります。より短い長さの可変長フィールドの場合、4 バイトまたは 1 バイトのフィールドヘッダーが使用され、値はヒープタプル内に格納されます。
-
- テーブルから削除された列も、列の最大制限に影響します。さらに、新しく作成されたタプルの削除された列値は、タプルの null ビットマップで内部的に null としてマークされますが、null ビットマップも領域を占有します。

1.1.1 Conventions used

この情報は、Linux システムと Windows システムの両方に適用されます。ディレクトリパスは、スラッシュを使用した Linux 形式で表示されます。Windows システムで作業する場合、ドライブ文字とそれに続くコロンでディレクトリパスを開始し、スラッシュをバックスラッシュに置き換えます。

この情報の一部は、PostgreSQL および EDB Postgres Advanced Server データベースシステムに適用される場合があります。* EDB Postgres Advanced Server という用語は、*EDB Postgres Advanced Server* を指すために使用されます。Postgres* という用語は、PostgreSQL と EDB Postgres Advanced Server の両方を一般的に指すために使用されます。これら 2 つのデータベースシステムを区別する必要がある場合は、特定の名前 (PostgreSQL または EDB Postgres Advanced Server) が使用されます。

PostgreSQL または EDB Postgres Advanced Server 製品のインストールディレクトリパスは、*POSTGRES_INSTALL_HOME* と呼ばれます。

- PostgreSQL Linux インストールの場合、デフォルトはバージョン 10 以前の `/opt/PostgreSQL/x.x` です。以降のバージョンでは、PostgreSQL コミュニティパッケージを使用してください。
- バージョン 10 以前の対話型インストーラーを使用して実行された EDB Postgres Advanced Server Linux インストールの場合、デフォルトは `/opt/edb/asx.x` です。
- RPM パッケージを使用して実行される EDB Postgres Advanced Server Linux インストールの場合、デフォルトは `/usr/edb/asxx` です。
- EDB Postgres Advanced Server Windows インストールの場合、デフォルトは `C:\Program Files\edb\asxx` です。製品バージョン番号は、バージョン 10 以降の場合は `x.x` または `xx` で表されます。

1.1.2 About the examples

この例では、EDB Postgres Advanced Server のインストール時に作成およびロードされるサンプルテーブル `dept`、`emp`、および `jobhist` を使用します。

次のスクリプトを実行することにより、サンプルデータベースのテーブルとプログラムをいつでも再作成できます。

```
/usr/edb/asxx/share/pg-sample.sql
```

`xx` は EDB Postgres Advanced Server のバージョン番号です。

さらに、同じディレクトリのスクリプトには、Oracle データベースと互換性のある構文を使用して作成されたデータベースオブジェクトが含まれています。このスクリプトファイルは `edb-sample.sql` です。スクリプト：

- 現在接続されているデータベースにサンプルテーブルとプログラムを作成します。
- `PUBLIC` グループにテーブルに対するすべての権限を付与します。

テーブルとプログラムは、現在のユーザーがテーブルとプロシージャを作成する権限を持つ検索パスの最初のスキーマに作成されます。次のコマンドを使用して検索パスを表示できます。

```
SHOW SEARCH_PATH;
```

PSQL コマンドを使用して、検索パスを変更できます。

サンプルデータベースの説明

サンプルデータベースは、組織の従業員を表します。これには、従業員、部門、および従業員の履歴レコードの 3 種類のレコードが含まれています。

各従業員には、識別番号、名前、入社日、給与、マネージャーがあります。一部の従業員は、給与に加えてコミッションを受け取ります。従業員関連の情報はすべて `emp` テーブルに保存されます。

サンプル会社は地域が多様であるため、部門の場所を追跡しています。会社員はそれぞれ部署に配属されます。各部門は、一意の部門番号と短い名前でも識別されます。各部門は 1 つの場所に関連付けられます。部門関連の情報はすべて `dept` テーブルに保存されます。

同社はまた、従業員が保有するジョブに関する情報を追跡します。長く勤めている社員の中には、さまざまな役職、昇給、部署異動などを行っている人もいます。従業員のステータスに変更が発生すると、会社は前の職位の終了日を記録します。開始日と、新しい役職、部門、給与、およびステータス変更の理由を含む新しいジョブレコードが追加されます。すべての従業員の履歴は `jobhist` テーブルに保持されます。

以下は `pg-sample.sql` スクリプトです。

```
SET datestyle TO iso, dmy;

--
-- Script that creates the sample tables, views
```

(次のページに続く)

(前のページからの続き)

```

- - functions, triggers, etc.
- -
- - Start new transaction - commit all or nothing
- -
BEGIN;
- -
- - Create and load tables used in the documentation examples.
- -
- - Create the dept table
- -
CREATE TABLE dept (
    deptno          NUMERIC(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
    dname           VARCHAR(14) CONSTRAINT dept_dname_uq UNIQUE,
    loc             VARCHAR(13)
);
- -
- - Create the emp table
- -
CREATE TABLE emp (
    empno           NUMERIC(4) NOT NULL CONSTRAINT emp_pk PRIMARY KEY,
    ename           VARCHAR(10),
    job             VARCHAR(9),
    mgr             NUMERIC(4),
    hiredate        DATE,
    sal             NUMERIC(7,2) CONSTRAINT emp_sal_ck CHECK (sal > 0),
    comm            NUMERIC(7,2),
    deptno          NUMERIC(2) CONSTRAINT emp_ref_dept_fk
                    REFERENCES dept(deptno)
);
- -
- - Create the jobhist table
- -
CREATE TABLE jobhist (
    empno           NUMERIC(4) NOT NULL,
    startdate       TIMESTAMP(0) NOT NULL,
    enddate         TIMESTAMP(0),
    job             VARCHAR(9),
    sal             NUMERIC(7,2),
    comm            NUMERIC(7,2),
    deptno          NUMERIC(2),
    chgdesc         VARCHAR(80),
    CONSTRAINT jobhist_pk PRIMARY KEY (empno, startdate),
    CONSTRAINT jobhist_ref_emp_fk FOREIGN KEY (empno)
        REFERENCES emp(empno) ON DELETE CASCADE,
    CONSTRAINT jobhist_ref_dept_fk FOREIGN KEY (deptno)

```

(次のページに続く)

(前のページからの続き)

```

REFERENCES dept (deptno) ON DELETE SET NULL,
CONSTRAINT jobhist_date_chk CHECK (startdate <= enddate)
);
- -
- - Create the salesemp view
- -
CREATE OR REPLACE VIEW salesemp AS
SELECT empno, ename, hiredate, sal, comm FROM emp WHERE job = SALESMAN;
- -
- - Sequence to generate values for function new_empno.
- -
CREATE SEQUENCE next_empno START WITH 8000 INCREMENT BY 1;
- -
- - Issue PUBLIC grants
- -
-GRANT ALL ON emp TO PUBLIC;
-GRANT ALL ON dept TO PUBLIC;
-GRANT ALL ON jobhist TO PUBLIC;
-GRANT ALL ON salesemp TO PUBLIC;
-GRANT ALL ON next_empno TO PUBLIC;
- -
- - Load the dept table
- -
INSERT INTO dept VALUES (10,ACCOUNTING,NEW YORK);
INSERT INTO dept VALUES (20,RESEARCH,DALLAS);
INSERT INTO dept VALUES (30,SALES,CHICAGO);
INSERT INTO dept VALUES (40,OPERATIONS,BOSTON);
- -
- - Load the emp table
- -
INSERT INTO emp VALUES (7369,SMITH,CLERK,7902,17-DEC-80,800,NULL,20);
INSERT INTO emp VALUES (7499,ALLEN,SALESMAN,7698,20-FEB-81,1600,300,30);
INSERT INTO emp VALUES (7521,WARD,SALESMAN,7698,22-FEB-81,1250,500,30);
INSERT INTO emp VALUES (7566,JONES,MANAGER,7839,02-APR-81,2975,NULL,20);
INSERT INTO emp VALUES (7654,MARTIN,SALESMAN,7698,28-SEP-81,1250,1400,30);
INSERT INTO emp VALUES (7698,BLAKE,MANAGER,7839,01-MAY-81,2850,NULL,30);
INSERT INTO emp VALUES (7782,CLARK,MANAGER,7839,09-JUN-81,2450,NULL,10);
INSERT INTO emp VALUES (7788,SCOTT,ANALYST,7566,19-APR-87,3000,NULL,20);
INSERT INTO emp VALUES (7839,KING,PRESIDENT,NULL,17-NOV-81,5000,NULL,10);
INSERT INTO emp VALUES (7844,TURNER,SALESMAN,7698,08-SEP-81,1500,0,30);
INSERT INTO emp VALUES (7876,ADAMS,CLERK,7788,23-MAY-87,1100,NULL,20);
INSERT INTO emp VALUES (7900,JAMES,CLERK,7698,03-DEC-81,950,NULL,30);
INSERT INTO emp VALUES (7902,FORD,ANALYST,7566,03-DEC-81,3000,NULL,20);
INSERT INTO emp VALUES (7934,MILLER,CLERK,7782,23-JAN-82,1300,NULL,10);
- -

```

(次のページに続く)

(前のページからの続き)

```

- - Load the jobhist table
- -
INSERT INTO jobhist VALUES (7369,17-DEC-80,NULL,CLERK,800,NULL,20,New Hire);
INSERT INTO jobhist VALUES (7499,20-FEB-81,NULL,SALESMAN,1600,300,30,New Hire);
INSERT INTO jobhist VALUES (7521,22-FEB-81,NULL,SALESMAN,1250,500,30,New Hire);
INSERT INTO jobhist VALUES (7566,02-APR-81,NULL,MANAGER,2975,NULL,20,New Hire);
INSERT INTO jobhist VALUES (7654,28-SEP-81,NULL,SALESMAN,1250,1400,30,New Hire);
INSERT INTO jobhist VALUES (7698,01-MAY-81,NULL,MANAGER,2850,NULL,30,New Hire);
INSERT INTO jobhist VALUES (7782,09-JUN-81,NULL,MANAGER,2450,NULL,10,New Hire);
INSERT INTO jobhist VALUES (7788,19-APR-87,12-APR-88,CLERK,1000,NULL,20,New Hire);
INSERT INTO jobhist VALUES (7788,13-APR-88,04-MAY-89,CLERK,1040,NULL,20,Raise);
INSERT INTO jobhist VALUES (7788,05-MAY-90,NULL,ANALYST,3000,NULL,20,Promoted to
↳Analyst);
INSERT INTO jobhist VALUES (7839,17-NOV-81,NULL,PRESIDENT,5000,NULL,10,New Hire);
INSERT INTO jobhist VALUES (7844,08-SEP-81,NULL,SALESMAN,1500,0,30,New Hire);
INSERT INTO jobhist VALUES (7876,23-MAY-87,NULL,CLERK,1100,NULL,20,New Hire);
INSERT INTO jobhist VALUES (7900,03-DEC-81,14-JAN-83,CLERK,950,NULL,10,New Hire);
INSERT INTO jobhist VALUES (7900,15-JAN-83,NULL,CLERK,950,NULL,30,Changed to Dept 30);
INSERT INTO jobhist VALUES (7902,03-DEC-81,NULL,ANALYST,3000,NULL,20,New Hire);
INSERT INTO jobhist VALUES (7934,23-JAN-82,NULL,CLERK,1300,NULL,10,New Hire);
- -
- - Populate statistics table and view (pg_statistic/pg_stats)
- -
ANALYZE dept;
ANALYZE emp;
ANALYZE jobhist;
- -
- - Function that lists all employees numbers and names
- - from the emp table using a cursor.
- -
CREATE OR REPLACE FUNCTION list_emp() RETURNS VOID
AS $$
DECLARE
    v_empno      NUMERIC(4);
    v_ename      VARCHAR(10);
    emp_cur CURSOR FOR
        SELECT empno, ename FROM emp ORDER BY empno;
BEGIN
    OPEN emp_cur;
    RAISE INFO EMPNO ENAME;
    RAISE INFO -----;
    LOOP
        FETCH emp_cur INTO v_empno, v_ename;
        EXIT WHEN NOT FOUND;
        RAISE INFO % %, v_empno, v_ename;

```

(次のページに続く)

(前のページからの続き)

```

        END LOOP;
        CLOSE emp_cur;
        RETURN;
END;
$$ LANGUAGE plpgsql;
--
--  Function that selects an employee row given the employee
--  number and displays certain columns.
--
CREATE OR REPLACE FUNCTION select_emp (
    p_empno          NUMERIC
) RETURNS VOID
AS $$
DECLARE
    v_ename          emp.ename%TYPE;
    v_hiredate       emp.hiredate%TYPE;
    v_sal            emp.sal%TYPE;
    v_comm           emp.comm%TYPE;
    v_dname          dept.dname%TYPE;
    v_disp_date      VARCHAR(10);
BEGIN
    SELECT INTO
        v_ename, v_hiredate, v_sal, v_comm, v_dname
        ename, hiredate, sal, COALESCE(comm, 0), dname
    FROM emp e, dept d
    WHERE empno = p_empno
        AND e.deptno = d.deptno;
    IF NOT FOUND THEN
        RAISE INFO Employee % not found, p_empno;
        RETURN;
    END IF;
    v_disp_date := TO_CHAR(v_hiredate, MM/DD/YYYY);
    RAISE INFO Number      : %, p_empno;
    RAISE INFO Name        : %, v_ename;
    RAISE INFO Hire Date   : %, v_disp_date;
    RAISE INFO Salary      : %, v_sal;
    RAISE INFO Commission: %, v_comm;
    RAISE INFO Department: %, v_dname;
    RETURN;
EXCEPTION
    WHEN OTHERS THEN
        RAISE INFO The following is SQLERRM : %, SQLERRM;
        RAISE INFO The following is SQLSTATE: %, SQLSTATE;
        RETURN;
END;

```

(次のページに続く)

(前のページからの続き)

```

$$ LANGUAGE plpgsql;
- -
- -  A RECORD type used to format the return value of
- -  function, emp_query.
- -
CREATE TYPE emp_query_type AS (
    empno      NUMERIC,
    ename       VARCHAR(10),
    job         VARCHAR(9),
    hiredate    DATE,
    sal         NUMERIC
);
- -
- -  Function that queries the emp table based on
- -  department number and employee number or name. Returns
- -  employee number and name as INOUT parameters and job,
- -  hire date, and salary as OUT parameters. These are
- -  returned in the form of a record defined by
- -  RECORD type, emp_query_type.
- -
CREATE OR REPLACE FUNCTION emp_query (
    IN p_deptno      NUMERIC,
    INOUT p_empno     NUMERIC,
    INOUT p_ename     VARCHAR,
    OUT p_job         VARCHAR,
    OUT p_hiredate    DATE,
    OUT p_sal         NUMERIC
)
AS $$
BEGIN
    SELECT INTO
        p_empno, p_ename, p_job, p_hiredate, p_sal
        empno, ename, job, hiredate, sal
    FROM emp
    WHERE deptno = p_deptno
        AND (empno = p_empno
            OR ename = UPPER(p_ename));
END;
$$ LANGUAGE plpgsql;
- -
- -  Function to call emp_query_caller with IN and INOUT
- -  parameters. Displays the results received from INOUT and
- -  OUT parameters.
- -
CREATE OR REPLACE FUNCTION emp_query_caller() RETURNS VOID

```

(次のページに続く)

(前のページからの続き)

```

AS $$
DECLARE
    v_deptno      NUMERIC;
    v_empno       NUMERIC;
    v_ename       VARCHAR;
    v_rows        INTEGER;
    r_emp_query    EMP_QUERY_TYPE;
BEGIN
    v_deptno := 30;
    v_empno := 0;
    v_ename := Martin;
    r_emp_query := emp_query(v_deptno, v_empno, v_ename);
    RAISE INFO Department : %, v_deptno;
    RAISE INFO Employee No: %, (r_emp_query).empno;
    RAISE INFO Name       : %, (r_emp_query).ename;
    RAISE INFO Job        : %, (r_emp_query).job;
    RAISE INFO Hire Date  : %, (r_emp_query).hiredate;
    RAISE INFO Salary     : %, (r_emp_query).sal;
    RETURN;
EXCEPTION
    WHEN OTHERS THEN
        RAISE INFO The following is SQLERRM : %, SQLERRM;
        RAISE INFO The following is SQLSTATE: %, SQLSTATE;
        RETURN;
END;
$$ LANGUAGE plpgsql;

--
--  Function to compute yearly compensation based on semimonthly
--  salary.
--
CREATE OR REPLACE FUNCTION emp_comp (
    p_sal      NUMERIC,
    p_comm     NUMERIC
) RETURNS NUMERIC
AS $$
BEGIN
    RETURN (p_sal + COALESCE(p_comm, 0)) * 24;
END;
$$ LANGUAGE plpgsql;

--
--  Function that gets the next number from sequence, next_empno,
--  and ensures it is not already in use as an employee number.
--
CREATE OR REPLACE FUNCTION new_empno() RETURNS INTEGER
AS $$

```

(次のページに続く)

(前のページからの続き)

```

DECLARE
    v_cnt          INTEGER := 1;
    v_new_empno    INTEGER;
BEGIN
    WHILE v_cnt > 0 LOOP
        SELECT INTO v_new_empno nextval(next_empno);
        SELECT INTO v_cnt COUNT(*) FROM emp WHERE empno = v_new_empno;
    END LOOP;
    RETURN v_new_empno;
END;
$$ LANGUAGE plpgsql;
--
--  Function that adds a new clerk to table emp.
--
CREATE OR REPLACE FUNCTION hire_clerk (
    p_ename        VARCHAR,
    p_deptno       NUMERIC
) RETURNS NUMERIC
AS $$
DECLARE
    v_empno        NUMERIC(4);
    v_ename        VARCHAR(10);
    v_job          VARCHAR(9);
    v_mgr          NUMERIC(4);
    v_hiredate     DATE;
    v_sal          NUMERIC(7,2);
    v_comm         NUMERIC(7,2);
    v_deptno       NUMERIC(2);
BEGIN
    v_empno := new_empno();
    INSERT INTO emp VALUES (v_empno, p_ename, CLERK, 7782,
        CURRENT_DATE, 950.00, NULL, p_deptno);
    SELECT INTO
        v_empno, v_ename, v_job, v_mgr, v_hiredate, v_sal, v_comm, v_deptno
        empno, ename, job, mgr, hiredate, sal, comm, deptno
    FROM emp WHERE empno = v_empno;
    RAISE INFO Department : %, v_deptno;
    RAISE INFO Employee No: %, v_empno;
    RAISE INFO Name       : %, v_ename;
    RAISE INFO Job        : %, v_job;
    RAISE INFO Manager    : %, v_mgr;
    RAISE INFO Hire Date  : %, v_hiredate;
    RAISE INFO Salary     : %, v_sal;
    RAISE INFO Commission : %, v_comm;
    RETURN v_empno;

```

(次のページに続く)

(前のページからの続き)

```

EXCEPTION
    WHEN OTHERS THEN
        RAISE INFO The following is SQLERRM : %, SQLERRM;
        RAISE INFO The following is SQLSTATE: %, SQLSTATE;
        RETURN -1;
END;
$$ LANGUAGE plpgsql;
--
--  Function that adds a new salesman to table emp.
--
CREATE OR REPLACE FUNCTION hire_salesman (
    p_ename          VARCHAR,
    p_sal            NUMERIC,
    p_comm           NUMERIC
) RETURNS NUMERIC
AS $$
DECLARE
    v_empno          NUMERIC(4);
    v_ename          VARCHAR(10);
    v_job            VARCHAR(9);
    v_mgr            NUMERIC(4);
    v_hiredate       DATE;
    v_sal            NUMERIC(7,2);
    v_comm           NUMERIC(7,2);
    v_deptno         NUMERIC(2);
BEGIN
    v_empno := new_empno();
    INSERT INTO emp VALUES (v_empno, p_ename, SALESMAN, 7698,
        CURRENT_DATE, p_sal, p_comm, 30);
    SELECT INTO
        v_empno, v_ename, v_job, v_mgr, v_hiredate, v_sal, v_comm, v_deptno
        empno, ename, job, mgr, hiredate, sal, comm, deptno
        FROM emp WHERE empno = v_empno;
    RAISE INFO Department : %, v_deptno;
    RAISE INFO Employee No: %, v_empno;
    RAISE INFO Name       : %, v_ename;
    RAISE INFO Job        : %, v_job;
    RAISE INFO Manager    : %, v_mgr;
    RAISE INFO Hire Date  : %, v_hiredate;
    RAISE INFO Salary     : %, v_sal;
    RAISE INFO Commission : %, v_comm;
    RETURN v_empno;
EXCEPTION
    WHEN OTHERS THEN
        RAISE INFO The following is SQLERRM : %, SQLERRM;

```

(次のページに続く)

(前のページからの続き)

```

        RAISE INFO The following is SQLSTATE: %, SQLSTATE;
        RETURN -1;
END;
$$ LANGUAGE plpgsql;
--
-- Rule to INSERT into view salesemp
--
CREATE OR REPLACE RULE salesemp_i AS ON INSERT TO salesemp
DO INSTEAD
    INSERT INTO emp VALUES (NEW.empno, NEW.ename, SALESMAN, 7698,
        NEW.hiredate, NEW.sal, NEW.comm, 30);
--
-- Rule to UPDATE view salesemp
--
CREATE OR REPLACE RULE salesemp_u AS ON UPDATE TO salesemp
DO INSTEAD
    UPDATE emp SET empno      = NEW.empno,
                  ename       = NEW.ename,
                  hiredate    = NEW.hiredate,
                  sal         = NEW.sal,
                  comm        = NEW.comm
    WHERE empno = OLD.empno;
--
-- Rule to DELETE from view salesemp
--
CREATE OR REPLACE RULE salesemp_d AS ON DELETE TO salesemp
DO INSTEAD
    DELETE FROM emp WHERE empno = OLD.empno;
--
-- After statement-level trigger that displays a message after
-- an insert, update, or deletion to the emp table. One message
-- per SQL command is displayed.
--
CREATE OR REPLACE FUNCTION user_audit_trig() RETURNS TRIGGER
AS $$
DECLARE
    v_action      VARCHAR(24);
    v_text        TEXT;
BEGIN
    IF TG_OP = INSERT THEN
        v_action := added employee(s) on ;
    ELSIF TG_OP = UPDATE THEN
        v_action := updated employee(s) on ;
    ELSIF TG_OP = DELETE THEN
        v_action := deleted employee(s) on ;

```

(次のページに続く)

(前のページからの続き)

```

    END IF;
    v_text := User || USER || v_action || CURRENT_DATE;
    RAISE INFO %, v_text;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER user_audit_trig
    AFTER INSERT OR UPDATE OR DELETE ON emp
    FOR EACH STATEMENT EXECUTE PROCEDURE user_audit_trig();
--
-- Before row-level trigger that displays employee number and
-- salary of an employee that is about to be added, updated,
-- or deleted in the emp table.
--
CREATE OR REPLACE FUNCTION emp_sal_trig() RETURNS TRIGGER
AS $$
DECLARE
    sal_diff          NUMERIC(7,2);
BEGIN
    IF TG_OP = INSERT THEN
        RAISE INFO Inserting employee %, NEW.empno;
        RAISE INFO ..New salary: %, NEW.sal;
        RETURN NEW;
    END IF;
    IF TG_OP = UPDATE THEN
        sal_diff := NEW.sal - OLD.sal;
        RAISE INFO Updating employee %, OLD.empno;
        RAISE INFO ..Old salary: %, OLD.sal;
        RAISE INFO ..New salary: %, NEW.sal;
        RAISE INFO ..Raise : %, sal_diff;
        RETURN NEW;
    END IF;
    IF TG_OP = DELETE THEN
        RAISE INFO Deleting employee %, OLD.empno;
        RAISE INFO ..Old salary: %, OLD.sal;
        RETURN OLD;
    END IF;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER emp_sal_trig
    BEFORE DELETE OR INSERT OR UPDATE ON emp
    FOR EACH ROW EXECUTE PROCEDURE emp_sal_trig();
COMMIT;

```


第2章 Enhanced compatibility features

EDB Postgres Advanced Server には、Oracle アプリケーションでサポートされる構文の互換性を提供する拡張機能が含まれています。EDB Postgres Advanced Server でサポートされている互換性機能の詳細については、Database Compatibility for Oracle Developers を参照してください。

次のセクションでは、EDB Postgres Advanced Server でサポートされている互換性機能のいくつかを強調します。

2.1 互換性機能を有効にする

互換性機能を利用できるように EDB Postgres Advanced Server をインストールする方法はいくつかあります。

- クラスターを初期化する前に、INITDBOPTS 変数（EDB Postgres Advanced Server サービス構成ファイル内）を使用して `--redwood-like` を指定します。
- `initdb` を呼び出して Oracle モードと互換性のあるクラスターを初期化する場合、`--redwood-like` オプションまたは `--no-redwood-compat` オプションを含めて、クラスターを Oracle 非互換モードで初期化します。

EDB Postgres Advanced Server インストーラーでサポートされているインストールオプションの詳細については、

Configuration parameters compatible with Oracle databases および Managing an EDB Postgres Advanced Server installation を参照してください。

2.2 ストアドプロシージャ言語

EDB Postgres Advanced Server は、カスタムプロシージャ、ファンクション、トリガー、パッケージを作成できる生産性の高い手続き型言語をサポートしています。手続き言語：

- SQL 言語と組み込みパッケージを補完します。
- シームレスな開発およびテスト環境を提供します。
- 再利用可能なコードを作成できます。

詳細については、Database Compatibility: Stored Procedural Language を参照してください。

2.3 オプティマイザーのヒント

DELETE、INSERT、SELECT、または UPDATE コマンドを呼び出すと、サーバーは一連の実行計画を生成します。サーバーは、これらの実行プランを分析した後、一般的に最短時間で結果セットを返すプランを選択します。サーバーのプランの選択は、いくつかの要因によって異なります。

- データ処理操作の推定実行コスト。
- `postgresql.conf` ファイルの Query Tuning セクションのパラメーターに割り当てられたパラメーター値。
- ANALYZE コマンドで収集された列の統計。

原則として、クエリプランナーは最も安価なプランを選択します。オプティマイザーヒントを使用して、サーバーがクエリプランを選択するときに影響を与えることができます。

オプティマイザーヒントは、DELETE、INSERT、SELECT または UPDATE コマンドの直後にあるコメントのような構文に埋め込まれた指示子（または複数の指示子）です。コメント内のキーワードは、結果セットを生成するときに特定のプランを使用するか回避するようにサーバーに指示します。詳細については、[オプティマイザーのヒント](#)を参照してください。

2.4 データディクショナリビュー

EDB Postgres Advanced Server には、Oracle データディクショナリビューと互換性のある方法でデータベースオブジェクトに関する情報を提供するビューのセットが含まれています。EDB Postgres Advanced Server で使用可能なビューの詳細については、[Database compatibility for Oracle developers: catalog views](#) を参照してください。

2.5 dblink_ora

`dblink_ora` は、Oracle システムに保存されている SELECT、INSERT、UPDATE または DELETE データを EDB Postgres Advanced Server 内からできるようにする OCI ベースのデータベースリンクを提供します。`dblink_ora` の使用、およびサポートされている関数とプロシージャの詳細については、

[`dblink\_ora`](#) を参照してください。

2.6 プロファイル管理

EDB Postgres Advanced Server は、プロファイル管理用の互換性のある SQL 構文をサポートしています。Profile management コマンドを使用すると、データベースのスーパーユーザーは名前付けの**プロファイル**を作成および管理できます。各プロファイルは、パスワードと MD5 認証を強化するパスワード管理のルールを定義します。プロファイルのルールでは次のことができます。

- 失敗したログイン試行を数える
- ログインに何度も失敗したためアカウントをロックする

- パスワードに有効期限をマークする
- パスワードの有効期限後の猶予期間を定義する
- パスワードの複雑さのルールを定義する
- パスワードの再利用を制限するルールを定義する

プロファイルは、同等の認証要件を共有するロールのグループを簡単に管理できるようにする属性の名前付けセットです。パスワード要件が変更された場合、プロファイルを変更して、そのプロファイルに関連付けられている各ユーザーに新しい要件を適用できます。

プロファイルを作成したら、プロファイルを 1 人以上のユーザーに関連付けることができます。ユーザーがサーバーに接続すると、サーバーはログインロールに関連付けられたプロファイルを適用します。プロファイルはクラスター内のすべてのデータベースで共有されますが、各クラスターには複数のプロファイルがある場合があります。複数のデータベースにアクセスできる単一のユーザーは、クラスター内の各データベースに接続するときに同じプロファイルを使用します。

プロファイル管理コマンドの使用については、
[プロファイル管理](#) を参照してください。

2.7 ビルトインパッケージ

EDB Postgres Advanced Server は、Oracle のプロシージャおよびファンクションとの互換性を提供する多くのビルトインパッケージをサポートしています。

Package name	Description
DBMS_ALERT	DBMS_ALERT パッケージは、アラートを登録、送信および受信する機能を提供します。
DBMS_AQ	DBMS_AQ パッケージは、EDB Postgres Advanced Server のメッセージキューイングと処理を提供します。
DBMS_AQADM	DBMS_AQADM パッケージは、アドバンスド・キューイング機能をサポートするプロシージャを提供します。
DBMS_CRYPTO	DBMS_CRYPTO パッケージは、RAW、BLOB または CLOB データを暗号化または復号化できるファンクションとプロシージャを提供します。DBMS_CRYPTO ファンクションを使用して、暗号的に強力なランダム値を生成することもできます。
DBMS_JOB	DBMS_JOB パッケージは、ジョブの作成、スケジュールおよび管理を提供します。
DBMS_LOB	DBMS_LOB パッケージは、ラージオブジェクトを操作する機能を提供します。
DBMS_LOCK	EDB Postgres Advanced Server は、DBMS_LOCK.SLEEP プロシージャのサポートを提供します。
DBMS_MVIEW	DBMS_MVIEW パッケージのプロシージャを使用して、マテリアライズドビューとその依存性を管理およびリフレッシュします。
DBMS_OUTPUT	DBMS_OUTPUT パッケージは、メッセージをメッセージバッファに送信したり、メッセージを取得したりできます。

次のページに続く

表 1 – 前のページからの続き

Package name	Description
DBMS_PIPE	DBMS_PIPE パッケージは、同じデータベースクラスターに接続されたセッション内またはセッション間でパイプを介してメッセージを送信する機能を提供します。
DBMS_PROFILER	DBMS_PROFILER パッケージは、パフォーマンスプロファイリングセッション中に実行される PL/pgSQL および SPL ステートメントに関するパフォーマンス情報を収集して保存します。
DBMS_RANDOM	DBMS_RANDOM パッケージは、ランダム値を生成する多くのメソッドを提供します。
DBMS_REDACT	DBMS_REDACT パッケージを使用すると、問合せによって返されるデータのリダクションまたはマスキングが有効になります。
DBMS_RLS	DBMS_RLS パッケージを使用すると、特定の EDB Postgres Advanced Server データベースオブジェクトに仮想プライベートデータベースを実装できます。
DBMS_SCHEDULER	DBMS_SCHEDULER パッケージは、ジョブ、プログラムおよびジョブ・スケジュールを作成および管理する方法を提供します。
DBMS_SESSION	EDB Postgres Advanced Server は、DBMS_SESSION.SET_ROLE プロシージャのサポートを提供します。
DBMS_SQL	DBMS_SQL パッケージは、EDB 動的 SQL 機能へのアプリケーションインタフェースを提供します。
DBMS_UTILITY	DBMS_UTILITY パッケージは、さまざまなユーティリティプログラムを提供します。
UTL_ENCODE	UTL_ENCODE パッケージは、データをエンコードおよびデコードする方法を提供します。
UTL_FILE	UTL_FILE パッケージは、オペレーティングシステムのファイルシステム上のファイルを読み書きする機能を提供します。
UTL_HTTP	UTL_HTTP パッケージは、HTTP または HTTPS プロトコルを使用して URL で見つかった情報を取得する方法を提供します。
UTL_MAIL	UTL_MAIL パッケージは、電子メールを管理する機能を提供します。
UTL_RAW	UTL_RAW パッケージを使用すると、生データ型の長さを操作または取得できます。
UTL_SMTP	UTL_SMTP パッケージは、Simple Mail Transfer Protocol (SMTP) を介して電子メールを送信する機能を提供します。
UTL_URL	UTL_URL パッケージは、URL 内の不正な予約文字をエスケープする方法を提供します。

各パッケージ内で使用可能なプロシージャとファンクションの詳細については、
[ビルトインパッケージ](#) を参照してください。

2.8 Open Client ライブラリ

Open Client Library は、Oracle Call Interface とのアプリケーションの相互運用性を提供し EDB Postgres Advanced Server. Open Client Library の EDB 実装は C で書かれています。

次の図は、Open Client Library と Oracle Call Interface のアプリケーションスタックを比較しています。

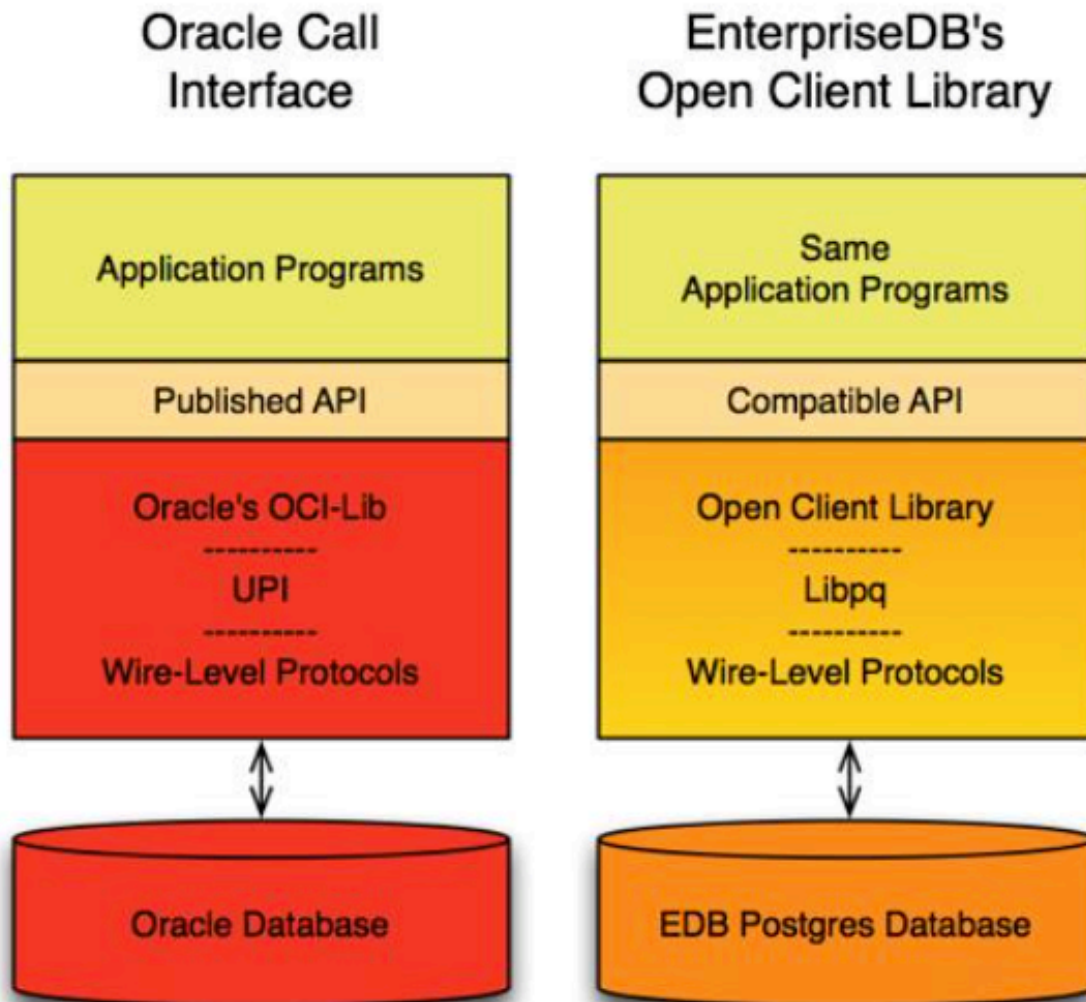


図 1: The Open Client Library

Open Client ライブラリでサポートされている関数の詳細については、EDB OCL Connector を参照してください。

2.9 ユーティリティ

以下にリストされているユーティリティでサポートされている互換性のある構文の詳細については、
Database Compatibility for Oracle Developers: Tools and Utilities を参照してください。

2.9.1 EDB*Plus

EDB*Plus は、Oracle の開発者とユーザーになじみのある EDB Postgres Advanced Server にコマンドライン ユーザーインターフェイスを提供するユーティリティプログラムです。EDB*Plus は、SQL コマンド、SPL 無名ブロックおよび EDB*Plus コマンドを受け入れます。

EDB*Plus を使用すると、次のことができます。

- 特定のデータベースオブジェクトを照会する
- ストアドプロシージャを実行する
- SQL コマンドからのフォーマット出力
- バッチスクリプトを実行する
- OS コマンドの実行
- レコード出力

詳細については、EDBPlus を参照してください。

2.9.2 EDB*Loader

EDB*Loader は、EDB Postgres Advanced Server の Oracle データベースと互換性のあるインターフェイスを提供する高性能なバルクデータローダーです。EDB*Loader コマンドラインユーティリティは、Oracle SQL*Loader が提供するパラメーターのサブセットを使用して、入力ソース（通常はファイル）から 1 つ以上のテーブルにデータをロードします。

EDB*Loader の機能は次のとおりです。

- Oracle SQL*Loader データロード方法のサポート - 従来のパスロード、ダイレクトパスロード、およびパラレルダイレクトパスロード
- 制御ファイル指示子の Oracle SQL*Loader 互換の構文
- 区切り文字または固定幅フィールドの入力データ
- リジェクトされたレコードを収集するための不正なファイル
- 複数のターゲットテーブルのロード
- 対象テーブルの選択基準を満たさないレコードを収集するための廃棄ファイル
- 標準入力からのデータロードとリモートロード

2.9.3 EDB*ラップ

EDB*Wrap ユーティリティは、独自のソースコードとプログラム（ファンクション、ストアドプロシージャ、トリガー、およびパッケージ）を不正な監視から保護します。EDB*Wrap プログラムは、SPL または PL/pgSQL のソースコード（プレーンテキスト）を含むファイルを、ほぼ不可能な形式の同じコードを含むファイルに変換します。難読化された形式のコードを入手したら、そのコードを EDB Postgres Advanced Server に送信すると、それらのプログラムが難読化された形式で保存されます。EDB*Wrap はあいまいなコードを実行しますが、テーブル定義はまだ公開されています。

ラップしたものはすべて難読化された形式で保存されます。パッケージ全体をラップすると、パッケージ本体のソース、パッケージヘッダーに含まれるプロトタイプ、パッケージ本体に含まれるファンクションとプロシージャが難読化された形式で保存されます。

2.9.4 動的ランタイム計測ツールアーキテクチャ（DRITA）

Dynamic Runtime Instrumentation Tools Architecture（DRITA）を使用すると、DBA はカタログビューを照会して、個々のセッションまたはシステム全体のパフォーマンスに影響を与える**待機イベント**を判断できます。DRITA は、各イベントの発生回数と待機時間を記録します。この情報を使用して、パフォーマンスの問題を診断できます。DRITA は、最小限のシステムリソースを消費しながら、この機能を提供します。

DRITA は**スナップショット**を比較して、システムのパフォーマンスを評価します。スナップショットは、特定の時点での保存されたシステムパフォーマンスデータです。各スナップショットは一意の ID 番号で識別されます。DRITA レポート機能でスナップショット ID 番号を使用して、システムパフォーマンス統計を返すことができます。

2.10 ECPGPlus

EDB は ECPG（PostgreSQL プリコンパイラ）を拡張して ECPGPlus を作成しました。ECPGPlus を使用すると、C アプリケーションに埋め込み SQL コマンドを含めることができます。ECPGPlus を使用して、埋め込み SQL コマンドを含むアプリケーションをコンパイルすると、SQL コードが構文チェックされ、C に変換されます。

ECPGPlus は、EDB Postgres Advanced Server データベースに接続したときに、C プログラムの Pro*C 構文をサポートします。ECPGPlus は以下をサポートしています。

- Oracle 動的 SQL –方法 4（ODS-M4）
- Pro*C 互換の匿名ブロック
- Oracle データベースと互換性のある CALL ステートメント

詳細については、[ECPGPlus](#) を参照してください。

2.11 テーブルのパーティショニング

パーティションテーブルでは、1つの論理的に大きなテーブルがより小さな物理的に分割されます。パーティション化にはいくつかの利点があります。

- 特定の状況、特にテーブルの頻繁にアクセスされる行のほとんどが単一のパーティションまたは少数のパーティションにある場合、クエリのパフォーマンスを劇的に向上させることができます。パーティション化を使用すると、インデックスの先頭からパーティション列を省略することができるため、インデックスサイズが削減され、インデックスの使用頻度の高い部分がメモリに収まる可能性が高くなります。
- クエリまたは更新が単一のパーティションの大部分にアクセスする場合、サーバーはインデックスとテーブル全体に分散したランダムアクセス読み取りを使用する代わりにパーティションのシーケンシャルスキャンを実行するため、パフォーマンスが向上する場合があります。
- パーティション設計にその要件を計画している場合、パーティションを追加または削除することにより、バルクロード（またはアンロード）を実装できます。`ALTER TABLE` は、一括操作よりもはるかに高速です。また、大量の `DELETE` によって引き起こされる `VACUUM` のオーバーヘッドを完全に回避します。
- 使用頻度の低いデータは、より安価な（またはより遅い）ストレージメディアに移行できます。

テーブルのパーティション化は、テーブルが非常に大きくなる場合にのみ価値があります。テーブルがパーティション化の恩恵を受ける正確なポイントは、アプリケーションによって異なります。経験則として、テーブルのサイズはデータベースサーバの物理メモリを超える必要があります。

EDB Postgres Advanced Server でサポートされているデータベース互換性機能の詳細については、

Database Compatibility: Table Partitioning を参照してください。

第3章 Database administration

この章では、EDB Postgres Advanced Server データベースの管理を支援する機能について説明します。

configuration_parameters index_advisor sql_profiler pgsnmpd edb_audit_logging unicode_collation_algorithm

3.1 Configuration parameters

EDB Postgres Advanced Server 構成パラメーターは、データファイルとログファイルの場所、接続、認証とセキュリティの設定、リソースの割り当てと消費、アーカイブとレプリケーションの設定、エラーログと統計の収集、最適化など、データベースサーバーの動作と環境のさまざまな側面を制御しますパフォーマンスチューニング、ロケールとフォーマット設定

EDB Postgres Advanced Server にのみ適用される構成パラメーターは、

Summary of configuration parameters に記載されています。

構成パラメーターの詳細については、[PostgreSQL core documentation](#) を参照してください。

setting_new_parameters summary_of_configuration_parameters configuration_parameters_by_functionity

3.1.1 Setting configuration parameters

名前/値のペアを使用して各構成パラメーターを設定します。パラメーター名は大文字と小文字を区別しません。パラメーター名は通常、オプションの等号(=)で値と区切ります。

この例は、`postgresql.conf` ファイルのいくつかの構成パラメーター設定を示しています。

```
# This is a comment

log_connections = yes
log_destination = syslog
search_path = "$user", public
shared_buffers = 128MB
```

パラメーター値は、次の5つのタイプのいずれかとして指定されます。

- ブール。許容値は、`on`、`off`、`true`、`false`、`yes`、`no`、`1`、`0`、またはこれらの明確な接頭辞です。
- 整数。小数部のない数値。
- 浮動小数点。小数点で区切られたオプションの小数部を持つ数値。

- 文字列。値が単純な識別子または数字でない場合、つまり値にスペースやその他の句読点が含まれる場合、一重引用符で囲まれたテキスト値。
- 列挙。特定の文字列値のセット。許可される値はシステムビュー `pg_settings.enumvals` にあります。列挙値は大文字と小文字を区別しません。

一部の設定では、メモリまたは時間の値を指定します。これらにはそれぞれ、キロバイト、ブロック（通常 8 キロバイト）、ミリ秒、秒、または分の暗黙的な単位があります。システムビュー `pg_settings.unit` を参照すると、デフォルトの単位を見つけることができます。別のを明示的に指定できます。

有効なメモリ単位は **kB** (キロバイト)、**MB** (メガバイト)、および **GB** (ギガバイト) です。有効な時間単位は、**ms** (ミリ秒)、**s** (秒)、**min** (分)、**h** (時間)、および **d** (日) です。メモリユニットの乗数は 1024 です。

構成パラメーター設定は、さまざまな方法で設定できます。

- EDB Postgres Advanced Server データベース製品をビルドするときに、多くのパラメーター設定が設定されます。これらは読み取り専用のパラメーターであり、値を変更することはできません。データベースの作成時に、データベースごとにいくつかのパラメーターも永続的に設定されます。これらのパラメーターは読み取り専用であり、データベースに対して後で変更することはできません。
- データベースクラスター全体のほぼすべての構成可能なパラメーターの初期設定は、構成ファイル `postgresql.conf` にリストされています。これらの設定は、データベースサーバの起動または再起動時に有効になります。これらの初期パラメーター設定の一部はオーバーライドできます。すべての構成パラメーターには、明示的にオーバーライドされない場合に有効な組み込みのデフォルト設定があります。
- `postgresql.conf` ファイルの構成パラメーターは、同じパラメーターが `postgresql.auto.conf` ファイルに含まれている場合にオーバーライドされます。`ALTER SYSTEM` コマンドは、`postgresql.auto.conf` ファイルの構成パラメーターを管理するために使用されます。
- データベースサーバの実行中に、構成ファイルのパラメーター設定を変更できます。構成ファイルがリロードされる（つまり `SIGHUP` シグナルが発行される）と、特定のパラメータータイプについて、変更されたパラメーター設定がすぐに有効になります。これらのパラメータータイプの一部では、リロード直後に現在実行中のセッションで新しい設定を使用できます。これらのパラメータータイプの他の場合、新しい設定を使用するには新しいセッションを開始する必要があります。ただし、他のパラメータータイプの場合、変更した設定はデータベースサーバを停止して再起動するまで有効になりません。構成ファイルをリロードする方法については、

[PostgreSQL core documentation](#) を参照してください。

- SQL コマンド `ALTER DATABASE`、`ALTER ROLE`、または `ALTER ROLE IN DATABASE` を使用して、特定のパラメーター設定を変更できます。変更したパラメーター設定は、コマンドを実行した後に新しいセッションで有効になります。`ALTER DATABASE` は、指定されたデータベースに接続する新しいセッションに影響します。`ALTER ROLE` は、指定されたロールによって開始された新しいセッションに影響します。`ALTER ROLE IN DATABASE` は、指定されたデータベースに接続する指定されたロールによって開始される新しいセッションに影響します。これらの SQL コマンドで確立されたパラメーター設定は、データベースサーバを再起動しても無期限に有効であり、他の方法で確立された設定をオーバーライドします。`ALTER DATABASE`、`ALTER ROLE`、または `ALTER ROLE IN DATABASE` コマンドを使用して確立されたパラメーター設定は、次のいずれかによってのみ変更できます。
- これらのコマンドを別のパラメーター値で再発行します。

- `SET parameter TO DEFAULT` 句または `RESET parameter` 句のいずれかを使用してこれらのコマンドを発行します。これらの句は、パラメーターを他のメソッドで設定された設定を使用するように変更します。これらの SQL コマンドの正確な構文については、

[PostgreSQL core documentation](#) を参照してください。

- `PGOPTIONS` 環境変数を使用するか、EDB-PSQL または PSQL コマンドラインプログラムで `SET` コマンドを使用して、個々のセッションの期間中に特定のパラメーター設定を変更できます。この方法で行われたパラメーター設定は、前に説明した方法のいずれかを使用して確立された設定をオーバーライドしますが、そのセッション中にのみです。

postgresql.conf ファイルの変更

`postgresql.conf` ファイルの構成パラメーターは、監査、認証、暗号化、およびその他の動作に関するサーバーの動作を指定します。Linux および Windows ホストでは、`postgresql.conf` ファイルは EDB Postgres Advanced Server インストールの下の `data` ディレクトリにあります。

シャープ記号 (#) が前に付いているパラメーターは、デフォルト値に設定されます。パラメーター値を変更するには、ポンド記号を削除して新しい値を入力します。パラメーターを設定または変更した後、新しいパラメーター値を有効にするには、サーバーを `reload` または `restart` する必要があります。

`postgresql.conf` ファイルでは、一部のパラメーターに `change requires restart` を示すコメントが含まれています。サーバーの再起動が必要なパラメーターのリストを表示するには、`psql` コマンドラインで次のクエリを実行します。

```
SELECT name FROM pg_settings WHERE context = postmaster;
```

pg_hba.conf ファイルの変更

適切な認証方法は、保護とセキュリティを提供します。`pg_hba.conf` ファイルのエントリは、サーバーが接続するクライアントに使用する認証方法を指定します。サーバーに接続する前に、`pg_hba.conf` ファイルで指定された認証プロパティの変更が必要になる場合があります。

`initdb` ユーティリティを呼び出してクラスターを作成すると、ユーティリティはそのクラスター用に、接続するクライアントに必要な認証のタイプを指定する `pg_hba.conf` ファイルを作成します。このファイルは変更できます。`pg_hba.conf` ファイルの認証設定を変更した後、サーバーを再起動して変更を適用します。

認証と `pg_hba.conf` ファイルの変更の詳細については、[PostgreSQL core documentation](#) を参照してください。

サーバーは接続要求を受信すると、データベースへの接続を許可する前に、`pg_hba.conf` ファイルの認証設定に対して提供された資格情報を検証します。サーバーへの接続を認証するために `pg_hba.conf` ファイルエントリをログに記録するには、`postgresql.conf` ファイルで `log_connections` パラメーターを `ON` に設定します。

レコードは、接続タイプ、データベース名、ユーザー名、クライアント IP アドレス、および `pg_hba.conf` ファイルでこれらのパラメーターが一致したときに接続を承認する認証方法を指定します。サーバーへの接続が承認されると、一致した行番号と認証レコードを `pg_hba.conf` ファイルから確認できます。

この例は、認証に成功した後の有効な `pg_hba.conf` エントリのログ詳細を示しています。

```
2020-05-08 10:42:17 IST LOG:  connection received: host=[local]
2020-05-08 10:42:17 IST LOG:  connection authorized: user=u1 database=edb
application_name=psql
2020-05-08 10:42:17 IST DETAIL:  Connection matched pg_hba.conf line 84:
"local    all             all             md5"
```

3.1.2 Summary of configuration parameters

この概要表には、すべての EDB Postgres Advanced Server 構成パラメーターと、パラメーターの多数のキー属性がリストされています。

これらの属性は、サマリー表の次の列で説明されます。

パラメーター。構成パラメーター名。

効果の範囲。構成パラメーター設定の効果の範囲：

- **Cluster**。この設定は、データベースクラスタ全体、つまりデータベースサーバーインスタンスによって管理されるすべてのデータベースに影響します。
- **Database**。設定はデータベースによって異なる場合があります、データベースの作成時に確立されます。ロケール設定に関連する少数のパラメーターに適用されます。
- **Session**。設定は、個々のセッションの粒度まで異なる場合があります。

言い換えれば、次のエンティティに対してさまざまな設定を行うことができ、このリストの後者の設定が前の設定をオーバーライドします。

- データベースクラスタ全体
- データベースクラスタ内の特定のデータベース
- 特定の役割
- 特定のデータベースに接続したときの特定のロール
- 特定のセッション

有効になる時期変更したパラメーター設定が有効になるとき。

- **Preset**。EDB Postgres Advanced Server 製品がビルドされるか、特定のデータベースが作成されると確立されます。これは読み取り専用のパラメーターであり、変更できません。
- **Restart**。データベースサーバを再起動する必要があります。
- **Reload**。構成ファイルをリロードするか、データベースサーバを再起動する必要があります。
- **Immediate**。PGOPTIONS 環境変数または SET コマンドを使用して現在のセッションで設定を変更した場合、セッションですぐに有効になります。

ALTER DATABASE、ALTER ROLE、または ALTER ROLE IN DATABASE コマンドを使用して設定を変更した場合、新しいセッションで有効です。

許可ユーザー。パラメーター設定を有効にするために使用する必要があるオペレーティングシステムアカウントまたはデータベースロールの種類。

- **EPAS service account**。EDB Postgres Advanced Server サービスアカウント (Oracle データベースと互換性のあるインストールの場合は **enterprisedb**、PostgreSQL 互換モードのインストールの場合は **postgres**)。
- **Superuser**。スーパーユーザー権限を持つデータベースロール。
- **User**。影響を受けるデータベースオブジェクト、つまり、**ALTER** コマンドで変更するデータベースまたはロール。
- **n/a** – ユーザーはパラメーター設定を変更できません。

説明。構成パラメーターの簡単な説明。

EPAS のみ。構成パラメーターは EDB Postgres Advanced Server にのみ適用されます。この列にエントリがない場合、構成パラメーターが PostgreSQL にも適用されることを示します。

注釈: 多くのパラメーターは変更できません。これらは、[説明] 列で**注: 内部使用のみ**と指定されています。

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>allow_system_table_mods</i>	Cluster	Restart	EPAS service account	システムテーブルの構造を変更できます。	
<i>application_name</i>	Session	Immediate	User	統計とログで報告されるアプリケーション名を設定します。	
<i>archive_command</i>	Cluster	Reload	EPAS service account	WAL ファイルをアーカイブするため呼び出されるシェルコマンドを設定します。	
<i>archive_mode</i>	Cluster	Restart	EPAS service account	<code>'archive_command'</code> を使用して WAL ファイルのアーカイブを許可します。	
<i>archive_timeout</i>	Cluster	Reload	EPAS service account	N 秒以内に新しいファイルが開始されなかった場合、次の xlog ファイルに強制的に切り替えます。	
<i>array_nulls</i>	Session	Immediate	User	配列の NULL 要素の入力を有効にします。	
<i>authentication_timeout</i>	Cluster	Reload	EPAS service account	クライアント認証を完了するまでの最大許容時間を設定します。	
<i>autovacuum</i>	Cluster	Reload	EPAS service account	autovacuum サブプロセスを開始します。	
<i>autovacuum_analyze_scale_factor</i>	Cluster	Reload	EPAS service account	<code>'reluples'</code> の割合として分析する前のタプルの挿入、更新、または削除の数。	
<i>autovacuum_analyze_threshold</i>	Cluster	Reload	EPAS service account	分析前のタプルの挿入、更新、または削除の最小数。	
<i>autovacuum_freeze_max_age</i>	Cluster	Restart	EPAS service account	トランザクション ID のラップアラウンドを防ぐためにテーブルを自動バキュームする期間。	
<i>autovacuum_max_workers</i>	Cluster	Restart	EPAS service account	同時に実行される自動バキュームワーカープロセスの最大数を設定します。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>autovacuum_multixact_freeze_max_age</i>	Cluster	Restart	EPAS service account	マルチランザクシヨンのラップアラウンドを防ぐためにテーブルを自動バキュームするマルチランザクシヨンの経過時間。	
<i>autovacuum_naptime</i>	Cluster	Reload	EPAS service account	自動バキュームの実行間のスリープ時間。	
<i>autovacuum_vacuum_cost_delay</i>	Cluster	Reload	EPAS service account	自動バキュームのミリ秒単位のバキュームコスト遅延。	
<i>autovacuum_vacuum_cost_limit</i>	Cluster	Reload	EPAS service account	オートバキュームの場合、昼寝前に利用可能なバキュームコスト。	
<i>autovacuum_vacuum_scale_factor</i>	Cluster	Reload	EPAS service account	*reluples`の割合としてのバキューム前のタプルの更新または削除の数。	
<i>autovacuum_vacuum_threshold</i>	Cluster	Reload	EPAS service account	バキューム前のタプルの更新または削除の最小数。	
<i>autovacuum_work_mem</i>	Cluster	Reload	EPAS service account	各自動バキュームワークプロセスが使用する最大メモリを設定します。	
<i>backslash_quote</i>	Session	Immediate	User	文字列リテラルで`""`を許可するかどうかを設定します。	
<i>bgwriter_delay</i>	Cluster	Reload	EPAS service account	バックグラウンドライタのラウンド間のスリープ時間。	
<i>bgwriter_lru_maxpages</i>	Cluster	Reload	EPAS service account	ラウンドごとにフラッシュする LRU ページのバックグラウンドライタの最大数。	
<i>bgwriter_lru_multiplier</i>	Cluster	Reload	EPAS service account	ラウンドごとに解放する平均バッファ使用量の倍数。	
<i>block_size</i>	Cluster	Preset	n/a	ディस्कブロックのサイズを示します。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>bonjour</i>	Cluster	Restart	EPAS service account	Bonjour を介したサーバーのアドバタイズを有効にします。	
<i>bonjour_name</i>	Cluster	Restart	EPAS service account	Bonjour サービス名を設定します。	
<i>bytea_output</i>	Session	Immediate	User	'bytea' の出力フォーマットを設定します。	
<i>check_function_bodies</i>	Session	Immediate	User	'CREATE FUNCTION' 中にファンクション本体をチェックします。	
<i>checkpoint_completion_target</i>	Cluster	Reload	EPAS service account	チェックポイント間隔の割合としての、チェックポイント中にデータベースのフラッシュに費やされた時間。	
<i>checkpoint_segments</i>	Deprecated	Deprecated	Deprecated	サーバーが起動しないようにするパラメーターの値を指定します。	
<i>checkpoint_timeout</i>	Cluster	Reload	EPAS service account	自動 WAL チェックポイント間の最大時間を設定します。	
<i>checkpoint_warning</i>	Cluster	Reload	EPAS service account	チェックポイントセグメントがこれよりも頻繁に充填される場合の警告を有効にします。	
<i>client_encoding</i>	Session	Immediate	User	クライアントの文字セットエンコーディングを設定します。	
<i>client_min_messages</i>	Session	Immediate	User	クライアントに送信されるメッセージレベルを設定します。	
<i>commit_delay</i>	Session	Immediate	Superuser	トランザクションのコミットと WAL のディスクへのフラッシュ間の遅延をマイクロ秒で設定します。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>commit_siblings</i>	Session	Immediate	User	`commit_delay` を実行する前に、最小の同時オープメントランザクションを設定します。	
<i>config_file</i>	Cluster	Restart	EPAS service account	サーバーのメイン構成ファイルを設定します。	
<i>constraint_exclusion</i>	Session	Immediate	User	プランナーが制約を使用してクエリを最適化できるようにします。	
<i>cpu_index_tuple_cost</i>	Session	Immediate	User	インデックススキャン中に各インデックスエントリを処理するコストのプランナの見積もりを設定します。	
<i>cpu_operator_cost</i>	Session	Immediate	User	各オペレーターまたは関数呼び出しを処理するコストのプランナーの見積もりを設定します。	
<i>cpu_tuple_cost</i>	Session	Immediate	User	各タプル (行) の処理コストのプランナーの見積もりを設定します。	
<i>cursor_tuple_fraction</i>	Session	Immediate	User	取得するカーソルの行の割合のプランナの推定値を設定します。	
<i>custom_variable_classes</i>	Cluster	Reload	EPAS service account	EDB Postgres Advanced Server 9.2 で非推奨になりました。	X
<i>data_checksums</i>	Cluster	Preset	n/a	このクラスターのデータチェックサムがオンかどうかを示します。	
<i>data_directory</i>	Cluster	Restart	EPAS service account	サーバーのデータディレクトリを設定します。	
<i>datestyle</i>	Session	Immediate	User	日付と時刻の値の表示形式を設定します。	
<i>db_dialect</i>	Session	Immediate	User	組み込みの名前空間の優先順位を設定します。	X

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>dbms_alert.max_alerts</i>	Cluster	Restart	EPAS service account	アラートの最大数を設定します。	X
<i>dbms_pipe.total_message_buffer</i>	Cluster	Restart	EPAS service account	DBMS_PIPE パッケージに使用されるバッファの合計サイズを指定します。	X
<i>db_user_namespace</i>	Cluster	Reload	EPAS service account	データベースごとのユーザー名を有効にします。	
<i>deadlock_timeout</i>	Session	Immediate	Superuser	デッドロックをチェックするまでにロックを待機する時間を設定します。	
<i>debug_assertions</i>	Cluster	Preset	n/a	さまざまなアサーション チェックをオンにします。(EPAS ビルドではサポートされていません。)	
<i>debug_pretty_print</i>	Session	Immediate	User	解析およびプランツリー表示をインデントします。	
<i>debug_print_parse</i>	Session	Immediate	User	各クエリの解析ツリーをログに記録します。	
<i>debug_print_plan</i>	Session	Immediate	User	各クエリの実行プランをログに記録します。	
<i>debug_print_rewritten</i>	Session	Immediate	User	各クエリを書き換えられた解析ツリーをログに記録します。	
<i>default_heap_fillfactor</i>	Session	Immediate	User	デフォルトでこのヒープ FILL FACTOR で新しいテーブルを作成します。	X
<i>default_statistics_target</i>	Session	Immediate	User	デフォルトの統計ターゲットを設定します。	
<i>default_tablespace</i>	Session	Immediate	User	テーブルとインデックスを作成するデフォルトのテーブルスペースを設定します。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>default_text_search_config</i>	Session	Immediate	User	デフォルトのテキスト検索構成を設定します。	
<i>de-fault_transaction_deferrable</i>	Session	Immediate	User	新しいトランザクションのデフォルトの遅延可能ステータスを設定します。	
<i>de-fault_transaction_isolation</i>	Session	Immediate	User	新しい各トランザクションのトランザクション分離レベルを設定します。	
<i>de-fault_transaction_read_only</i>	Session	Immediate	User	新しいトランザクションのデフォルトの読み取り専用ステータスを設定します。	
<i>default_with_oids</i>	Session	Immediate	User	デフォルトで OID を使用して新しいテーブルを作成します。	
<i>default_with_rowids</i>	Session	Immediate	User	デフォルトで ROWID サポート (インデックス付きの ROWID) を使用して新しいテーブルを作成します。	X
<i>dynamic_library_path</i>	Session	Immediate	Superuser	動的にロード可能なモジュールのパスを設定します。	
<i>dynamic_shared_memory_type</i>	Cluster	Restart	EPAS service account	使用される動的共有メモリーの実装を選択します。	
<i>edb_audit</i>	Cluster	Reload	EPAS service account	EDB Auditing で XML または CSV 形式で監査レポートを作成できるようにします。	X
<i>edb_audit_archiver</i>	Cluster	Restart	EPAS service account	監査ログアーカイバプロセスを有効にします。	X
<i>edb_audit_archiver_timeout</i>	Cluster	Reload	EPAS service account	<code>edb_audit_archiver_timeout</code> パラメーターに基づいて監査ログファイルを確認します。	X

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<code>edb_audit_archiver_filename_prefix</code>	Cluster	Reload	EPAS service account	<code>`edb_audit_archiver_filename_prefix`</code> を持つ監査ログファイルが圧縮または有効期限切れの対象であると判断します。パラメーターは <code>`edb_audit_filename`</code> と揃える必要があります。	X
<code>edb_audit_archiver_compress_timeout</code>	Cluster	Reload	EPAS service account	監査ログが圧縮の対象になるまでの時間が秒単位で表示されます。	X
<code>edb_audit_archiver_compress_blocksize</code>	Cluster	Reload	EPAS service account	監査ログが圧縮の対象になるまでの合計サイズをメガバイト単位で示します。	X
<code>edb_audit_archiver_compress_level</code>	Cluster	Reload	EPAS service account	監査ログファイルを圧縮します。	X
<code>edb_audit_archiver_compress_command</code>	Cluster	Reload	EPAS service account	既に圧縮されているログファイルのサフックスを決定します。パラメーターは <code>edb_audit_archiver_compress_command</code> と揃える必要があります。	X
<code>edb_audit_archiver_expire_time</code>	Cluster	Reload	EPAS service account	監査ログが有効期限切れになるまでの時間が秒単位で表示されます。	X
<code>edb_audit_archiver_expire_size</code>	Cluster	Reload	EPAS service account	監査ログが有効期限切れになるまでの合計サイズを MB 単位で示します。	X
<code>edb_audit_archiver_expire_command</code>	Cluster	Reload	EPAS service account	期限切れの監査ログを削除する前に実行します。	X
<code>edb_audit_archiver_sort_file</code>	Cluster	Reload	EPAS service account	最も古いログファイルを識別し、アルファベット順または <code>`mtime`</code> に基づいて並べ替えます。	X

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>edb_audit_connect</i>	Cluster	Reload	EPAS service account	成功した各接続を監査します。	X
<i>edb_audit_destination</i>	Cluster	Reload	EPAS service account	監査ファイルの宛先ディレクトリとして <code>`edb_audit_directory`</code> または <code>syslog</code> を設定します。 <code>syslog</code> 設定は、Linux システムでのみ有効です。	X
<i>edb_audit_directory</i>	Cluster	Reload	EPAS service account	監査ファイルの宛先ディレクトリを設定します。	X
<i>edb_audit_disconnect</i>	Cluster	Reload	EPAS service account	セッションの終了を監査します。	X
<i>edb_audit_filename</i>	Cluster	Reload	EPAS service account	監査ファイルのファイル名パターンを設定します。	X
<i>edb_audit_rotation_day</i>	Cluster	Reload	EPAS service account	曜日に基づいたログファイルの自動ローテーション。	X
<i>edb_audit_rotation_seconds</i>	Cluster	Reload	EPAS service account	自動ログファイルローテーションは N 秒後に発生します。	X
<i>edb_audit_rotation_size</i>	Cluster	Reload	EPAS service account	自動ログファイルローテーションは、N メガバイト後に発生します。	X
<i>edb_audit_statement</i>	Cluster	Reload	EPAS service account	監査するステートメントのタイプを設定します。	X
<i>edb_audit_tag</i>	Session	Immediate	User	監査ログに含めるタグを指定します。	X
<i>edb_connectby_order</i>	Session	Immediate	User	<code>`ORDER BY`</code> を使用しない <code>`CONNECT BY`</code> クエリの結果を深さ優先でソートします。注：内部使用のみ。	X
<i>edb_custom_plan_tries</i>	Session	Immediate	User	プランナーが汎用実行プランを選択する前にプランナーが検討するカスタム実行プランの数を指定します。	X
<i>edb_data_redaction</i>	Session	Immediate	User	データリダクションを有効にします。	X

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>edb_dynatune</i>	Cluster	Restart	EPAS service account	edb 使用率を設定します。	X
<i>edb_dynatune_profile</i>	Cluster	Restart	EPAS service account	dynatune のワークロードプロファイルを設定します。	X
<i>edb_enable_pruning</i>	Session	Immediate	User	プランナーがパーティションテーブルを早期に整理できるようにします。	X
<i>edb_log_every_bulk_value</i>	Session	Immediate	Superuser	一括処理のためにログに記録されるステートメントを設定します。	X
<i>edb_max_resource_groups</i>	Cluster	Restart	EPAS service account	同時に使用するリソースグループの最大数を指定します。	X
<i>edb_max_spins_per_delay</i>	Cluster	Restart	EPAS service account	ロックの待機中にセッションがスピ ンする回数を指定します。	X
<i>edb_redwood_date</i>	Session	Immediate	User	'DATE' が 'TIMESTAMP' のように動作するかどうかを決定します。	X
<i>edb_redwood_greatest_least</i>	Session	Immediate	User	'GREATEST' および 'LEAST' 関数が NULL パラメーターを処理する方法 を決定します。	X
<i>edb_redwood_raw_names</i>	Session	Immediate	User	Redwood インタフェースから Post- greSQL システムカタログに保存され ている変更されていない名前を返し ます。	X
<i>edb_redwood_strings</i>	Session	Immediate	User	テキスト値と連結すると、 <i>NULL</i> を 空の文字列として扱います。	X
<i>edb_resource_group</i>	Session	Immediate	User	現在のプロセスが使用するリソース グループを指定します。	X
<i>edb_sql_protect.enabled</i>	Cluster	Reload	EPAS service account	SQL/Protect がクエリを追跡するかど うかを定義します。	X

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>edb_sql_protect.level</i>	Cluster	Reload	EPAS service account	イベントが検出されたときの SQL/Protect の動作を定義します。	X
<i>edb_sql_protect.max_protected_transactions</i>	Cluster	Restart	EPAS service account	ロールごとに SQL/Protect によって保護されるリレーションの最大数を設定します。	X
<i>edb_sql_protect.max_protected_queries</i>	Cluster	Restart	EPAS service account	SQL/Protect によって保護されるロールの最大数を設定します。	X
<i>edb_sql_protect.max_queries_per_cluster</i>	Cluster	Restart	EPAS service account	SQL/Protect によって保存する問題のあるクエリの最大数を設定します。	X
<i>edb_stmt_level_tx</i>	Session	Immediate	User	トランザクションのアポートを要求する代わりに、エラーで続行できます。	X
<i>edb_wait_states.directory</i>	Cluster	Restart	EPAS service account	このディレクトリに EDB の待機状態ログを保存します。	X
<i>edb_wait_states.retention_period</i>	Cluster	Reload	EPAS service account	保持期間後に EDB 待機状態ログファイルを削除します。	X
<i>edb_wait_states.sampling_interval</i>	Cluster	Reload	EPAS service account	2 つの EDB 待機状態のサンプリングサイクル間の間隔。	X
<i>edbldr.empty_csv_field</i>	Session	Immediate	Superuser	EDB*Loader が空の文字列を処理する方法を指定します。	X
<i>effective_cache_size</i>	Session	Immediate	User	ディस्कキャッシュのサイズに関するプランナーの仮定を設定します。	
<i>effective_io_concurrency</i>	Session	Immediate	User	ディस्कサブシステムが効率的に処理できる同時要求の数。	
<i>enable_bitmapscan</i>	Session	Immediate	User	プランナーによるビットマップスキャンプランの使用を有効にします。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>enable_hashagg</i>	Session	Immediate	User	プランナーによるハッシュ集計プランの使用を有効にします。	
<i>enable_hashjoin</i>	Session	Immediate	User	プランナーによるハッシュ結合プランの使用を有効にします。	
<i>enable_hints</i>	Session	Immediate	User	SQL ステートメントでオプティマイザヒントを有効にします。	X
<i>enable_indexonlyscan</i>	Session	Immediate	User	プランナーによるインデックス専用スキャンプランの使用を有効にします。	
<i>enable_indexscan</i>	Session	Immediate	User	プランナーによるインデックススキャンプランの使用を有効にします。	
<i>enable_material</i>	Session	Immediate	User	プランナーがマテリアライゼーションを使用できるようにします。	
<i>enable_mergejoin</i>	Session	Immediate	User	プランナーがマージ結合プランを使用できるようにします。	
<i>enable_nestloop</i>	Session	Immediate	User	プランナーでのネストされたループの結合プランの使用を有効にします。	
<i>enable_seqscan</i>	Session	Immediate	User	プランナーによるシーケンシャルスキャンプランの使用を有効にします。	
<i>enable_sort</i>	Session	Immediate	User	プランナーが明示的な並べ替え手順を使用できるようにします。	
<i>enable_tidscan</i>	Session	Immediate	User	プランナーによる TID スキャンプランの使用を有効にします。	
<i>escape_string_warning</i>	Session	Immediate	User	通常の文字列リテラルでのバックslash エスケープについて警告します。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>event_source</i>	Cluster	Restart	EPAS service account	イベントログで PostgreSQL メッセージを識別するために使用されるアプリケーション名を設定します。	
<i>exit_on_error</i>	Session	Immediate	User	エラーが発生するとセッションを終了します。	
<i>external_pid_file</i>	Cluster	Restart	EPAS service account	指定されたファイルに postmaster PID を書き込みます。	
<i>extra_float_digits</i>	Session	Immediate	User	浮動小数点値の表示桁数を設定します。	
<i>from_collapse_limit</i>	Session	Immediate	User	サブクエリが折りたたまれない `FROM`-list サイズを設定します。	
<i>fsync</i>	Cluster	Reload	EPAS service account	更新をディスクに強制的に同期します。	
<i>full_page_writes</i>	Cluster	Reload	EPAS service account	チェックポイント後に最初に変更されたときに、ページ全体を WAL に書き込みます。	
<i>geqo</i>	Session	Immediate	User	ジェネリッククエリの最適化を有効にします。	
<i>geqo_effort</i>	Session	Immediate	User	GEQO : エフォートは、他の GEQO パラメーターのデフォルトを設定するために使用されます。	
<i>geqo_generations</i>	Session	Immediate	User	GEQO : アルゴリズムの反復数。	
<i>geqo_pool_size</i>	Session	Immediate	User	GEQO : 母集団内の個体の数。	
<i>geqo_seed</i>	Session	Immediate	User	GEQO : ランダムパス選択のシード。	
<i>geqo_selection_bias</i>	Session	Immediate	User	GEQO : 集団内の選択圧。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>geqo_threshold</i>	Session	Immediate	User	GEQO を使用する `FROM` アイテムのしきい値を設定します。	
<i>gin_fuzzy_search_limit</i>	Session	Immediate	User	GIN による完全一致検索の最大結果を設定します。	
<i>hba_file</i>	Cluster	Restart	EPAS service account	サーバーの「hba」構成ファイルを設定します。	
<i>hot_standby</i>	Cluster	Restart	EPAS service account	リカバリ中の接続とクエリを許可します。	
<i>hot_standby_feedback</i>	Cluster	Reload	EPAS service account	クエリの競合を回避するホットスタンバイからプライマリへのフィードバックを許可します。	
<i>huge_pages</i>	Cluster	Restart	EPAS service account	Linux で huge page を使用します。	
<i>icu_short_form</i>	Database	Preset	n/a	ICU の照合順構成を表示します。	X
<i>ident_file</i>	Cluster	Restart	EPAS service account	サーバーの「ident」構成ファイルを設定します。	
<i>ignore_checksum_failure</i>	Session	Immediate	Superuser	チェックサムに失敗しても処理を続行します。	
<i>ignore_system_indexes</i>	Cluster/Session	Reload/Immediate	EPAS service account/User	システムインデックスからの読み取りを無効にします。(セッション開始時に `PGOPTIONS` で設定することもできます。)	
<i>index_advisor.enabled</i>	Session	Immediate	User	Index Advisor プラグインを有効にします。	X
<i>integer_datetimes</i>	Cluster	Preset	n/a	日時は整数ベースです。	
<i>intervalStyle</i>	Session	Immediate	User	間隔値の表示形式を設定します。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>join_collapse_limit</i>	Session	Immediate	User	<i>JOIN</i> コンストラクトがフラット化されない `FROM-list` サイズを設定します。	
<i>krb_caseins_users</i>	Cluster	Reload	EPAS service account	Kerberos および GSSAPI のユーザー名を大文字と小文字を区別しないかどうかを設定します。	
<i>krb_server_keyfile</i>	Cluster	Reload	EPAS service account	Kerberos サーバーキーファイルの場合所を設定します。	
<i>lc_collate</i>	Database	Preset	n/a	照合順のロケールを表示します。	
<i>lc_ctype</i>	Database	Preset	n/a	文字分類と大文字と小文字の変換ロケールを表示します。	
<i>lc_messages</i>	Session	Immediate	Superuser	メッセージが表示される言語を設定します。	
<i>lc_monetary</i>	Session	Immediate	User	金額をフォーマットするロケールを設定します。	
<i>lc_numeric</i>	Session	Immediate	User	数値をフォーマットするためのロケールを設定します。	
<i>lc_time</i>	Session	Immediate	User	日付と時刻の値をフォーマットするためのロケールを設定します。	
<i>listen_addresses</i>	Cluster	Restart	EPAS service account	リッスンするホスト名または IP アドレスを設定します。	
<i>local_preload_libraries</i>	Cluster/Session	Reload/Immediate	EPAS service account/User	各バックエンドにプリロードする共有ライブラリをリストします。(セッション開始時に `PGOPTIONS` で設定することもできます。)	
<i>lock_timeout</i>	Session	Immediate	User	ステートメントがロックを待機できる最大時間を設定します。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>lo_comput_privileges</i>	Session	Immediate	Superuser	ラージ オブジェクトの特権チェックの低位互換性モードを有効にします。	
<i>log_autovacuum_min_duration</i>	Cluster	Reload	EPAS service account	自動バキュームアクションがログに記録される最小実行時間を設定します。	
<i>log_checkpoints</i>	Cluster	Reload	EPAS service account	各チェックポイントをログに記録します。	
<i>log_connections</i>	Cluster/Session	Reload/Immediate	EPAS service account/User	成功した接続をそれぞれログに記録します。(セッション開始時に 'PGOPTIONS' で設定することもできます。)	
<i>log_destination</i>	Cluster	Reload	EPAS service account	サーバーログの出力先を設定します。	
<i>log_directory</i>	Cluster	Reload	EPAS service account	ログファイルの宛先ディレクトリを設定します。	
<i>log_disconnections</i>	Cluster/Session	Reload/Immediate	EPAS service account/User	期間を含む、セッションの終了をログに記録します。(セッション開始時に 'PGOPTIONS' で設定することもできます。)	
<i>log_duration</i>	Session	Immediate	Superuser	完了した各 SQL ステートメントの期間をログに記録します。	
<i>log_error_verbosity</i>	Session	Immediate	Superuser	ログに記録されるメッセージの詳細度を設定します。	
<i>log_executor_stats</i>	Session	Immediate	Superuser	エグゼキューターのパフォーマンス統計をサーバーログに書き込みます。	
<i>log_file_mode</i>	Cluster	Reload	EPAS service account	ログファイルのファイル権限を設定します。	
<i>log_filename</i>	Cluster	Reload	EPAS service account	ログファイルのファイル名パターンを設定します。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>log_hostname</i>	Cluster	Reload	EPAS service account	接続ログにホスト名を記録します。	
<i>log_line_prefix</i>	Cluster	Reload	EPAS service account	各ログ行に付加される情報を制御します。	
<i>log_lock_waits</i>	Session	Immediate	Superuser	長いロック待機をログに記録します。	
<i>log_min_duration_statement</i>	Session	Immediate	Superuser	ステートメントがログに記録される最小実行時間を設定します。	
<i>log_min_error_statement</i>	Session	Immediate	Superuser	このレベル以上のエラーを生成するすべてのステートメントがログに記録されます。	
<i>log_min_messages</i>	Session	Immediate	Superuser	ログに記録されるメッセージレベルを設定します。	
<i>log_parser_stats</i>	Session	Immediate	Superuser	パーサーのパフォーマンス統計をサーバーログに書き込みます。	
<i>log_planner_stats</i>	Session	Immediate	Superuser	Planner のパフォーマンス統計をサーバーログに書き込みます。	
<i>log_rotation_age</i>	Cluster	Reload	EPAS service account	自動ログファイルローテーションは N 分後に発生します。	
<i>log_rotation_size</i>	Cluster	Reload	EPAS service account	自動ログファイルローテーションは、N キロバイト後に発生します。	
<i>log_statement</i>	Session	Immediate	Superuser	ログに記録されるステートメントのタイプを設定します。	
<i>log_statement_stats</i>	Session	Immediate	Superuser	累積的なパフォーマンス統計をサーバーログに書き込みます。	
<i>log_temp_files</i>	Session	Immediate	Superuser	このキロバイト数より大きい一時ファイルの使用をログに記録します。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>log_timezone</i>	Cluster	Reload	EPAS service account	ログメッセージで使用するタイムゾーンを設定します。	
<i>log_truncate_on_rotation</i>	Cluster	Reload	EPAS service account	ログのローテーション中に同じ名前の既存のログファイルを切り捨てます。	
<i>logging_collector</i>	Cluster	Restart	EPAS service account	stderr 出力および/または csv ログをログファイルにキャプチャするサブプロセスを開始します。	
<i>maintenance_work_mem</i>	Session	Immediate	User	メンテナンス操作に使用される最大メモリを設定します。	
<i>max_connections</i>	Cluster	Restart	EPAS service account	同時接続の最大数を設定します。	
<i>max_files_per_process</i>	Cluster	Restart	EPAS service account	サーバープロセスごとに同時に開くファイルの最大数を設定します。	
<i>max_function_args</i>	Cluster	Preset	n/a	関数の引数の最大数を表示します。	
<i>max_identifier_length</i>	Cluster	Preset	n/a	識別子の最大長を表示します。	
<i>max_index_keys</i>	Cluster	Preset	n/a	インデックスキーの最大数を示します。	
<i>max_locks_per_transaction</i>	Cluster	Restart	EPAS service account	トランザクションごとのロックの最大数を設定します。	
<i>max_pred_locks_per_transaction</i>	Cluster	Restart	EPAS service account	トランザクションごとの述語ロックの最大数を設定します。	
<i>max_prepared_transactions</i>	Cluster	Restart	EPAS service account	同時に準備されるトランザクションの最大数を設定します。	
<i>max_replication_slots</i>	Cluster	Restart	EPAS service account	同時に定義されるレプリケーションスロットの最大数を設定します。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>max_stack_depth</i>	Session	Immediate	Superuser	最大スタックの深さをキロバイト単位で設定します。	
<i>max_standby_archive_delay</i>	Cluster	Reload	EPAS service account	ホットスタンバイサーバーがアーカイブされた WAL データを処理しているときにクエリをキャンセルするまでの最大遅延を設定します。	
<i>max_standby_streaming_delay</i>	Cluster	Reload	EPAS service account	ホットスタンバイサーバーがストリーミングされた WAL データを処理しているときにクエリをキャンセルするまでの最大遅延を設定します。	
<i>max_wal_senders</i>	Cluster	Restart	EPAS service account	同時に実行される WAL 送信者プロセスの最大数を設定します。	
<i>max_wal_size</i>	Cluster	Reload	EPAS service account	自動 WAL チェックポイント間で WAL が成長する最大サイズを設定します。デフォルトは 1GB です。	
<i>max_worker_processes</i>	Cluster	Restart	EPAS service account	同時ワーカースプロセスの最大数。	
<i>min_wal_size</i>	Cluster	Reload	EPAS service account	WAL ログが削除ではなくリサイクルされるしきい値を設定します。デフォルトは 80MB です。	
<i>nls_length_semantics</i>	Session	Immediate	Superuser	char、varchar、varchar2、および long 列に使用するセマンティクスを設定します。	X
<i>odbc_lib_path</i>	Cluster	Restart	EPAS service account	ODBC ライブラリのパスを設定します。	X
<i>optimizer_mode</i>	Session	Immediate	User	デフォルトのオプティマイザモード。	X
<i>oracle_home</i>	Cluster	Restart	EPAS service account	Oracle ホームディレクトリのパスを設定します。	X

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>password_encryption</i>	Session	Immediate	User	パスワードを暗号化します。	
<i>pg_prewarm.autoprewarm</i>	Cluster	Restart	EPAS service account	<i>autoprewarm</i> バックグラウンドワーカーを有効にします。	X
<i>pg_prewarm.autoprewarm_intel_cluster</i>	Cluster	Reload	EPAS service account	<i>autoprewarm</i> が共有バッファをダンプするまでの最小秒数を設定します。	X
<i>port</i>	Cluster	Restart	EPAS service account	サーバーがリッスンする TCP ポートを設定します。	
<i>post_auth_delay</i>	Cluster/Session	Reload/Immediate	EPAS service account/User	認証後の接続起動で N 秒待ちます。(セッション開始時に 'PGOPTIONS' で設定することもできます。)	
<i>pre_auth_delay</i>	Cluster	Reload	EPAS service account	接続開始時に認証まで N 秒待ちます。	
<i>qreplace_function</i>	Session	Immediate	Superuser	Query Replace 機能で使用される関数。注：内部使用のみ。	X
<i>query_rewrite_enabled</i>	Session	Immediate	User	Skills dchild テーブルは、その制約がクエリに一致する行がないことを保証する場合にスキップします。	X
<i>query_rewrite_integrity</i>	Session	Immediate	Superuser	クエリの書き換えを強制する程度を設定します。	X
<i>quote_all_identifiers</i>	Session	Immediate	User	SQL フラグメントを生成するときに、すべての識別子を引用します。	
<i>random_page_cost</i>	Session	Immediate	User	ノンシークエンシャルにフェッチされるディスクページのコストのプランナーの見積もりを設定します。	
<i>restart_after_crash</i>	Cluster	Reload	EPAS service account	バックエンドがクラッシュした後にサーバーを再初期化します。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>search_path</i>	Session	Immediate	User	スキーマ修飾されていない名前のスキーマ検索順序を設定します。	
<i>segment_size</i>	Cluster	Preset	n/a	ディस्कファイルごとのページ数を示します。	
<i>seq_page_cost</i>	Session	Immediate	User	順次フェッチされるディスクページのコストのプランナーの見積もりを設定します。	
<i>server_encoding</i>	Database	Preset	n/a	サーバー (データベース) の文字セットエンコーディングを設定します。	
<i>server_version</i>	Cluster	Preset	n/a	サーバーバージョンを表示します。	
<i>server_version_num</i>	Cluster	Preset	n/a	サーバーバージョンを整数で表示します。	
<i>session_preload_libraries</i>	Session	Immediate but only at connection start	Superuser	各バックエンドにプリロードする共有ライブラリをリストします。	
<i>session_replication_role</i>	Session	Immediate	Superuser	トリガーと書き換えルールセッションの動作を設定します。	
<i>shared_buffers</i>	Cluster	Restart	EPAS service account	サーバーが使用する共有メモリーバッファの数を設定します。	
<i>shared_preload_libraries</i>	Cluster	Restart	EPAS service account	サーバーにプリロードする共有ライブラリをリストします。	
<i>sql_inheritance</i>	Session	Immediate	User	さまざまなコマンドにデフォルトでサブクエリが含まれます。	
<i>ssl</i>	Cluster	Reload	EPAS service account	SSL 接続を有効にします。	
<i>ssl_ca_file</i>	Cluster	Reload	EPAS service account	SSL 認証局ファイルの場所。	
<i>ssl_cert_file</i>	Cluster	Reload	EPAS service account	SSL サーバー証明書ファイルの場所。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>ssl_ciphers</i>	Cluster	Reload	EPAS service account	許可される SSL 暗号のリストを設定します。	
<i>ssl_crl_file</i>	Cluster	Reload	EPAS service account	SSL 証明書失効リストファイルの場所。	
<i>ssl_ecdh_curve</i>	Cluster	Reload	EPAS service account	ECDH に使用するカーブを設定します。	
<i>ssl_key_file</i>	Cluster	Reload	EPAS service account	SSL サーバーのプライベートキーファイルの場所。	
<i>ssl_prefer_server_ciphers</i>	Cluster	Reload	EPAS service account	サーバーの暗号スイートの順序を優先します。	
<i>ssl_renegotiation_limit</i>	Session	Immediate	User	暗号化キーを再ネゴシエートする前に送信するトラフィックの量を設定します。	
<i>standard_conforming_strings</i>	Session	Immediate	User	'...' 文字列にバックスラッシュを文字どおりに処理させます。	
<i>statement_timeout</i>	Session	Immediate	User	ステートメントの最大許可時間を設定します。	
<i>stats_temp_directory</i>	Cluster	Reload	EPAS service account	指定されたディレクトリに一時統計ファイルを書き込みます。	
<i>superuser_reserved_connections</i>	Cluster	Restart	EPAS service account	スーパーユーザー用に予約される接続スロットの数を設定します。	
<i>synchronize_seqscans</i>	Session	Immediate	User	同期シークエンシャルスキャンを有効にします。	
<i>synchronous_commit</i>	Session	Immediate	User	コミット時に即時の fsync を設定します。	
<i>synchronous_standby_names</i>	Cluster	Reload	EPAS service account	潜在的な同期スタンバイの名前をリストします。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>syslog_facility</i>	Cluster	Reload	EPAS service account	syslog が有効になっているときに使用される syslog 「ファシリティ」を設定します。	
<i>syslog_ident</i>	Cluster	Reload	EPAS service account	syslog で PostgreSQL メッセージを識別するために使用されるプログラム名を設定します。	
<i>tcp_keepalives_count</i>	Session	Immediate	User	TCP キープアライブ再送信の最大数。	
<i>tcp_keepalives_idle</i>	Session	Immediate	User	TCP キープアライブを発行する間隔。	
<i>tcp_keepalives_interval</i>	Session	Immediate	User	TCP キープアライブ再送信の時間。	
<i>temp_buffers</i>	Session	Immediate	User	各セッションで使用される一時バッファの最大数を設定します。	
<i>temp_file_limit</i>	Session	Immediate	Superuser	各セッションで使用されるすべての一時ファイルの合計サイズを制限します。	
<i>temp_tablespaces</i>	Session	Immediate	User	一時テーブルとソートファイルに使用するテーブルスペースを設定します。	
<i>timed_statistics</i>	Session	Immediate	User	時限統計の記録を有効にします。	X
<i>timezone</i>	Session	Immediate	User	タイムスタンプを表示および解釈するためのタイムゾーンを設定します。	
<i>timezone_abbreviations</i>	Session	Immediate	User	タイムゾーンの略語のファイルを選択します。	
<i>trace_hints</i>	Session	Immediate	User	尊重されるヒントに関するデバッグ情報を出力します。	X
<i>trace_notify</i>	Session	Immediate	User	<code>LISTEN</code> および <code>NOTIFY</code> のデバッグ出力を生成します。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>trace_recovery_messages</i>	Cluster	Reload	EPAS service account	リカバリ関連のデバッグ情報のロギングを有効にします。	
<i>trace_sort</i>	Session	Immediate	User	並べ替えでのリソース使用量に関する情報を生成します。	
<i>track_activities</i>	Session	Immediate	Superuser	コマンドの実行に関する情報を収集します。	
<i>track_activity_query_size</i>	Cluster	Restart	EPAS service account	<code>'pg_stat_activity,current_query'</code> 用に予約されたサイズをバイト単位で設定します。	
<i>track_counts</i>	Session	Immediate	Superuser	データベースアクティビティに関する統計を収集します。	
<i>track_functions</i>	Session	Immediate	Superuser	データベースアクティビティに関する機能レベルの統計を収集します。	
<i>track_io_timing</i>	Session	Immediate	Superuser	データベース I/O アクティビティのタイミング統計を収集します。	
<i>transaction_deferrable</i>	Session	Immediate	User	シリアル化が失敗する可能性がなくなるまで、読み取り専用のシリアル化可能なトランザクションを遅延するかどうかを指定します。	
<i>transaction_isolation</i>	Session	Immediate	User	現在のトランザクションの分離レベルを設定します。	
<i>transaction_read_only</i>	Session	Immediate	User	現在のトランザクションの読み取り専用ステータスを設定します。	
<i>transform_null_equals</i>	Session	Immediate	User	<code>"expr=NULL"</code> を <code>"expr IS NULL"</code> として扱います。	
<i>unix_socket_directories</i>	Cluster	Restart	EPAS service account	Unix ドメインソケットが作成されるディレクトリを設定します。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>unix_socket_group</i>	Cluster	Restart	EPAS service account	Unix ドメインソケットの所有グループを設定します。	
<i>unix_socket_permissions</i>	Cluster	Restart	EPAS service account	Unix ドメインソケットのアクセス許可を設定します。	
<i>update_process_title</i>	Session	Immediate	Superuser	プロセスのタイトルを更新して、アクティブな SQL コマンドを表示します。	
<i>utl_encode.uidecode_redwood</i>	Session	Immediate	User	Oracle で作成された <code>uencode</code> データのデコードを許可します。	X
<i>utl_file.umask</i>	Session	Immediate	User	<code>'UTL_FILE'</code> パッケージで作成されたファイルに使用される <code>umask</code> 。	X
<i>vacuum_cost_delay</i>	Session	Immediate	User	ミリ秒単位のバキュームコスト遅延。	
<i>vacuum_cost_limit</i>	Session	Immediate	User	昼寝前に利用できるバキューム代。	
<i>vacuum_cost_page_dirty</i>	Session	Immediate	User	掃除機で汚れたページの掃除機のコスト。	
<i>vacuum_cost_page_hit</i>	Session	Immediate	User	バットファークャッシュで見つかったページのバキュームコスト。	
<i>vacuum_cost_page_miss</i>	Session	Immediate	User	バットファークャッシュで見つからないページのバキュームコスト。	
<i>vacuum_defer_cleanup_age</i>	Cluster	Reload	EPAS service account	<code>'VACUUM'</code> および <code>'HOT'</code> のクリーンアップを遅延させる必要があるトラップの数の数。	
<i>vacuum_freeze_min_age</i>	Session	Immediate	User	<code>'VACUUM'</code> がテーブル行をリリースする最小経過時間。	
<i>vacuum_freeze_table_age</i>	Session	Immediate	User	<code>'VACUUM'</code> がテーブル全体をスキップしてタブルをリリースする年齢。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>vacuum_multixact_freeze_min_age</i>	Session	Immediate	User	*VACUUM がテーブル行の MultiXactId をフリーズする最小年齢。	
<i>vacuum_multixact_freeze_table_age</i>	Session	Immediate	User	*VACUUM がテーブル全体をスキャンしてタブルをフリーズする Multixact の年齢。	
<i>wal_block_size</i>	Cluster	Preset	n/a	ブロックサイズを先行書き込みログに表示します。	
<i>wal_buffers</i>	Cluster	Restart	EPAS service account	WAL の共有メモリ内のディスクページバッファの数を設定します。	
<i>wal_keep_segments</i>	Cluster	Reload	EPAS service account	スタンバイサーバが保持する WAL ファイルの数を設定します。	
<i>wal_level</i>	Cluster	Restart	EPAS service account	WAL に書き込まれる情報のレベルを設定します。	
<i>wal_log_hints</i>	Cluster	Restart	EPAS service account	チェックポイントの後に最初に変更されたときに、重要ではない変更の場合でも、ページ全体を WAL に書き込みます。	
<i>wal_receiver_status_interval</i>	Cluster	Reload	EPAS service account	プライマリへの WAL 受信機ステータスレポート間の最大間隔を設定します。	
<i>wal_receiver_timeout</i>	Cluster	Reload	EPAS service account	プライマリからデータを受信するための最大待機時間を設定します。	
<i>wal_segment_size</i>	Cluster	Preset	n/a	先行書き込みログセグメントごとのページ数を示します。	
<i>wal_sender_timeout</i>	Cluster	Reload	EPAS service account	WAL レプリケーションを待機する最大時間を設定します。	

次のページに続く

表 1 – 前のページからの続き

Parameter	Scope of effect	When takes effect	Authorized	Description	EPAS only
<i>wal_sync_method</i>	Cluster	Reload	EPAS service account	WAL 更新をディスクに強制するため に使用される方法を選択します。	
<i>wal_writer_delay</i>	Cluster	Reload	EPAS service account	WAL フラッシュ間の WAL ライタの スリープ時間。	
<i>work_mem</i>	Session	Immediate	User	クエリワークスペースに使用される 最大メモリを設定します。	
<i>xloginsert_locks</i>	Cluster	Restart	EPAS service account	同時 xlog 挿入に使用されるロックの 数を設定します。	
<i>xmlbinary</i>	Session	Immediate	User	バイナリ値を XML でエンコードする 方法を設定します。	
<i>xmloption</i>	Session	Immediate	User	暗黙的な解析およびシリアル化操作の XML データをドキュメントまたはコ ンテンツフラグメントと見なすかど うかを設定します。	
<i>zero_damaged_pages</i>	Session	Immediate	Superuser	過去の破損したページ ヘッダーの処 理を続行します。	

3.1.3 Configuration parameters by functionality

パラメーターの各グループの説明には、次の属性のリストが含まれています。

- **パラメータータイプ。**パラメーターが受け入れることができる値のタイプ。パラメータータイプの値については、

Setting configuration parameters を参照してください。

- **デフォルト値。**値が構成ファイルで明示的に設定されていない場合のデフォルト設定。
- **範囲。**値の許容範囲。
- **効果の最小範囲。**個別に設定できる最小の範囲。通常、個別の設定の最小スコープは次のいずれかです。-クラスター全体、つまり設定はクラスター内のすべてのデータベースとそのセッションで同じです。この属性は、

Summary of configuration parameters の表の **Scope of effect** 列と同じ意味です。

- **値の変更が有効になるとき。**パラメーターの値の変更をアクティブにするために必要な最小侵襲性のアクション。構成ファイルで行われたすべてのパラメーター設定の変更は、データベースサーバを再起動することにより有効にできます。ただし、特定のパラメーターにはデータベースサーバ **restart** が必要です。一部のパラメータ設定の変更は、データベースサーバを停止せずに構成ファイルの **reload** を有効にできます。最後に、他のパラメーター設定の変更は、結果が **immediate** のクライアント側アクションで有効にできます。この属性は、

Summary of configuration parameters の表の **When takes effect** 列と同じ意味です。

- **アクティブ化に必要な認証。**パラメーター設定の変更をアクティブ化するためのユーザー認証のタイプ。データベースサーバの再起動または構成ファイルのリロードが必要な場合、ユーザーは EDB Postgres Advanced Server サービスアカウント（EDB Postgres Advanced Server 互換性インストールモードに応じて **enterprisedb** または **postgres**）である必要があります。この属性は、

Summary of configuration parameters の表の **Authorized user** 列と同じ意味です。

top_performance_related_parameters resource_usage_memory resource_usage_edb_resource_manager
query_tuning query_tuning_planner_method_configuration reports_and_logging_what_to_log auditing_settings
ccd_locale_and_formatting ccd_statement_behaviour ccd_other_defaults audits unisted_options

Top performance-related parameters

これらの構成パラメーターは、パフォーマンスに最も直接影響します。

shared_buffers work_mem maintenance_work_mem wal_buffers checkpoint_segments check-
point_completion_target checkpoint_timeout max_wal_size min_wal_size bgwriter_delay seq_page_cost
random_page_cost effective_cache_size synchronous_commit edb_max_spins_per_delay pg_prewarm_prewarm

shared_buffers

パラメータータイプ： 整数

デフォルト値： 32MB

範囲： 128kB～システムに依存

影響の最小範囲： クラスタ

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

データベースサーバが共有メモリバッファに使用するメモリの量を設定します。通常、デフォルトは 32MB ですが、カーネル設定がサポートしていない場合は、`initdb` で決定されます。この設定は 128kB 以上にする必要があります。(`BLCKSZ` のデフォルト以外の値により最小値が変更されます。) ただし、通常、良好なパフォーマンスを得るには、最小値よりも大幅に高い設定が必要です。

1GB 以上の RAM を備えた専用データベースサーバがある場合、`shared_buffers` の妥当な開始値はシステムのメモリの 25% です。一部のワークロードでは、`shared_buffers` の設定を大きくすると効果的です。ただし、EDB Postgres Advanced Server はオペレーティングシステムのキャッシュにも依存しているため、40 % を超える RAM を `shared_buffers` に割り当てると、より適切に機能する可能性があります。

1GB 未満の RAM を搭載したシステムでは、オペレーティングシステム用の領域を確保するために、より少ない割合の RAM が適切です。これらの状況では、メモリの 15 % がより一般的です。また、Windows、`shared_buffers` の値がそれほど効果的ではありません。設定を比較的低く保ち、代わりにオペレーティングシステムのキャッシュをより多く使用すると、より良い結果が得られる場合があります。Windows システムでの `shared_buffers` の有効範囲は、一般に 64MB から 512MB です。

このパラメーターを増やすと、EDB Postgres Advanced Server はオペレーティングシステムのデフォルト構成で許可されているよりも多くの System V 共有メモリを要求する場合があります。これらのパラメーターを調整する方法については、PostgreSQL コアドキュメントの [Shared Memory and Semaphores](#) を参照してください。

work_mem

パラメータータイプ： 整数

デフォルト値： 1MB

範囲： 64kB～2097151kB

効果の最小範囲： セッションごと

値の変更が有効になるとき： 即時

アクティブ化に必要な承認： セッションユーザー

一時ディスクファイルに書き込む前に使用する内部ソート操作とハッシュテーブルのメモリ量を指定します。

複雑なクエリの場合、いくつかの並べ替えまたはハッシュ操作が並列に実行される場合があります。各操作は、一時ファイルへのデータの書き込みを開始する前に、この値で指定された量のメモリを使用できま

す。また、複数の実行中のセッションがこのような操作を同時に実行する場合があります。したがって、使用される合計メモリは `work_mem` の値の数倍になる場合があります。値を選択するときは、この情報を念頭に置いてください。

ソート操作は、`ORDER BY`、`DISTINCT`、およびマージ結合に使用されます。ハッシュテーブルは、ハッシュ結合、ハッシュベースの集計、および `IN` サブクエリのハッシュベースの処理で使用されます。

妥当な値は、マシンのサイズ、予想される同時接続数（`max_connections` で決定）、およびクエリの複雑さに応じて、通常 4MB～64MB です。

maintenance_work_mem

パラメータータイプ： 整数

デフォルト値： 16MB

範囲： 1024kB～2097151kB

効果の最小範囲： セッションごと

値の変更が有効になるとき： 即時

アクティブ化に必要な承認： セッションユーザー

`VACUUM`、`CREATE INDEX`、`ALTER TABLE ADD FOREIGN KEY` などのメンテナンス操作で使用するメモリの最大量を指定します。デフォルトは 16MB です。データベースセッションは一度に 1 つの操作のみを実行でき、インストールでは通常、多くの操作を同時に実行しないため、この値を `work_mem` よりも大幅に大きく設定しても安全です。設定を大きくすると、バキュームとデータベースダンプのパフォーマンスが向上する場合があります。

`autovacuum` を実行すると `autovacuum_max_workers` 回まで割り当てられるので、デフォルト値を高くしすぎないように注意してください。

経験則として、これをシステムメモリの約 5% に設定しますが、約 512MB 以下に設定します。値を大きくしてもパフォーマンスが向上するとは限りません。

wal_buffers

パラメータータイプ： 整数

デフォルト値： 64kB

範囲： 32kB～システムに依存

影響の最小範囲： クラスタ

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

WAL データの共有メモリで使用するメモリ量。データはトランザクションがコミットされるたびにディスクに書き込まれるため、設定は、1 つの一般的なトランザクションによって生成される WAL データの量のみを保持するのに十分な大きさにする必要があります。

このパラメーターを増やすと、EDB Postgres Advanced Server はオペレーティングシステムのデフォルト構成で許可されているよりも多くの System V 共有メモリを要求する場合があります。これらのパラメーターを調整する方法については、PostgreSQL コアドキュメントの [Shared Memory and Semaphores](#) を参照してください。

この非常に小さな設定でも常に問題が発生するわけではありませんが、状況によっては、余分な `fsync` 呼び出しが発生し、システム全体のスループットが低下する可能性があります。この値を約 1MB に増やすと、この問題に対処できます。非常にビジーなシステムでは、最大約 16MB のさらに高い値が必要になる場合があります。`shared_buffers` と同様に、このパラメーターは EDB Postgres Advanced Server の初期共有メモリ割り当てを増やします。それを増やすと EDB Postgres Advanced Server の起動に失敗する場合は、オペレーティングシステムの制限を増やします。

checkpoint_segments

非推奨。このパラメーターは EDB Postgres Advanced Server ではサポートされていません。

checkpoint_completion_target

パラメータータイプ：浮動小数点

デフォルト値：0.5

範囲：0～1

影響の最小範囲：クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認：EPAS サービスアカウント

チェックポイント完了のターゲットをチェックポイント間の合計時間の割合として指定します。このパラメーターは、システムが次のチェックポイントに向けて作業を開始しているときに、チェックポイントの書き込みを分散します。

デフォルトの 0.5 は、次のチェックポイントの 50 % の準備ができたときにチェックポイントの書き込みを終了することを目的としています。0.9 の値は、次のチェックポイントの 90 % が完了したときにチェックポイントの書き込みを完了することを目的としています。この値は、長時間にわたるチェックポイント書き込みを調整し、パフォーマンスのボトルネックのスパイクを回避します。

checkpoint_timeout

パラメータータイプ：整数

デフォルト値：5 分

範囲：30s～3600s

影響の最小範囲：クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認：EPAS サービスアカウント

自動 WAL チェックポイント間の最大時間（秒単位）。このパラメーターを増やすと、クラッシュからの回復に必要な時間が長くなる可能性があります。

`checkpoint_timeout` を 15 分などの大きな値に増やすと、特に `shared_buffers` に大きな値を使用する場合に、システムの I/O 負荷を軽減できます。

チェックポイントパラメーターにこれらの調整を行うことのマイナス面は、システムが適度な量の追加ディスク領域を使用し、クラッシュが発生した場合に回復に時間がかかることです。ただし、ほとんどのユーザーにとって、これはパフォーマンスを大幅に向上させるための価値があります。

`max_wal_size`

パラメータータイプ：整数

デフォルト値：1GB

範囲：2～2147483647

影響の最小範囲：クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認：EPAS サービスアカウント

`max_wal_size` は、自動 WAL チェックポイント間で到達する WAL の最大サイズを指定します。これはソフトリミットです。WAL サイズは、負荷が高いとき、`archive_command` が失敗したとき、または `wal_keep_segments` 設定が高いときなど、特別な状況下で `max_wal_size` を超える場合があります。

このパラメーターを増やすと、クラッシュからの回復に必要な時間が長くなる可能性があります。このパラメーターは、`postgresql.conf` ファイルまたはサーバーのコマンドラインで設定できます。

`min_wal_size`

パラメータータイプ：整数

デフォルト値：80MB

範囲：2～2147483647

影響の最小範囲：クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認：EPAS サービスアカウント

WAL ディスク使用量が `min_wal_size` で指定された値を下回っている場合、古い WAL ファイルは削除されるのではなく、チェックポイントでリサイクルされます。この機能により、大規模なバッチジョブを実行する場合など、WAL 使用量のスパイクを処理するために十分な WAL スペースが予約されます。このパラメーターは、`postgresql.conf` ファイルまたはサーバーのコマンドラインで設定できます。

bgwriter_delay

パラメータータイプ： 整数

デフォルト値： 200 ミリ秒

範囲： 10ms～10000ms

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

バックグラウンドライタのアクティビティラウンド間の遅延を指定します。各ラウンドで、ライタはいくつかのダーティバッファーに対する書き込みを発行します。（`bgwriter_lru_maxpages` および `bgwriter_lru_multiplier` パラメーターを使用して、ダーティバッファーの数を制御できます。）次に、`bgwriter_delay` ミリ秒間スリープし、繰り返します。

多くのシステムでは、スリープ遅延の有効な解像度は 10 ミリ秒です。`bgwriter_delay` を 10 の倍数ではない値に設定すると、1 つ上の 10 の倍数に設定した場合と同じ結果になる場合があります。

通常、`bgwriter_delay` を調整するときは、デフォルト値から減らします。このパラメーターが増加することはほとんどありません。使用率が非常に低いシステムで消費電力を節約する場合は、値を増やすことができます。

seq_page_cost

パラメータータイプ： 浮動小数点

デフォルト値： 1

範囲： 0～1.79769e+308

効果の最小範囲： セッションごと

値の変更が有効になるとき： 即時

アクティブ化に必要な承認： セッションユーザー

一連のシーケンシャルフェッチの一部であるディスクページフェッチのコストのプランナーの見積もりを設定します。同じ名前のテーブルスペースパラメーターを設定することにより、特定のテーブルスペースのこの値をオーバーライドできます。PostgreSQL コアドキュメントの [ALTER TABLESPACE](#) を参照してください。

デフォルト値は非常に少ないキャッシングを想定しているため、多くの場合、これを減らすことをお勧めします。データベースが物理メモリよりも大幅に大きい場合でも、このパラメーターをデフォルト値の 1 よりも低い値に設定して、その方法でより良いクエリプランが得られるかどうかを確認することをお勧めします。データベースが完全にメモリに収まる場合は、この値をさらに小さくできます（たとえば、0.1）。

random_page_cost

パラメータータイプ：浮動小数点

デフォルト値：4

範囲：0～1.79769e+308

効果の最小範囲：セッションごと

値の変更が有効になるとき：即時

アクティブ化に必要な承認：セッションユーザー

ノンシーケンシャルにフェッチされるディスクページのコストのプランナーの見積もりを設定します。同じ名前の表領域パラメーターを設定することにより、特定の表領域のデフォルトをオーバーライドできます。PostgreSQL コアドキュメントの [ALTER TABLESPACE](#) を参照してください。

seq_page_cost に対してこの値を減らすと、システムはインデックススキャンを優先します。それを上げると、インデックススキャンが比較的高価に見えます。両方の値を一緒に上げたり下げたりして、cpu_tuple_cost および cpu_index_tuple_cost パラメーターで説明される CPU コストに対するディスク I/O コストの重要性を変更できます。

デフォルト値は非常に少ないキャッシングを想定しているため、多くの場合、これを減らすことをお勧めします。データベースが物理メモリよりも大幅に大きい場合でも、このパラメーターを2（つまり、デフォルトよりも低い値）に設定して、その方法でより良いクエリプランが得られるかどうかを確認することをお勧めします。データベースが完全にメモリに収まる場合は、この値をさらに小さくできます（たとえば、0.1）。

システムで許可されていますが、random_page_cost を seq_page_cost 未満に設定しないでください。ただし、データベースがほとんどまたは完全にメモリに収まる場合、それらを等しいか非常に等しく設定することは理にかなっています。また、キャッシュの多いデータベースでは、既に RAM にあるページをフェッチするコストが通常よりもはるかに少ないため、CPU パラメーターに対して両方の値を小さくします。

effective_cache_size

パラメータータイプ：整数

デフォルト値：128MB

範囲：8kB～17179869176kB

効果の最小範囲：セッションごと

値の変更が有効になるとき：即時

アクティブ化に必要な承認：セッションユーザー

1つのクエリで使用できるディスクキャッシュの有効なサイズに関するプランナーの前提を設定します。この仮定は、インデックスを使用するコストの見積もりに考慮されます。値が高いほど、インデックススキャンが使用される可能性が高くなります。値が小さいほど、シーケンシャルスキャンが使用される可能性が高くなります。

このパラメーターを設定するときは、EDB Postgres Advanced Server の共有バッファと、EDB Postgres Advanced Server データファイルに使用されるカーネルのディスクキャッシュの両方を考慮してください。また、使用可能なスペースを共有する必要があるため、さまざまなテーブルに対する同時クエリの予想数を検討してください。

このパラメーターは、EDB Postgres Advanced Server によって割り当てられた共有メモリのサイズには影響せず、カーネルディスクキャッシュは予約されません。見積もりのみにご利用ください。

このパラメーターの設定が低すぎると、プランナーはインデックスを使用すると役立つ場合でも、使用しないことを決定する場合があります。`effective_cache_size` を物理メモリの 50% に設定するのは、通常の保守的な設定です。より積極的な設定は、物理メモリの約 75% です。

synchronous_commit

パラメータータイプ: ブール

デフォルト値: true

範囲: {true | false}

効果の最小範囲: セッションごと

値の変更が有効になるとき: 即時

アクティブ化に必要な承認: セッションユーザー

コマンドが `success` 指示をクライアントに返す前に、トランザクションのコミットが WAL レコードがディスクに書き込まれるのを待機するかどうかを指定します。セーフ設定はオンです。オフにすると、クライアントに成功が報告されてから、トランザクションがサーバークラッシュに対して安全であることが実際に保証されるまでに遅延が発生する場合があります。(最大遅延は `wal_writer_delay` の 3 倍です。)

`fsync` とは異なり、このパラメーターをオフに設定しても、データベースの一貫性が失われるリスクはありません。オペレーティングシステムまたはデータベースがクラッシュすると、最近の「おそらくコミットされた」トランザクションが失われる場合があります。ただし、データベースの状態は、これらのトランザクションが正常に中止された場合と同じです。

したがって、`synchronous_commit` をオフにすることは、トランザクションの耐久性に関する正確な確実性よりもパフォーマンスが重要な場合に役立ちます。詳細については、PostgreSQL コードキュメントの [Asynchronous Commit](#) を参照してください。

このパラメーターはいつでも変更できます。1 つのトランザクションの動作は、コミット時に有効な設定によって決まります。したがって、一部のトランザクションを同期的にコミットし、他のトランザクションを非同期的にコミットすることが可能であり、便利です。たとえば、デフォルトが逆の場合に単一のマルチステートメントトランザクションを非同期にコミットするには、トランザクションで `SET LOCAL synchronous_commit TO OFF` を発行します。

edb_max_spins_per_delay

パラメータータイプ： 整数

デフォルト値： 1000

範囲： 10～1000

影響の最小範囲： クラスターごと

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

スピンロックの待機中にセッションがスピンする最大回数を指定します。ロックが取得されない場合、セッションはスリープします。

このパラメーターは、Non-Uniform Memory Access (NUMA) アーキテクチャを使用するシステムで役立ちます。

pg_prewarm.autoprewarm

パラメータータイプ： ブール

デフォルト値： true

範囲： {true | false}

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

データベースサーバが、シャットダウン前に共有バッファをディスクにダンプするバックグラウンドワーカプロセスである `autoprewarm` を実行するかどうかを制御します。次に、サーバーが次に起動されたときに共有バッファを**事前にウォーム**します。つまり、ブロックをディスクからバッファプールにロードします。

このパラメーターの利点は、シャットダウン前にディスクにダンプされたデータをロードすることにより、サーバー再起動後のウォームアップ時間が短縮されることです。

`pg_prewarm.autoprewarm` を on に設定して、`autoprewarm` ワーカーを有効にします。`autoprewarm` を無効にするには off に設定します。

`autoprewarm` を使用する前に、次の例に示すように、`postgresql.conf` ファイルの `shared_preload_libraries` 構成パラメーターにリストされているライブラリに `$libdir/pg_prewarm` を追加する必要があります。

```
shared_preload_libraries = $libdir/dbms_pipe,$libdir/edb_gen,$libdir/dbms_aq,$libdir/  
→pg_prewarm
```

`shared_preload_libraries` パラメータを変更した後、データベースサーバを再起動します。再起動後、サーバーが一貫した状態に達した直後に `autoprewarm` バックグラウンドワーカーが起動します。

`autoprewarm` プロセスは、バッファープールに空きバッファ領域がなくなるまで、以前に `$PGDATA/autoprewarm.blocks` に記録されたブロックのロードを開始します。この方法では、回復プロセスまたはクエリを実行するクライアントによって読み込まれた新しいブロックは置き換えられません。

`autoprewarm` プロセスは、ディスクからのバッファのロードを終了すると、`pg_prewarm.autoprewarm_interval` パラメーターで指定された間隔で定期的に共有バッファをディスクにダンプします。次のサーバー再起動時に、`autoprewarm` プロセスは、最後にディスクにダンプされたブロックで共有バッファを事前にウォームします。

`autoprewarm` バックグラウンドワーカーについては、

[*pg_prewarm.autoprewarm_interval*](#) を参照してください。

`pg_prewarm.autoprewarm_interval`

パラメータータイプ： 整数

デフォルト値： 300s

範囲： 0s～2147483s

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

`autoprewarm` バックグラウンドワーカーが共有バッファをディスクにダンプするまでの最小秒数。0 に設定すると、共有バッファは定期的にダンプされません。サーバーをシャットダウンした場合にのみダンプされます。

`autoprewarm` バックグラウンドワーカーについては、

[*pg_prewarm.autoprewarm*](#) を参照してください。

Resource usage/memory

これらの構成パラメーターは、メモリに関連するリソースの使用を制御します。

`edb_dynatune`

パラメータータイプ： 整数

デフォルト値： 0

範囲： 0～100

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

ホストマシンの合計使用可能なリソースとホストマシンの用途に基づいて、データベースサーバが使用するホストシステムのリソースを決定します。

EDB Postgres Advanced Server を最初にインストールするときは、インストールされたホストマシンの用途、つまり開発マシン、混合使用マシン、または専用サーバーに応じて `edb_dynatune` を設定します。ほとんどの場合、データベース管理者はパフォーマンスを向上させるために `postgresql.conf` ファイルのさまざまな構成パラメーターを調整する必要はありません。

`edb_dynatune` パラメーターは 0 から 100 の任意の整数値に設定できます。値 0 を指定すると動的チューニング機能がオフになり、データベースサーバーリソースの使用が `postgresql.conf` ファイルの他の構成パラメーターによって制御されます。

1 から 33 などの低いゼロ以外の値は、ホストマシンのリソースを最小限に抑えます。これらの値は、他の多くのアプリケーションが使用されている開発マシンに適しています。

34 から 66 の範囲の値は、適度な量のリソースをデータベースサーバーに割り当てます。この設定は、EDB Postgres Advanced Server と同じマシンで固定数の他のアプリケーションが実行されている専用のアプリケーションサーバーに使用される場合があります。

67 から 100 の最大値は、サーバーのリソースの大部分をデータベースサーバーに割り当てます。EDB Postgres Advanced Server の実行専用のホストマシンには、この範囲の設定を使用します。

`edb_dynatune` の値を選択した後、`postgresql.conf` ファイルの他の構成パラメーターを調整することにより、データベースサーバーのパフォーマンスをさらに調整できます。調整された設定は、`edb_dynatune` によって選択された対応する値をオーバーライドします。パラメーターの値を変更するには、構成パラメーターのコメントを外し、目的の値を指定して、データベースサーバを再起動します。

`edb_dynatune_profile`

パラメータータイプ: Enum

デフォルト値: `oltp`

範囲: {`oltp` | `reporting` | `mixed`}

影響の最小範囲: クラスター

値の変更が有効になるとき:

アクティブ化に必要な承認: EPAS サービスアカウント

データベース サーバで予想されるワークロードプロファイルに基づいて、チューニングの側面を制御します。

可能な値は次のとおりです。

- `oltp`。データベースサーバが重いオンライントランザクション処理ワークロードを処理している場合にお勧めします。
- `reporting`。大量のデータのレポートに使用されるデータベースサーバーに推奨されます。
- `mixed`。トランザクション処理とデータレポートを組み合わせたサーバーに推奨されます。

Resource usage/EDB Resource Manager

これらの構成パラメーターは、EDB Resource Manager を介してリソースの使用を制御します。

edb_max_resource_groups

パラメータータイプ： 整数

デフォルト値： 16

範囲： 0～65536

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

EDB Resource Manager が同時に使用できるリソースグループの最大数を制御します。
edb_max_resource_groups で指定した値よりも多くのリソースグループを作成できます。ただし、これらのグループのプロセスによってアクティブに使用されているリソースグループの数はこの値を超えることはできません。

このパラメーターは、維持することが予想されるグループの数を処理できるように設定します。

edb_resource_group

パラメータータイプ： 文字列

デフォルト値： なし

範囲： 該当なし

効果の最小範囲： セッションごと

値の変更が有効になるとき： 即時

アクティブ化に必要な承認： セッションユーザー

グループのリソースタイプ設定に従って現在のセッションで制御する EDB Resource Manager のリソースグループの名前。

このパラメーターを設定しない場合、現在のセッションは EDB Resource Manager を使用しません。

Query tuning

これらの構成パラメーターは、オプティマイザーのヒントに使用されます。

enable_hints

パラメータータイプ：ブール

デフォルト値：true

範囲：{true | false}

効果の最小範囲：セッションごと

値の変更が有効になるとき：即時

アクティブ化に必要な承認：セッションユーザー

SQL コマンドに埋め込まれたオプティマイザーヒントを有効にします。このパラメーターがオフの場合、オプティマイザーのヒントは無視されます。

Query tuning/planner method configuration

これらの構成パラメーターは、プランナーメソッドの構成に使用されます。

edb_enable_pruning

パラメータータイプ：ブール

デフォルト値：false

範囲：{true | false}

効果の最小範囲：セッションごと

値の変更が有効になるとき：即時

アクティブ化に必要な承認：セッションユーザー

クエリプランナーがパーティションテーブルを**早期にブルーニング**できるようにします。早期ブルーニングとは、クエリプランナーがクエリプランを生成する前に、クエリで検索されないパーティションを「ブルーニング」（つまり、無視）できることを意味します。この設定は、検索されないパーティションのクエリプランが生成されないようにするため、パフォーマンスの向上に役立ちます。

逆に、**レイトブルーニング**は、クエリプランナーが各パーティションのクエリプランを生成した後にパーティションをブルーニングすることを意味します。`constraint_exclusion` 構成パラメーターは、レイトブルーニングを制御します。

早期にブルーニングする機能は、WHERE 句のクエリの性質によって異なります。早期ブルーニングは、WHERE `column = literal`、たとえば、WHERE `deptno = 10` のような制約を持つ単純なクエリでのみ使用できます。

WHERE column = expression、たとえば、WHERE deptno = 10 + 5 など、より複雑なクエリには早期プルーニングを使用しないでください。

このパラメーターは、バージョン 15 以降では非推奨です。代わりに enable_partition_pruning を使用してください。

Reporting and logging/what to log

これらの構成パラメーターは、レポートとログを制御します。

trace_hints

パラメータータイプ：ブール

デフォルト値：false

範囲：{true | false}

効果の最小範囲：セッションごと

値の変更が有効になるとき：即時

アクティブ化に必要な承認：セッションユーザー

オプティマイザーのヒント機能とともに使用して、プランナーがヒントを使用したかどうかに関するより詳細な情報を提供します。client_min_messages および trace_hints 構成パラメーターを次のように設定します。

```
SET client_min_messages TO info;
SET trace_hints TO true;
```

この例は、NO_INDEX ヒントを指定した SELECT コマンドが、これらの構成パラメーターを設定したときに生成される追加情報を表示する方法を示しています。

```
EXPLAIN SELECT /*+ NO_INDEX(accounts accounts_pkey) */ * FROM accounts
WHERE aid = 100;

INFO: [HINTS] Index Scan of [accounts].[accounts_pkey] rejected because
of NO_INDEX hint.

INFO: [HINTS] Bitmap Heap Scan of [accounts].[accounts_pkey] rejected
because of NO_INDEX hint.
__OUTPUT__

          QUERY PLAN

-----
Seq Scan on accounts (cost=0.00..14461.10 rows=1 width=97)
```

(次のページに続く)

(前のページからの続き)

```
Filter: (aid = 100)
(2 rows)
```

edb_log_every_bulk_value

パラメータータイプ: ブール

デフォルト値: false

範囲: {true | false}

効果の最小範囲: セッションごと

値の変更が有効になるとき: 即時

アクティブ化に必要な承認: スーパーユーザー

一括処理は、結果のステートメントを EDB Postgres Advanced Server ログファイルと EDB Audit ログファイルの両方に記録します。ただし、一括処理ですべてのステートメントをログに記録するにはコストがかかります。edb_log_every_bulk_value 構成パラメーターを使用して、ログに記録される一括処理ステートメントを制御できます。

このパラメーターを true に設定すると、一括処理のすべてのステートメントがログに記録されます。バルク実行中に edb_log_every_bulk_value が false に設定されている場合、ログメッセージは処理された行数とともにバルク処理ごとに記録されます。期間は、一括処理ごとに 1 回発行されます。

Auditing settings

これらの構成パラメーターは、EDB Postgres Advanced Server データベース監査機能で使します。

```
edb_audit edb_audit_directory edb_audit_filename edb_audit_rotation_size edb_audit_rotation_seconds
edb_audit_connect edb_audit_disconnect edb_audit_statement edb_audit_audit_value
```

edb_audit

パラメータータイプ: Enum

デフォルト値: none

範囲: {none | csv | xml}

影響の最小範囲: クラスタ

値の変更が有効になるとき:

アクティブ化に必要な承認: EPAS サービスアカウント

データベース監査を有効または無効にします。xml または csv の値はデータベース監査を有効にします。これらの値により、監査情報をキャプチャするファイル形式が決まります。none は、データベース監査を無効にします。

edb_audit_directory

パラメータータイプ： 文字列

デフォルト値： edb_audit

範囲： 該当なし

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

監査ログファイルが作成されるディレクトリを指定します。 EDB Postgres Advanced Server の data ディレクトリからの絶対パスまたは相対パスを指定します。

edb_audit_filename

パラメータータイプ： 文字列

デフォルト値： audit-%Y%m%d_%H%M%S

範囲： 該当なし

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

監査情報を格納する監査ファイルのファイル名を指定します。デフォルトのファイル名は audit-%Y%m%d_%H%M%S です。 %Y や %m などのエスケープシーケンスは、システムの日付と時刻の適切な現在値に置き換えられます。

edb_audit_rotation_day

パラメータータイプ： 文字列

デフォルト値： every

範囲： {none | every | sun | mon | tue | wed | thu | fri | sat} ...

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

監査ファイルをローテーションする曜日を指定します。有効な値は、sun、mon、tue、wed、thu、fri、sat、every、および none です。回転を無効にするには、値を none に設定します。ファイルを毎日ローテーションするには、edb_audit_rotation_day の値を every に設定します。特定の曜日にファイルをローテーションするには、値を目的の曜日に設定します。

edb_audit_rotation_size

パラメータータイプ： 整数

デフォルト値： 0MB

範囲： 0MB～5000MB

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

ファイルのローテーションが発生するときのファイルサイズのしきい値をメガバイトで指定します。パラメーターがコメントアウトされているか、0 に設定されている場合、サイズに基づいたファイルの回転は発生しません。

edb_audit_rotation_seconds

パラメータータイプ： 整数

デフォルト値： 0s

範囲： 0s～2147483647s

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

新しいログファイルを作成するときのローテーション時間を秒単位で指定します。この機能を無効にするには、このパラメーターを 0 のままにします。

edb_audit_connect

パラメータータイプ： Enum

デフォルト値： failed

範囲： {none | failed | all}

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

ユーザーによるデータベース接続試行の監査を有効にします。すべての接続試行の監査を無効にするには、edb_audit_connect を none に設定します。失敗した接続試行をすべて監査するには、値を failed に設定します。すべての接続試行を監査するには、値を all に設定します。

edb_audit_disconnect

パラメータータイプ: Enum

デフォルト値: none

範囲: {none | all}

影響の最小範囲: クラスター

値の変更が有効になるとき:

アクティブ化に必要な承認: EPAS サービスアカウント

接続しているユーザーによるデータベース切断の監査を有効にします。切断の監査を有効にするには、値を all に設定します。無効にするには、値を none に設定します。

edb_audit_statement

パラメータータイプ: 文字列

デフォルト値: ddl, error

範囲: {none | ddl | dml | insert | update | delete | truncate | select | error | create | drop | alter | grant | revoke | rollback | set | all | { select | update | delete | insert }@groupname} ...

影響の最小範囲: クラスター

値の変更が有効になるとき:

アクティブ化に必要な承認: EPAS サービスアカウント

さまざまなカテゴリの SQL ステートメントと、特定の SQL コマンドに関連するステートメントの監査を指定します。

- エラーをログに記録するには、パラメーター値を **error** に設定します。
- CREATE TABLE や ALTER TABLE などのすべての DDL ステートメントを監査するには、パラメーター値を **ddl** に設定します。
- 特定の種類の DDL ステートメントを監査するには、パラメーター値にそれらの特定の SQL コマンド (create、drop、または alter) を含めることができます。また、コマンドの後に create table、create view、drop role などのオブジェクトタイプを指定できます。
- INSERT、UPDATE、DELETE、TRUNCATE などのすべての変更ステートメントを監査するには、edb_audit_statement を **dml** に設定します。
- 特定の種類の DML ステートメントを監査するには、パラメーター値に特定の SQL コマンド insert、update、delete、または truncate を含めることができます。パラメーター値 select、grant、revoke、または rollback を含めて、これらの SQL コマンドに関するステートメントを監査します。
- SET ステートメントを監査するには、パラメーター値 **SET** を含めます。
- すべてのステートメントを監査するには、値を **all** に設定します。

- この機能を無効にするには、値を `none` に設定します。

オブジェクトごとのレベルの監査は、特定のテーブルの (@) を含むおよび (-) グループを除外する `SELECT`、`UPDATE`、`DELETE`、および `INSERT` ステートメントなどのオブジェクト特権によって許可される操作を監査します。特定のタイプのオブジェクトを監査するには、監査するオブジェクトグループの名前を指定します。`edb_audit_statement` パラメーターには、(@) インクルードシンボルおよび (-) エクスクルードシンボルを使用して、グループ名に関連付けられた特定の SQL コマンド (`select`、`update`、`delete`、または `insert`) を含めることができます。

`edb_audit_tag`

パラメータータイプ：文字列

デフォルト値：なし

効果の最小範囲：セッション

値の変更が有効になるとき：即時

アクティブ化に必要な承認：

`edb_audit` パラメーターが `csv` または `xml` に設定されている場合に監査ログに含める文字列値を指定します。

`edb_audit_destination`

パラメータータイプ：Enum

デフォルト値：file

範囲：{file | syslog}

影響の最小範囲：クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認：EPAS サービスアカウント

`edb_audit_directory` パラメーターで指定されたディレクトリに監査ログ情報を記録するか、`syslog` プロセスによって管理されるディレクトリとファイルに記録するかを指定します。`edb_audit_directory` で指定されたディレクトリを使用するには、`file` に設定します。

`/etc/syslog.conf` ファイルで構成された `syslog` プロセスとその場所を使用するには、`syslog` に設定します。`syslog` 設定は、Linux ホストで実行されている EDB Postgres Advanced Server でのみ有効です。Windows システムではサポートされていません。

注釈: 最近の Linux バージョンでは、`syslog` は `rsyslog` に置き換えられました。構成ファイルは `/etc/rsyslog.conf` にあります。

edb_log_every_bulk_value

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

Client connection defaults/locale and formatting

これらの構成パラメーターは、ロケールとフォーマットに影響します。

icu_short_form

パラメータータイプ： 文字列

デフォルト値： なし

範囲： 該当なし

影響の最小範囲： データベース

値の変更が有効になる時期： 該当なし

アクティブ化に必要な承認： 該当なし

データベースまたは EDB Postgres Advanced Server インスタンスに割り当てられたデフォルトの ICU の短い形式の名前が含まれています。ICU 短縮形と Unicode 照合アルゴリズムに関する一般的な情報については、

[Unicode collation algorithm](#) を参照してください。

次のいずれかの場合に、この構成パラメーターを設定します。

- ICU_SHORT_FORM パラメーターを指定して CREATE DATABASE コマンドを使用します。この場合、指定された短縮形の名前を設定します。このデータベースに接続すると、icu_short_form 構成パラメーターに表示されます。
- --icu_short_form オプションで使用する initdb コマンドで EDB Postgres Advanced Server インスタンスを作成します。この場合、指定された短縮形の名前を設定します。その EDB Postgres Advanced Server インスタンスのデータベースに接続すると、icu_short_form 構成パラメーターに表示されます。データベースは、独自の ICU_SHORT_FORM パラメーターと別の短い形式でオーバーライドしません。

このパラメーターは、一度設定すると変更できません。

Client connection defaults/statement behavior

これらの構成パラメーターは、ステートメントの動作に影響します。

default_heap_fillfactor

パラメータータイプ： 整数

デフォルト値： 100

範囲： 10～100

効果の最小範囲： セッションごと

値の変更が有効になるとき： 即時

アクティブ化に必要な承認： セッションユーザー

CREATE TABLE コマンドから FILLFACTOR ストレージパラメーターが省略された場合、テーブルのフィルファクターを設定します。

テーブルのフィルファクターは 10～100 のパーセンテージで、100 は完全なパッキングです。より小さいフィルファクターを指定すると、INSERT 操作は指定されたパーセンテージのみにテーブルページをパックします。各ページの残りの領域は、そのページの行を更新するために予約されています。このアプローチにより、UPDATE は、行の更新されたコピーを元のページと同じページに配置する機会を提供します。これは、別のページに配置するよりも効率的です。

エントリが更新されないテーブルの場合、完全なパッキングが最適です。頻繁に更新されるテーブルでは、より小さい FILL FACTOR を使用します。

edb_data_redaction

パラメータータイプ： ブール

デフォルト値： true

範囲： {true | false}

効果の最小範囲： セッションごと

値の変更が有効になるとき： 即時

アクティブ化に必要な承認： セッションユーザー

データリダクションは、表示される情報を変更することにより、特定の機密データの特定のユーザーへの公開を制限するポリシーのサポートです。

TRUE に設定すると、データリダクションはスーパーユーザーとテーブル所有者を除くすべてのユーザーに適用されます。

- スーパーユーザーとテーブル所有者はデータ編集をバイパスします。
- 他のすべてのユーザーには、リダクションポリシーが適用され、再フォーマットされたデータが表示されます。

FALSE に設定すると、次のことが発生します。

- スーパーユーザーとテーブル所有者は引き続きデータ編集をバイパスします。
- 他のすべてのユーザーはエラーになります。

データリダクションについては、EDB Postgres Advanced Server Security Features を参照してください。

Client connection defaults/other defaults

これらのパラメーターは、その他のクライアント接続のデフォルトを設定します。

oracle_home

パラメータータイプ：文字列

デフォルト値：なし

範囲：該当なし

影響の最小範囲：クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認：EPAS サービスアカウント

Oracle サーバーへのリンクを作成する前に、EDB Postgres Advanced Server を正しい Oracle ホームディレクトリに指示する必要があります。Linux の場合は LD_LIBRARY_PATH、Windows の場合は PATH を Oracle クライアントのインストールディレクトリの lib ディレクトリに設定します。

または、postgresql.conf ファイルで oracle_home 構成パラメーターの値を設定できます。oracle_home 構成パラメーターで指定された値は、Linux の LD_LIBRARY_PATH 環境変数および Windows の PATH 環境変数をオーバーライドします。

注釈: oracle_home 構成パラメーターは、Oracle クライアント、つまり OCI ライブラリへの正しいパスを提供する必要があります。

postgresql.conf ファイルで oracle_home 構成パラメーターを設定するには、次の行を追加します。

```
oracle_home = <lib_directory>
```

<lib_directory>は、Linux の libclntsh.so および Windows の oci.dll を含む Oracle クライアントのインストールディレクトリへの oracle_home パスの名前です。

oracle_home 構成パラメーターを設定した後、サーバーを再起動して変更を有効にする必要があります。サーバーを再起動します。

- Linux では、systemctl コマンドまたは pg_ctl サービスを使用
- Windows の場合、Windows サービスコンソールから

odbc_lib_path

パラメータータイプ：文字列

デフォルト値：なし

範囲：該当なし

影響の最小範囲：クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認：EPAS サービスアカウント

ODBC ドライバーマネージャーを使用していて、標準以外の場所にインストールする場合は、`postgresql.conf` ファイルで `odbc_lib_path` 構成パラメーターを設定して場所を指定します。

```
odbc_lib_path = 'complete_path_to_libodbc.so'
```

構成ファイルには、ドライバーマネージャー共有ライブラリへの完全なパス名を含める必要があります。これは通常、`libodbc.so` です。

Compatibility options

これらの構成パラメーターは、さまざまなデータベース互換性機能を制御します。

edb_redwood_date

パラメータータイプ：ブール

デフォルト値：false

範囲：{true | false}

効果の最小範囲：セッションごと

値の変更が有効になるとき：即時

アクティブ化に必要な承認：セッションユーザー

コマンドに列のデータ型として `DATE` が表示され、テーブル定義がデータベースに保存されている場合、`DATE` を `TIMESTAMP` に変換します。時間コンポーネントは、日付とともに列に格納されます。

`edb_redwood_date` が `FALSE` に設定されている場合、`CREATE TABLE` または `ALTER TABLE` コマンドの列のデータ型はネイティブの PostgreSQL `DATE` データ型として残り、データベースに保存されます。PostgreSQL の `DATE` データ型は、列に時間コンポーネントなしで日付のみを格納します。

`edb_redwood_date` の設定に関係なく、`DATE` が他のコンテキストでデータ型として表示される場合、常に内部で `TIMESTAMP` に変換されます。したがって、時間コンポーネントが存在する場合はそれを処理できます。これらのコンテキストの例は次のとおりです。

- SPL 宣言セクションの変数のデータ型
- SPL プロシージャまたは SPL ファンクションの仮パラメーターのデータ型

- SPL ファンクションの戻り値

edb_redwood_greatest_least

パラメータータイプ: ブール

デフォルト値: true

範囲: {true | false}

効果の最小範囲: セッションごと

値の変更が有効になるとき: 即時

アクティブ化に必要な承認: セッションユーザー

GREATEST ファンクションは、パラメーターのリストから最大の値を持つパラメーターを返します。LEAST ファンクションは、パラメーターのリストから最小値を持つパラメーターを返します。

edb_redwood_greatest_least が TRUE に設定されている場合、少なくとも 1 つのパラメーターが null の場合、GREATEST および LEAST 関数は null を返します。

```
SET edb_redwood_greatest_least TO on;
```

```
SELECT GREATEST(1, 2, NULL, 3);
```

```
__OUTPUT__
```

```
greatest
```

```
- - - - -
```

```
(1 row)
```

edb_redwood_greatest_least が FALSE に設定されている場合、すべてのパラメーターが null の場合を除き、null パラメーターは無視されます。その場合、関数は null を返します。

```
SET edb_redwood_greatest_least TO off;
```

```
SELECT GREATEST(1, 2, NULL, 3);
```

```
__OUTPUT__
```

```
greatest
```

```
- - - - -
```

```
3
```

```
(1 row)
```

```
SELECT GREATEST(NULL, NULL, NULL);
```

```
__OUTPUT__
```

```
greatest
```

```
- - - - -
```

(次のページに続く)

(前のページからの続き)

(1 row)

edb_redwood_raw_names

パラメータータイプ: ブール

デフォルト値: false

範囲: {true | false}

効果の最小範囲: セッションごと

値の変更が有効になるとき: 即時

アクティブ化に必要な承認: セッションユーザー

edb_redwood_raw_names が FALSE に設定されている場合、Redwood カタログ（つまり、ALL_、DBA_、または USER_ のプレフィックスが付いたシステムカタログ）から見ると、テーブル名、列名、トリガー名、プログラム名、ユーザー名などのデータベースオブジェクト名が大文字で表示されます。また、引用符は、作成された名前を引用符で囲みます。

edb_redwood_raw_names が TRUE に設定されている場合、データベースオブジェクト名は、Redwood カタログから表示したときに PostgreSQL システムカタログに保存されているとおりに表示されます。引用符なしで作成された名前は、PostgreSQL で想定どおりに小文字で表示されます。引用符で囲まれて作成された名前は、作成されたとおりに表示されますが、引用符はありません。

たとえば、次のユーザー名が作成され、そのユーザーでセッションが開始されます。

```
CREATE USER reduser IDENTIFIED BY password;
edb=# \c - reduser
Password for user reduser:
You are now connected to database "edb" as user "reduser".
```

reduser としてデータベースに接続すると、次のテーブルが作成されます。

```
CREATE TABLE all_lower (col INTEGER);
CREATE TABLE ALL_UPPER (COL INTEGER);
CREATE TABLE "Mixed_Case" ("Col" INTEGER);
```

edb_redwood_raw_names をデフォルト値 FALSE に設定して、Redwood カタログ USER_TABLES から見ると、名前が大文字で表示されます。例外は Mixed_Case の名前です。これは、作成済みとして表示され、引用符で囲まれています。

```
edb=> SELECT * FROM USER_TABLES;
__OUTPUT__
 schema_name | table_name | tablespace_name | status | temporary
-----+-----+-----+-----+-----
```

(次のページに続く)

(前のページからの続き)

REDUSER	ALL_LOWER		VALID	N
REDUSER	ALL_UPPER		VALID	N
REDUSER	"Mixed_Case"		VALID	N

(3 rows)

edb_redwood_raw_names を TRUE に設定して表示すると、名前は Mixed_Case の名前を除き小文字で表示されます。作成された名前は引用符なしで表示されます。

```
edb=> SET edb_redwood_raw_names TO true;
SET
edb=> SELECT * FROM USER_TABLES;
__OUTPUT__
 schema_name | table_name | tablespace_name | status | temporary
-----+-----+-----+-----+-----
reduser      | all_lower  |                  | VALID  | N
reduser      | all_upper  |                  | VALID  | N
reduser      | Mixed_Case |                  | VALID  | N
(3 rows)
```

これらの名前は、PostgreSQL pg_tables カタログから表示した場合に大文字と小文字が一致するようになりました。

```
edb=> SELECT schemaname, tablename, tableowner FROM pg_tables WHERE
tableowner = reduser;
__OUTPUT__
 schemaname | tablename | tableowner
-----+-----+-----
reduser     | all_lower | reduser
reduser     | all_upper | reduser
reduser     | Mixed_Case | reduser
(3 rows)
```

edb_redwood_strings

パラメータータイプ：ブール

デフォルト値：false

範囲：{true | false}

効果の最小範囲：セッションごと

値の変更が有効になるとき：即時

アクティブ化に必要な承認：セッションユーザー

edb_redwood_strings パラメーターが TRUE に設定されている場合、文字列がヌル変数またはヌル列と連結されると、結果は元の文字列になります。edb_redwood_strings が FALSE に設定されている場合、ネ

イティブの PostgreSQL の動作は維持されます。これは、文字列と null 変数または null 結果を返す null 列を連結します。

この例は違いを示しています。サンプルアプリケーションには、従業員のテーブルが含まれています。このテーブルには、ほとんどの従業員にとって null である comm という名前の列があります。次のクエリは、edb_redwood_string を FALSE に設定して実行されます。null 列と空でない文字列を連結すると最終結果が null になるため、コミッションのある従業員のみがクエリ結果に表示されます。他のすべての従業員の出力行は null です。

```
SET edb_redwood_strings TO off;

SELECT RPAD(ename,10) ||      || TO_CHAR(sal,99,999.99) ||      ||
TO_CHAR(comm,99,999.99) "EMPLOYEE COMPENSATION" FROM emp;
__OUTPUT__
      EMPLOYEE COMPENSATION
-----

ALLEN      1,600.00      300.00
WARD       1,250.00      500.00

MARTIN     1,250.00     1,400.00

TURNER     1,500.00          .00

(14 rows)
```

以下は、edb_redwood_strings が TRUE に設定されている場合に実行される同じクエリです。ここで、null 列の値は空の文字列として扱われます。空の文字列と空でない文字列を連結すると、空でない文字列が生成されます。

```
SET edb_redwood_strings TO on;

SELECT RPAD(ename,10) ||      || TO_CHAR(sal,99,999.99) ||      ||
TO_CHAR(comm,99,999.99) "EMPLOYEE COMPENSATION" FROM emp;
__OUTPUT__
      EMPLOYEE COMPENSATION
-----

SMITH 800.00
ALLEN 1,600.00 300.00
WARD 1,250.00 500.00
JONES 2,975.00
MARTIN 1,250.00 1,400.00
BLAKE 2,850.00
CLARK 2,450.00
SCOTT 3,000.00
```

(次のページに続く)

(前のページからの続き)

```

KING 5,000.00
TURNER 1,500.00 .00
ADAMS 1,100.00
JAMES 950.00
FORD 3,000.00
MILLER 1,300.00
(14 rows)

```

edb_stmt_level_tx

パラメータータイプ：ブール

デフォルト値：false

範囲：{true | false}

効果の最小範囲：セッションごと

値の変更が有効になるとき：即時

アクティブ化に必要な承認：セッションユーザー

ステートメントレベルのトランザクション分離という用語は、SQL コマンドで実行時エラーが発生し、その単一のコマンドによって発生したデータベース上のすべての更新がロールバックされる動作を説明します。たとえば、単一の UPDATE コマンドで 5 つの行が正常に更新されたが、6 番目の行を更新しようとすると例外が発生した場合、この UPDATE コマンドによる 6 つすべての行への更新はロールバックされます。まだコミットまたはロールバックされていない以前の SQL コマンドの効果は、COMMIT または ROLLBACK コマンドが実行されるまで保留されます。

EDB Postgres Advanced Server では、SQL コマンドの実行中に例外が発生すると、トランザクションの開始以降のデータベースでのすべての更新がロールバックされます。さらに、トランザクションは中止された状態になり、別のトランザクションを開始する前に COMMIT または ROLLBACK コマンドを発行する必要があります。

edb_stmt_level_tx が TRUE に設定されている場合、例外はコミットされていないデータベースの更新をロールバックしません。edb_stmt_level_tx が FALSE に設定されている場合、例外により、コミットされていないデータベース更新がロールバックされます。

注釈： TRUE を `edb\_stmt\_level\_tx` に設定すると、パフォーマンスに悪影響を与える可能性があるため、必要な場合にのみ使用します。

PSQL で実行されるこの例は、edb_stmt_level_tx が FALSE の場合、2 番目の INSERT コマンドを中止すると最初の INSERT コマンドもロールバックすることを示しています。PSQL では、コマンド `\set AUTOCOMMIT off` を発行する必要があります。それ以外の場合、すべてのステートメントは自動的にコミットされますが、これは edb_stmt_level_tx の効果を示しません。

```

\set AUTOCOMMIT off
SET edb_stmt_level_tx TO off;

INSERT INTO emp (empno,ename,deptno) VALUES (9001, JONES, 40);
INSERT INTO emp (empno,ename,deptno) VALUES (9002, JONES, 00);
ERROR: insert or update on table "emp" violates foreign key constraint
"emp_ref_dept_fk"
DETAIL: Key (deptno)=(0) is not present in table "dept".

COMMIT;
SELECT empno, ename, deptno FROM emp WHERE empno > 9000;
__OUTPUT__
empno | ename | deptno
- - - - - + - - - - - + - - - - -
(0 rows)

```

この例では、`edb_stmt_level_tx` が `TRUE` に設定されているため、2 番目の `INSERT` コマンドでエラーが発生した後、最初の `INSERT` コマンドはロールバックされませんでした。この時点で、最初の `INSERT` コマンドをコミットまたはロールバックできます。

```

\set AUTOCOMMIT off
SET edb_stmt_level_tx TO on;

INSERT INTO emp (empno,ename,deptno) VALUES (9001, JONES, 40);
INSERT INTO emp (empno,ename,deptno) VALUES (9002, JONES, 00);
ERROR: insert or update on table "emp" violates foreign key constraint
"emp_ref_dept_fk"
DETAIL: Key (deptno)=(0) is not present in table "dept"

SELECT empno, ename, deptno FROM emp WHERE empno > 9000;
__OUTPUT__
empno | ename | deptno
- - - - - + - - - - - + - - - - -
  9001 | JONES |    40
(1 row)

COMMIT;

```

`COMMIT` コマンドの代わりに `ROLLBACK` コマンドを発行すると、従業員番号 9001 の挿入もロールバックされます。

db_dialect

パラメータータイプ: Enum

デフォルト値: postgres

範囲: {postgres | redwood}

効果の最小範囲: セッションごと

値の変更が有効になるとき: 即時

アクティブ化に必要な承認: セッションユーザー

ネイティブ PostgreSQL システムカタログ `pg_catalog` に加えて、EDB Postgres Advanced Server には拡張カタログビューが含まれています。これは、展開されたカタログビューの `sys` カタログです。 `db_dialect` パラメーターは、名前解決のためにこれらのカタログが検索される順序を制御します。

`postgres` に設定すると、名前空間の優先順位は `pg_catalog`、次に `sys` になり、PostgreSQL カタログが最も優先されます。 `redwood` に設定すると、名前空間の優先順位は `sys`、次に `pg_catalog` になり、展開されたカタログビューが最も優先されます。

default_with_rowids

パラメータータイプ: ブール

デフォルト値: false

範囲: {true | false}

効果の最小範囲: セッションごと

値の変更が有効になるとき: 即時

アクティブ化に必要な承認: セッションユーザー

`CREATE TABLE` を `on` に設定すると、新しく作成されたテーブルに `ROWID` 列が含まれ、SQL コマンドで参照できます。 EDB Postgres Advanced Server の以前のバージョンでは、`ROWIDs` は `OIDs` にマップされました。 EDB Postgres Advanced Server バージョン 12 以降では、`ROWID` は 1 で始まるシーケンスに基づくオートインクリメント値です。`ROWIDs` オプションで作成されたテーブルの各行に割り当てられます。デフォルトでは、`ROWID` 列に一意のインデックスが作成されます。

`ALTER` および `DROP` 操作は、`ROWID` 列で制限されています。

EDB Postgres Advanced Server 11 以前のバージョンで `ROWIDs` を使用してデータベースを復元するには、次を実行する必要があります。

- `pg_dump` : テーブルに `OIDs` が含まれる場合、`--convert-oids-to-rowids` を指定してデータベースをダンプします。それ以外の場合は、`OIDs` を無視して、EDB Postgres Advanced Server バージョン 12 以降でテーブルの作成を続行します。
- `pg_upgrade` : エラーアウト。ただし、テーブルに `OIDs` または `ROWIDs` が含まれる場合は、次を実行する必要があります。1. `--convert-oids-to-rowids` オプションを指定して、テーブルのダンプを

取得します。2. テーブルを削除し、アップグレードを実行します。3. アップグレードが成功したら、ダンプされたテーブルを含む新しいクラスターにダンプをターゲットデータベースに復元します。

optimizer_mode

パラメータータイプ：Enum

デフォルト値：choose

範囲：{choose | ALL_ROWS | FIRST_ROWS | FIRST_ROWS_10 | FIRST_ROWS_100 | FIRST_ROWS_1000}

効果の最小範囲：セッションごと

値の変更が有効になるとき：即時

アクティブ化に必要な承認：セッションユーザー

オプティマイザーのヒントを分析するためのデフォルトの最適化モードを設定します。

次の表に、指定可能な値を示します。

Hint	Description
<i>ALL_ROWS</i>	結果セットのすべての行の取得を最適化します。
<i>CHOOSE</i>	結果セットから取得する想定行数に基づいたデフォルトの最適化は行いません。これがデフォルトです。
<i>FIRST_ROWS</i>	結果セットの最初の行のみを取得するように最適化します。
<i>FIRST_ROWS_10</i>	結果セットの最初の 10 行の取得を最適化します。
<i>FIRST_ROWS_100</i>	結果セットの最初の 100 行の取得を最適化します。
<i>FIRST_ROWS_1000</i>	結果セットの最初の 1000 行の取得を最適化します。

これらの最適化モードは、SQL コマンドを送信するクライアントが結果セットの最初の *n* 行のみを表示し、残りの結果セットを破棄するという仮定に基づいています。クエリに割り当てられたリソースはそうのように調整されます。

Customized options

custom_variable_classes

`custom_variable_classes` パラメーターは、EDB Postgres Advanced Server 9.2 で非推奨になりました。以前はこのパラメーターに依存していたパラメーターは、サポートがなくなりました。EDB Postgres Advanced Server の以前のリリースでは、`custom_variable_classes` は、手続き型言語などのアドオンモジュールによって追加されることが通常は知られていないパラメーターで必要でした。

dbms_alert.max_alerts

パラメータータイプ： 整数

デフォルト値： 100

範囲： 0～500

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

DBMS_ALERTS パッケージを使用して、システムで許可される同時アラートの最大数を指定します。

dbms_pipe.total_message_buffer

パラメータータイプ： 整数

デフォルト値： 30 Kb

範囲： 30 Kb～256 Kb

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

DBMS_PIPE パッケージに使用されるバッファの合計サイズを指定します。

index_advisor.enabled

パラメータータイプ： ブール

デフォルト値： true

範囲： {true | false}

効果の最小範囲： セッションごと

値の変更が有効になるとき： 即時

アクティブ化に必要な承認： セッションユーザー

EDB-PSQL または PSQL セッションで Index Advisor を一時的に中断します。この構成パラメーターを使用するには、Index Advisor プラグイン `index_advisor` を EDB-PSQL または PSQL セッションにロードする必要があります。

Index Advisor プラグインは次のようにロードできます。

```
$ psql -d edb -U enterprisedb
Password for user enterprisedb:
psql (14.0.0)
Type "help" for help.

edb=# LOAD index_advisor;
LOAD
```

SET コマンドを使用して、パラメーター設定を変更して、インデックスアドバイザーが代替のクエリプランを生成するかどうかを制御します。

```
edb=# SET index_advisor.enabled TO off;
SET
edb=# EXPLAIN SELECT * FROM t WHERE a < 10000;
__OUTPUT__

          QUERY PLAN

-----
Seq Scan on t (cost=0.00..1693.00 rows=9864 width=8)
  Filter: (a < 10000)
(2 rows)
```

```
edb=# SET index_advisor.enabled TO on;
SET
edb=# EXPLAIN SELECT * FROM t WHERE a < 10000;
__OUTPUT__

          QUERY PLAN

-----
Seq Scan on t (cost=0.00..1693.00 rows=9864 width=8)
  Filter: (a < 10000)
Result (cost=0.00..327.88 rows=9864 width=8)
  One-Time Filter: ==[ HYPOTHETICAL PLAN ]==::text
  -> Index Scan using "<hypothetical-index>:1" on t (cost=0.00..327.88
rows=9864 width=8)
    Index Cond: (a < 10000)
(6 rows)
```

edb_sql_protect.enabled

パラメータータイプ: ブール

デフォルト値: false

範囲: {true | false}

影響の最小範囲: クラスター

値の変更が有効になるとき:

アクティブ化に必要な承認: EPAS サービスアカウント

それらのロールによって発行された SQL ステートメントを分析し、edb_sql_protect.level の設定に従って反応することにより、SQL/Protect が保護されたロールをアクティブに監視しているかどうかを制御します。SQL/Protect で監視を開始する準備ができたなら、このパラメーターを on に設定します。

edb_sql_protect.level

パラメータータイプ: Enum

デフォルト値: passive

範囲: {learn | passive | active}

影響の最小範囲: クラスター

値の変更が有効になるとき:

アクティブ化に必要な承認: EPAS サービスアカウント

保護されたロールによって SQL ステートメントが発行されたときに SQL/Protect が実行するアクションを設定します。

このパラメーターを次のいずれかの値に設定して、学習モード、パッシブモード、またはアクティブモードを使用できます。

- **learn**。保護されたロールのアクティビティを追跡し、ロールが使用するリレーションを記録します。保護されたアプリケーションの予想される動作が学習されるように、SQL/Protect を最初に構成するときにこの値を使用します。
- **passive**。保護されたロールが定義されたルールに違反している場合に警告を発行しますが、SQL ステートメントの実行は停止しません。これは、SQL/Protect が保護されたロールの予想される動作を学習した後の次の手順です。これは基本的に侵入検出モードで動作します。モニタすれば、本番環境で実行できます。
- **active**。保護されたロールの無効なステートメントをすべて停止します。この動作は、危険なクエリの実行を防ぐ SQL ファイアウォールとして機能します。これは、攻撃者がアプリケーションの背後にあるデータベースの脆弱性ポイントと種類を特定しようとしている場合の早期の侵入テストに対して特に効果的です。SQL/Protect は、これらの脆弱性ポイントを閉じるだけでなく、ブロックされたクエリを追跡します。この動作により、攻撃者がシステムに侵入する別の方法を見つける前に、管理者に警告できます。

初めて SQL/Protect を使用する場合は、`edb_sql_protect.level` を `learn` に設定します。

`edb_sql_protect.max_protected_relations`

パラメータータイプ：整数

デフォルト値：1024

範囲：1～2147483647

影響の最小範囲：クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認：EPAS サービスアカウント

ロールごとに保護できるリレーションの最大数を設定します。サーバーの保護されたりレーションの総数は、保護されたりレーションの数に保護されたロールの数を掛けた数です。すべての保護されたりレーションは、共有メモリの領域を消費します。可能な限り保護されたりレーションの領域は、データベースサーバーの起動中に予約されます。

`edb_sql_protect.max_protected_relations` が有効な範囲外の値（たとえば、値 2,147,483,648）に設定されているときにサーバーが起動されると、データベースサーバーのログファイルに警告メッセージが記録されます。

```
2014-07-18 16:04:12 EDT WARNING: invalid value for parameter
"edb_sql_protect.max_protected_relations": "2147483648"
2014-07-18 16:04:12 EDT HINT: Value exceeds integer range.
```

データベースサーバーは正常に起動しますが、`edb_sql_protect.max_protected_relations` がデフォルト値の 1024 に設定されています。

パラメータの上限は整数データ型の最大値としてリストされていますが、実際の設定は、使用可能な共有メモリの量とデータベースサーバーの起動時に使用されるパラメータ値によって異なります。

必要な領域を共有メモリに予約できる限り、この値は受け入れられます。共有メモリの領域を予約できない値の場合、データベースサーバーの起動は次のようなエラーメッセージで失敗します。

```
2014-07-18 15:22:17 EDT FATAL: could not map anonymous shared memory:
Cannot allocate memory
2014-07-18 15:22:17 EDT HINT: This error usually means that PostgreSQLs
request for a shared memory segment exceeded available memory, swap
space or huge pages. To reduce the request size (currently 2070118400
bytes), reduce PostgreSQLs shared memory usage, perhaps by reducing
shared_buffers or max_connections.
```

この場合、データベースサーバーを正常に起動できるまでパラメータ値を減らします。

edb_sql_protect.max_protected_roles

パラメータータイプ： 整数

デフォルト値： 64

範囲： 1～2147483647

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

保護できるロールの最大数を設定します。

すべての保護されたロールは、共有メモリの領域を消費します。サーバーは、保護されたロールの数と保護されたリレーションの数（`edb_sql_protect.max_protected_relations`）の領域を予約します。データベースサーバの起動時に、可能な最大の保護ロール用の領域が予約されます。

`edb_sql_protect.max_protected_roles` が有効な範囲外の値（たとえば、値 2,147,483,648）に設定されているときにデータベースサーバが起動されると、データベースサーバのログファイルに警告メッセージが記録されます。

```

2014-07-18 16:04:12 EDT WARNING: invalid value for parameter
"edb_sql_protect.max_protected_roles": "2147483648"
2014-07-18 16:04:12 EDT HINT: Value exceeds integer range.

```

データベースサーバは正常に起動しますが、`edb_sql_protect.max_protected_roles` がデフォルト値の 64 に設定されています。

パラメータの上限は整数データ型の最大値としてリストされていますが、実際の設定は、使用可能な共有メモリの量とデータベースサーバの起動時に使用されるパラメータ値によって異なります。

必要な領域を共有メモリに予約できる限り、この値は受け入れられます。共有メモリの領域を予約できない値の場合、データベースサーバの起動は次のようなエラーメッセージで失敗します。

```

2014-07-18 15:22:17 EDT FATAL: could not map anonymous shared memory:
Cannot allocate memory
2014-07-18 15:22:17 EDT HINT: This error usually means that PostgreSQLs
request for a shared memory segment exceeded available memory, swap
space or huge pages. To reduce the request size (currently 2070118400
bytes), reduce PostgreSQLs shared memory usage, perhaps by reducing
shared_buffers or max_connections.

```

この場合、データベースサーバを正常に起動できるまでパラメータ値を減らします。

edb_sql_protect.max_queries_to_save

パラメータータイプ： 整数

デフォルト値： 5000

範囲： 100～2147483647

影響の最小範囲： クラスタ

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

ビュー `edb_sql_protect_queries` に保存する問題のあるクエリの最大数を設定します。

保存されたクエリはすべて共有メモリの領域を消費します。保存できる最大クエリ用の領域は、データベースサーバの起動中に予約されます。

`edb_sql_protect.max_queries_to_save` が有効な範囲外の値（たとえば、値 10）に設定されているときにデータベースサーバが起動されると、データベースサーバのログファイルに警告メッセージが記録されます。

```
2014-07-18 13:05:31 EDT WARNING:  10 is outside the valid range for
parameter "edb_sql_protect.max_queries_to_save" (100 .. 2147483647)
```

データベースサーバは正常に起動しますが、`edb_sql_protect.max_queries_to_save` がデフォルト値の 5000 に設定されています。

パラメータの上限は整数データ型の最大値としてリストされていますが、実際の設定は、使用可能な共有メモリの量とデータベースサーバの起動時に使用されるパラメータ値によって異なります。

必要な領域を共有メモリに予約できる限り、この値は受け入れられます。共有メモリの領域を予約できない値の場合、データベースサーバの起動は次のようなエラーメッセージで失敗します。

```
2014-07-18 15:22:17 EDT FATAL: could not map anonymous shared memory:
Cannot allocate memory
2014-07-18 15:22:17 EDT HINT: This error usually means that PostgreSQLs
request for a shared memory segment exceeded available memory, swap
space or huge pages. To reduce the request size (currently 2070118400
bytes), reduce PostgreSQLs shared memory usage, perhaps by reducing
shared_buffers or max_connections.
```

この場合、データベースサーバを正常に起動できるまでパラメータ値を減らします。

edb_wait_states.directory

パラメータータイプ： 文字列

デフォルト値： edb_wait_states

範囲： 該当なし

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

EDB の待機状態のログファイルが格納されるディレクトリパスを設定します。相対パスではなく、完全な絶対パスを使用します。ただし、デフォルト設定は `edb_wait_states` であるため、`$PGDATA/edb_wait_states` がデフォルトのディレクトリの場所になります。EDB の待機状態については、

Performance analysis and tuning の `EDB wait states` を参照してください。

edb_wait_states.retention_period

パラメータータイプ： 整数

デフォルト値： 604800s

範囲： 86400～2147483647

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

EDB の待機状態のログファイルを削除するまでの待機時間を設定します。デフォルトは 604,800 秒、つまり 7 日です。EDB の待機状態については、*Performance analysis and tuning* の `EDB wait dtates` を参照してください。

edb_wait_states.sampling_interval

パラメータータイプ： 整数

デフォルト値： 1s

範囲： 1～2147483647

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

EDB ウェイトステートの 2 つのサンプリングサイクル間のタイミング間隔を設定します。デフォルト設定は 1 秒です。EDB の待機状態については、*Performance analysis and tuning* の `EDB wait states` を参照してください。

edbldr.empty_csv_field

パラメータータイプ：Enum

デフォルト値：NULL

範囲：{NULL | empty_string | pgsql}

効果の最小範囲：セッションごと

値の変更が有効になるとき：即時

アクティブ化に必要な承認：セッションユーザー

edbldr.empty_csv_field パラメータを使用して、EDB*Loader が空の文字列を処理する方法を指定します。この表は、edbldr.empty_csv_field パラメーターの有効な値を示しています。

Parameter setting	EDB*Loader behavior
<i>NULL</i>	An empty field is treated as <i>NULL</i> .
<i>empty_string</i>	An empty field is treated as a string of length zero.
<i>pgsql</i>	An empty field is treated as a <i>NULL</i> if it doesn't contain quotes and as an empty string if it contains quotes.

EDB*Loader の edbldr.empty_csv_field パラメーターの詳細は、

Database Compatibility for Oracle Developers: Tools and Utilities を参照してください。

utl_encode.uudecode_redwood

パラメータータイプ：ブール

デフォルト値：false

範囲：{true | false}

効果の最小範囲：セッションごと

値の変更が有効になるとき：即時

アクティブ化に必要な承認：セッションユーザー

TRUE に設定すると、EDB Postgres Advanced Server の UTL_ENCODE.UUDECODE ファンクションは、UTL_ENCODE.UUENCODE ファンクションの Oracle 実装によって作成された uuencode されたデータをデコードできます。

FALSE に設定すると、EDB Postgres Advanced EDB Postgres Advanced の UTL_ENCODE.UUENCODE ファンクションで作成された uuencode されたデータをデコードできます。

utl_file.umask

パラメータータイプ：文字列

デフォルト値：0077

範囲：umask 設定の 8 進数

効果の最小範囲：セッションごと

値の変更が有効になるとき：即時

アクティブ化に必要な承認：セッションユーザー

utl_file.umask パラメーターは、Linux の `umask` コマンドと同様の方法でファイルモード作成マスクを設定します。これは、EDB Postgres Advanced Server UTL_FILE パッケージ内でのみ使用します。

注釈: `utl_file.umask` パラメーターは Windows システムでサポートされていません。

utl_file.umask に指定された値は、Linux `umask` コマンドで有効な 3 または 4 文字の 8 進文字列です。この設定は、UTL_FILE のファンクションとプロシージャによって作成されたファイルのアクセス許可を決定します。

以下は、デフォルトの utl_file.umask 設定 0077 の結果を示しています。enterprisedb グループに属するユーザーおよび他のすべてのユーザーでは、すべての権限が拒否されます。ユーザー enterprisedb のみがファイルの読み取りおよび書き込み権限を持っています。

```
- rw----- 1 enterprisedb enterprisedb 21 Jul 24 16:08 utlfile
```

Ungrouped

これらの構成パラメーターは EDB Postgres Advanced Server にのみ適用され、特定の制限された目的のためのものです。

nls_length_semantics

パラメータータイプ：Enum

デフォルト値：byte

範囲：{byte | char}

効果の最小範囲：セッションごと

値の変更が有効になるとき：即時

アクティブ化に必要な承認：スーパーユーザー

このパラメーターは EDB Postgres Advanced Server では効果がありません。たとえば、この形式の **ALTER SESSION** コマンドは、構文エラーをスローせずに EDB Postgres Advanced Server で受け入れられます。ただし、セッション環境は変更しません。

```
ALTER SESSION SET nls_length_semantics = char;
```

注釈: このパラメーターを設定してもサーバー環境に影響がないため、システムビュー `pg\_settings` には表示されません。

query_rewrite_enabled

パラメータータイプ: Enum

デフォルト値: false

範囲: {true | false | force}

効果の最小範囲: セッションごと

値の変更が有効になるとき: 即時

アクティブ化に必要な承認: セッションユーザー

このパラメーターは EDB Postgres Advanced Server では効果がありません。たとえば、この形式の **ALTER SESSION** コマンドは、構文エラーをスローせずに EDB Postgres Advanced Server で受け入れられます。ただし、セッション環境は変更しません。

```
ALTER SESSION SET query_rewrite_enabled = force;
```

注釈: このパラメーターを設定してもサーバー環境に影響がないため、システムビュー `pg\_settings` には表示されません。

query_rewrite_integrity

パラメータータイプ: Enum

デフォルト値: enforced

範囲: {enforced | trusted | stale_tolerated}

効果の最小範囲: セッションごと

値の変更が有効になるとき: 即時

アクティブ化に必要な承認: スーパーユーザー

このパラメーターは EDB Postgres Advanced Server では効果がありません。たとえば、この形式の `ALTER SESSION` コマンドは、構文エラーをスローせずに EDB Postgres Advanced Server で受け入れられます。ただし、セッション環境は変更しません。

```
ALTER SESSION SET query_rewrite_integrity = stale_tolerated;
```

注釈: このパラメーターを設定してもサーバー環境に影響がないため、システムビュー `pg_settings` には表示されません。

timed_statistics

パラメータータイプ: ブール

デフォルト値: true

範囲: {true | false}

効果の最小範囲: セッションごと

値の変更が有効になるとき: 即時

アクティブ化に必要な承認: セッションユーザー

Dynamic Runtime Instrumentation Tools Architecture (DRITA) 機能のタイミングデータの収集を制御します。on に設定すると、タイミングデータが収集されます。

注釈: EDB Postgres Advanced Server をインストールすると、`postgresql.conf` ファイルには `timed_statistics` を `off` に設定する明示的なエントリが含まれます。このエントリがコメントアウトされ、構成ファイルがリロードされると、時限統計収集ではデフォルト値である `on` が使用されます。

Audit archiver

これらの構成パラメーターは、EDB Postgres Advanced Server データベース監査アーカイブ機能によって使用されます。

edb_audit_archiver

パラメータータイプ: Enum

デフォルト値: false

範囲: {true | false}

影響の最小範囲: クラスター

値の変更が有効になるとき:

アクティブ化に必要な承認：EPAS サービスアカウント

データベース監査アーカイブを有効または無効にします。

edb_audit_archiver_timeout

パラメータータイプ：整数

デフォルト値：300s

範囲：30 秒～1 日

影響の最小範囲：クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認：EPAS サービスアカウント

データベースがログファイルをアーカイブしようとするときにタイムアウトを秒単位で適用します。

edb_audit_archiver_filename_prefix

パラメータータイプ：文字列

デフォルト値：audit-

範囲：該当なし

影響の最小範囲：クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認：EPAS サービスアカウント

アーカイブが必要な監査ログファイルのファイル名を指定します。ファイル名は、`edb_audit_filename` パラメーターと揃える必要があります。`edb_audit_directory` に `edb_audit_archiver_filename_prefix` を持つ監査ファイルは、圧縮または有効期限切れの対象です。

edb_audit_archiver_compress_time_limit

パラメータータイプ：整数

デフォルト値：-1

許可される値：0、-1、または秒単位の正の数値

影響の最小範囲：クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認：EPAS サービスアカウント

監査ログが圧縮の対象になるまでの時間を秒単位で指定します。このパラメーターを設定できる値は次のとおりです。

- 0。ログファイルが現在のファイルでなくなるとすぐに圧縮が開始されます。
- -1。タイムリーなログファイルの圧縮は発生しません。

edb_audit_archiver_compress_size_limit

パラメータータイプ： 整数

デフォルト値： -1

許可される値： 0、-1、またはメガバイト単位の正の数値

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

監査ログが圧縮の対象となるファイルサイズのしきい値をメガバイトで指定します。このパラメーターが-1に設定されている場合、サイズに基づいてログファイルは圧縮されません。

edb_audit_archiver_compress_command

パラメータータイプ： 文字列

デフォルト値： gzip %p

範囲： 該当なし

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

監査ログファイルの圧縮を実行するコマンドを指定します。edb_audit_archiver_compress_command のデフォルト値は gzip %p です。gzip は、ファイルを圧縮する標準の方法を提供します。文字列内の%p は、アーカイブするファイルのパス名に置き換えられます。

edb_audit_archiver_compress_suffix

パラメータータイプ： 文字列

デフォルト値： .gz

範囲： 該当なし

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

圧縮済みのログファイルのファイル名を指定します。ファイル名は `edb_audit_archiver_compress_command` に合わせる必要があります。デフォルトのファイル名は `.gz` です。

edb_audit_archiver_expire_time_limit

パラメータータイプ： 整数

デフォルト値： -1

許可される値： 0、-1、または秒単位の正の数値

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

監査ログが期限切れになるまでの時間を秒単位で指定します。このパラメーターを設定できる値は次のとおりです。

- 0。有効期限は、ログファイルが現在のファイルでなくなるとすぐに始まります。
- -1。タイムリーなログファイルの有効期限は発生しません。

edb_audit_archiver_expire_size_limit

パラメータータイプ： 整数

デフォルト値： -1

許可される値： 0、-1、またはメガバイト単位の正の数値

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認： EPAS サービスアカウント

監査ログが期限切れになるまでのファイルサイズのしきい値をメガバイトで指定します。このパラメーターが-1に設定されている場合、サイズに基づいたログファイルの有効期限は発生しません。

edb_audit_archiver_expire_command

パラメータータイプ： 文字列

デフォルト値： ''

範囲： 該当なし

影響の最小範囲： クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認：EPAS サービスアカウント

削除する前に期限切れの監査ログファイルで実行するコマンドを指定します。

edb_audit_archiver_sort_file

パラメータータイプ：文字列

デフォルト値：mtime

範囲：該当なし

影響の最小範囲：クラスター

値の変更が有効になるとき：

アクティブ化に必要な承認：EPAS サービスアカウント

最も古いログファイルを識別して、アルファベット順または `mtime` に基づいて並べ替えます。

- `mtime` は、ファイルの変更時刻に基づいてファイルをソートします。
- `alphabetic` は、ファイル名に基づいてファイルをアルファベット順にソートします。

3.2 Index Advisor

Index Advisor ユーティリティは、インデックスを付ける列を決定して、特定のワークロードのパフォーマンスを向上させるのに役立ちます。Index Advisor は B ツリー（単一列またはコンポジット）のインデックスタイプを考慮します。パフォーマンスを向上させる可能性のある他のインデックスタイプ、つまり GIN、GiST、および Hash は識別しません。Index Advisor 拡張機能は EDB Postgres Advanced Server とともにインストールされます。

Index Advisor は、EDB Postgres Advanced Server のクエリプランナーと連携して、クエリプランナーがそのようなインデックスが利用可能であるかのように実行コストを計算するために使用する**仮想インデックス**を作成します。Index Advisor は、ワークロードで提供される SQL クエリを分析してインデックスを識別します。

Index Advisor を使用して、次のいずれかの方法で SQL クエリを分析できます。

- Index Advisor ユーティリティプログラムを呼び出し、分析する SQL クエリを含むテキストファイルを指定します。Index Advisor は、推奨されるインデックスの `CREATE INDEX` ステートメントを含むテキストファイルを生成します。
- Index Advisor で分析する EDB-PSQL コマンドラインでクエリを提供します。
- Postgres Enterprise Manager (PEM) クライアントを介して Index Advisor にアクセスします。PEM クライアントを使用してアクセスすると、Index Advisor は SQL プロファイラーと連携し、SQL トレースでキャプチャされたコードに関するインデックス作成の推奨事項を提供します。PEM での SQL プロファイラーと Index Advisor の使用の詳細については、PEM ドキュメントの *Using the Index Advisor* を参照してください。

Index Advisor は、INSERT、UPDATE、DELETE、および SELECT ステートメントでインデックス付けの推奨を試みます。Index Advisor を呼び出すときは、次のいずれかの形式でワークロードを提供します。

- SQL ファイルでコマンドを提供している場合、一連のクエリ
- psql コマンドラインで SQL ステートメントを指定している場合、EXPLAIN ステートメント

Index Advisor は、指定されたクエリのクエリプランと推定実行コストを表示しますが、クエリは実行しません。

分析中に、Index Advisor は仮想インデックスの有無にかかわらずクエリ実行コストを比較します。仮想インデックスを使用した実行コストが、仮想インデックスを使用しない場合よりも小さい場合：

- 両方のプランが EXPLAIN ステートメントの出力で報告されます。
- 改善を定量化するメトリックが計算されます。
- Index Advisor は、インデックスの作成に必要な CREATE INDEX ステートメントを生成します。

実行コストを削減する仮想インデックスが見つからない場合、Index Advisor は EXPLAIN ステートメントの元のクエリプラン出力のみを表示します。

注釈: Index Advisor はテーブルにインデックスを作成しません。Index Advisor が提供する `CREATE INDEX` ステートメントを使用して、推奨されるインデックスをテーブルに追加します。

EDB Postgres Advanced Server で提供される拡張機能は、Index Advisor が分析によって生成されたインデックス作成の推奨事項を保存するテーブルを作成します。この拡張機能は、テーブルのファンクションとビューも作成して、結果の取得と解釈を簡素化します。

スクリプトの実行をやめる場合、Index Advisor は現在の Index Advisor セッションでのみ使用可能な一時テーブルに推奨事項をログに記録します。

index_advisor_components index_advisor_configuration using_index_advisor review-
ing_the_index_advisor_recommendations index_advisor_limitations

3.2.1 Index Advisor components

Index Advisor 共有ライブラリは、クエリプランナーと対話して、インデックス付けの推奨事項を作成します。

Windows では、EDB Postgres Advanced Server インストーラーは、EDB Postgres Advanced Server ホームディレクトリの libdir サブディレクトリに共有ライブラリを作成します。共有ライブラリは次のとおりです。

index_advisor.dll

Linux の場合、edb-asxx-server-indexadvisor RPM パッケージをインストールします。ここで、xx は EDB Postgres Advanced Server のバージョン番号です。共有ライブラリは次のとおりです。

index_advisor.so

`libdir` ディレクトリのライブラリをロードできるのはスーパーユーザーだけです。データベース管理者は、Index Advisor ファイルを `libdir` ディレクトリから EDB Postgres Advanced Server ホームディレクトリの下に `libdir/plugins` ディレクトリにコピーすることにより、非スーパーユーザーに Index Advisor の使用を許可できます。信頼できる非スーパーユーザーのみにプラグインへのアクセスを許可します。これは、実稼働環境では安全でない方法です。

インストーラーは、Index Advisor ユーティリティプログラムとセットアップスクリプトも作成します。

`pg_advise_index`

SQL クエリを含むユーザー指定の入力ファイルを読み取り、`CREATE INDEX` ステートメントを含むテキストファイルを生成するユーティリティプログラム。これらのステートメントを使用して、Index Advisor が推奨するインデックスを作成できます。`pg_advise_index` プログラムは、EDB Postgres Advanced Server ホームディレクトリの `bin` サブディレクトリにあります。

注釈: `pg_advise_index` は、最初に `$libdir/plugins` から `index_advisor` プラグインをロードするようにバックエンドプロセスに依頼します。見つからない場合、サーバーログファイルにエラーを書き込み、`$libdir` からロードしようとします。

`index_advisor_log`

Index Advisor は、インデックス作成の推奨事項を `index_advisor_log` テーブルに記録します。Index Advisor は、ユーザーの検索パスで `index_advisor_log` テーブルを見つけられない場合、インデックス付けの推奨事項を同じ名前の一時テーブルに保存します。一時テーブルは、現在のセッションの残りの部分にのみ存在します。

`show_index_recommendations()`

バックエンドプロセス ID で識別される特定の Index Advisor セッション中に行われた推奨を解釈して表示する PL/pgSQL ファンクション。

`index_recommendations`

Index Advisor は、クエリ分析中に `index_advisor_log` テーブルに格納された情報に基づいて `index_recommendations` ビューを作成します。ビューは `show_index_recommendations()` 関数と同じ形式で出力を生成しますが、保存されているすべてのセッションに対する Index Advisor の推奨事項が含まれています。`show_index_recommendations()` 関数によって返される結果セットは、指定されたセッションに制限されます。

3.2.2 Index Advisor configuration

Index Advisor は、現在のセッションの残りの部分でのみ使用可能な推奨事項を生成するための構成を必要としません。複数のセッションの結果を保存するには、EDB Postgres Advanced Server が Index Advisor の推奨事項を保存する `index_advisor_log` テーブルを作成する必要があります。`index_advisor_log` テーブルを作成するには、拡張機能 `index_advisor` を作成します。

Index Advisor のテーブル、ファンクション、およびビューのストレージスキーマを選択するときは、Index Advisor を呼び出して結果セットを照会するすべてのユーザーがスキーマに対する `USAGE` 特権を持ってい

必要があることに注意してください。スキーマは、Index Advisor と対話するすべてのユーザーの検索パスにある必要があります。

1. 選択したスキーマを `search_path` パラメーターの先頭に配置します。たとえば、現在検索パスが次のとおりであるとしします。

```
search_path=public, accounting
```

`advisor` という名前のスキーマに Index Advisor オブジェクトを作成する場合は、次のコマンドを使用します。

```
SET search_path = advisor, public, accounting;
```

2. データベースオブジェクトを作成する `index_advisor` 拡張機能を作成します。スーパーユーザーとしてログインし、次のコマンドを使用します。

```
edb=# create extension index_advisor ;
__OUTPUT__
CREATE EXTENSION
```

注釈: 以前のバージョンの EDB Postgres Advanced Server で Index Advisor を使用していて、バージョン 15 にアップグレードしている場合は、次のコマンドを使用して `index_advisor` 拡張機能を作成します。

```
edb=# create extension index_advisor version 1.1;
__OUTPUT__
CREATE EXTENSION
```

このコマンドは、拡張機能を作成し、古いデータベースオブジェクトをリンクします。

3. Grant 表に対する特権をすべての Index Advisor ユーザーに付与します。Index Advisor ユーザーがスーパーユーザーまたはこれらのデータベースオブジェクトの所有者の場合、この手順は必要ありません。

- Grant テーブルに対する SELECT および INSERT 特権を付与して、ユーザーが Index Advisor を呼び出せるようにします。 - Grant テーブルに DELETE 特権を付与して、指定されたユーザーがテーブルの内容を削除できるようにします。 - Grant ビューで SELECT 特権を付与します。

この例では、`ia` という名前のスキーマに Index Advisor データベースオブジェクトを作成する方法を示しています。このスキーマには、ユーザー名 `ia_user` の Index Advisor ユーザーがアクセスできます。

```
$ edb-psql -d edb -U enterprisedb
psql.bin (15.2.0, server 15.2.0)
Type "help" for help.

edb=# CREATE SCHEMA ia;
CREATE SCHEMA
edb=# SET search_path TO ia;
SET
```

(次のページに続く)

(前のページからの続き)

```

edb=# CREATE EXTENSION index_advisor;
CREATE EXTENSION
edb=# GRANT USAGE ON SCHEMA ia TO ia_user;
GRANT
edb=# GRANT SELECT, INSERT, DELETE ON index_advisor_log TO ia_user;
GRANT
edb=# GRANT SELECT ON index_recommendations TO ia_user;
GRANT

```

Index Advisor の使用中は、指定されたスキーマ (ia) を `ia_user search_path` パラメーターに含める必要があります。

3.2.3 Using Index Advisor

Index Advisor を呼び出すときは、ワークロードを提供する必要があります。ワークロードは、コマンドラインで指定されたクエリ、または一連のクエリを含むファイルです。これらのクエリは、`pg_advise_index()` 関数によって実行されます。ワークロードを分析した後、Index Advisor は結果セットを一時テーブルまたは永続テーブルに保存します。Index Advisor によって生成されたインデックス作成の推奨事項を確認し、Index Advisor によって生成された `CREATE INDEX` ステートメントを使用して、推奨されるインデックスを作成できます。

注釈: 読み取り専用トランザクションで Index Advisor を実行しないでください。

次の例では、スーパーユーザー `enterprisedb` が Index Advisor ユーザーであることを前提としています。Index Advisor データベースオブジェクトは、スーパーユーザー `enterprisedb` の `search_path` のスキーマに作成されました。

例では、次のステートメントで作成されたテーブルを使用します。

```

CREATE TABLE t( a INT, b INT );
INSERT INTO t SELECT s, 99999 - s FROM generate_series(0,99999) AS s;
ANALYZE t;

```

結果のテーブルには次の行が含まれます。

```

__OUTPUT__
a      | b
-----+-----
0      | 99999
1      | 99998
2      | 99997
3      | 99996
.
.

```

(次のページに続く)

(前のページからの続き)

```

.
99997 | 2
99998 | 1
99999 | 0

```

pg_advise_index ユーティリティの使用

pg_advise_index ユーティリティを呼び出すときは、pg_advise_index によって実行されるクエリを含むファイルの名前を含める必要があります。クエリは同じ行または別の行に置くことができますが、各クエリはセミコロンで終了する必要があります。ファイル内のクエリを EXPLAIN キーワードで始めることはできません。

この例は、サンプルの workload.sql ファイルの内容を示しています。

```

SELECT * FROM t WHERE a = 500;
SELECT * FROM t WHERE b < 1000;

```

pg_advise_index プログラムを実行します。

```

$ pg_advise_index -d edb -h localhost -U enterprisedb -s 100M -o advisory.sql
workload.sql
poolsize = 102400 KB
load workload from file workload.sql
Analyzing queries .. done.
size = 2184 KB, benefit = 1684.720000
size = 2184 KB, benefit = 1655.520000
/* 1. t(a): size=2184 KB, benefit=1684.72 */
/* 2. t(b): size=2184 KB, benefit=1655.52 */
/* Total size = 4368KB */

```

サンプルコードでは、-d、-h、および-U オプションは psql 接続オプションです。

-s

Index Advisor が推奨するインデックスの最大サイズを制限するオプションのパラメーター。Index Advisor が結果セットを返さない場合、-s の設定が低すぎる可能性があります。

-o

推奨されるインデックスは、-o オプションの後に指定されたファイルに書き込まれます。

pg_advise_index プログラムによって表示される情報は、index_advisor_log テーブルに記録されます。例に示されているコマンドに応答して、Index Advisor は次の CREATE INDEX ステートメントを advisory.sql 出力ファイルに書き込みます。

```

create index idx_t_1 on t (a);
create index idx_t_2 on t (b);

```

推奨されるインデックスは、ファイル内の `CREATE INDEX` ステートメントを使用して `psql` コマンドラインで作成できます。または、`advisory.sql` スクリプトを実行してインデックスを作成できます。

```
$ edb-psql -d edb -h localhost -U enterprisedb -e -f advisory.sql
create index idx_t_1 on t (a);
CREATE INDEX
create index idx_t_2 on t (b);
CREATE INDEX
```

注釈: `pg_advise_index` は、最初に ``$libdir/plugins`` から `index_advisor` プラグインをロードするようにバックエンドプロセスに要求します。見つからない場合、サーバーログファイルにエラーを書き込み、`$libdir` からロードしようとします。

psql コマンドラインで Index Advisor を使用する

Index Advisor を使用して、`edb-psql` または `psql` コマンドラインで入力された SQL ステートメントを分析できます。

Index Advisor プラグインをロードし、Index Advisor を使用するには：

1. `edb-psql` コマンドラインユーティリティを使用してサーバーに接続し、Index Advisor プラグインをロードします。

```
$ edb-psql -d edb -U enterprisedb
...
edb=# LOAD index_advisor;
LOAD
```

2. `edb-psql` コマンドラインを使用して、Index Advisor で分析する各 SQL コマンドを呼び出します。Index Advisor は、クエリの推奨事項を `index_advisor_log` テーブルに保存します。`index_advisor_log` テーブルがユーザーの `search_path` に存在しない場合、同じ名前で一時テーブルが作成されます。この一時テーブルは、ユーザーの現在のセッションの残りの間のみ存在します。

Index Advisor プラグインをロードすると、Index Advisor はすべての SQL ステートメントを分析し、残りのセッションのインデックス作成の推奨事項をログに記録します。

Index Advisor でクエリを分析し、クエリを実行せずにインデックス作成の推奨事項を作成するには、SQL ステートメントの前に `EXPLAIN` キーワードを付けます。

ステートメントの前に `EXPLAIN` キーワードを付けない場合、Index Advisor はステートメントの実行中にステートメントを分析します。後で確認できるように、インデックス付けの推奨事項を `index_advisor_log` テーブルに書き込みます。

この例では、`EXPLAIN` ステートメントは通常のクエリプランを表示します。クエリが推奨される仮想インデックスを使用していた場合、同じクエリのクエリプランが続きます。

```

edb=# EXPLAIN SELECT * FROM t WHERE a < 10000;
__OUTPUT__

              QUERY PLAN
-----
Seq Scan on t (cost=0.00..1693.00 rows=10105 width=8)
  Filter: (a < 10000)
Result (cost=0.00..337.10 rows=10105 width=8)
  One-Time Filter: ===[ HYPOTHETICAL PLAN ]===::text
  -> Index Scan using "<hypothetical-index>:1" on t
      (cost=0.00..337.10 rows=10105 width=8)
      Index Cond: (a < 10000)
(6 rows)

```

```

edb=# EXPLAIN SELECT * FROM t WHERE a = 100;
              QUERY PLAN
-----
Seq Scan on t (cost=0.00..1693.00 rows=1 width=8)
  Filter: (a = 100)
Result (cost=0.00..8.28 rows=1 width=8)
  One-Time Filter: ===[ HYPOTHETICAL PLAN ]===::text
  -> Index Scan using "<hypothetical-index>:3" on t
      (cost=0.00..8.28 rows=1 width=8)
      Index Cond: (a = 100)
(6 rows)

```

Index Advisor プラグインをロードした後の `index_advisor.enabled` のデフォルト値は `on` です。SET または SHOW コマンドを使用して `index_advisor.enabled` の現在の値を表示するには、Index Advisor プラグインをロードする必要があります。

`index_advisor.enabled` パラメーターを使用して、psql セッションを中断せずに Index Advisor を一時的に無効にすることができます。

```

edb=# SET index_advisor.enabled TO off;
SET

```

Index Advisor を有効にするには、パラメーターを `on` に設定します。

```

edb=# SET index_advisor.enabled TO on;
SET

```

3.2.4 Reviewing the Index Advisor recommendations

Index Advisor によって生成されたインデックスの推奨事項は、いくつかの方法で確認できます。次のことができます。

- `index_advisor_log` テーブルを照会します。
- `show_index_recommendations` ファンクションを実行します。
- `index_recommendations` ビューを照会します。

`show_index_recommendations()` 関数の使用

`show_index_recommendations()` ファンクションを使用して Index Advisor ユーティリティの推奨事項を確認するには、セッションのプロセス ID を指定してファンクションを呼び出します。

```
SELECT show_index_recommendations( pid );
```

`pid` は、現在のセッションのプロセス ID です。現在のセッションのプロセス ID がわからない場合、NULL の値を渡すと、現在のセッションの結果セットも返されます。

このコードフラグメントは、結果セットの行の例を示しています。

```
edb=# SELECT show_index_recommendations(null);
__OUTPUT__
               show_index_recommendations
-----
create index idx_t_a on t(a);/* size: 2184 KB, benefit: 3040.62,
gain: 1.39222666981456 */
(1 row)
```

この例では、`create index idx_t_a on t(a)` は、Index Advisor によって提案されたインデックスを作成するために必要な SQL ステートメントです。結果セットの各行は次のことを示しています。

- 推奨インデックスを作成するために必要なコマンド。
- インデックスの最大推定サイズ。
- インデックスを使用することの計算された利点。
- インデックスの実装から生じる推定ゲイン。

次のビューから、すべての Index Advisor セッションの結果を表示できます。

```
SELECT * FROM index_recommendations;
```

index_advisor_log テーブルのクエリ

Index Advisor は、インデックス作成の推奨事項を `index_advisor_log` という名前のテーブルに保存します。`index_advisor_log` テーブルの各行には、インデックスアドバイザーがそのクエリの実行コストを削減するために架空のインデックスを推奨できると判断したクエリの結果が含まれています。

Column	Type	Description
<i>relويد</i>	<i>oid</i>	インデックスの実表の OID
<i>relname</i>	<i>name</i>	インデックスの実表の名前
<i>attrs</i>	<i>integer[]</i>	推奨されるインデックス列 (列番号で識別)
<i>benefit</i>	<i>real</i>	このクエリのインデックス の計算された利点
<i>index_size</i>	<i>integer</i>	ディスクページの推定イン デックスサイズ
<i>backend_pid</i>	<i>integer</i>	このレコメンデーションを 生成するプロセスのプロセ ス ID
<i>timestamp</i>	<i>timestamp</i>	おすすが生成された日時

psql コマンドラインで `index_advisor_log` テーブルを照会できます。この例は、2つの Index Advisor セッションの結果得られた `index_advisor_log` テーブルエントリを示しています。各セッションには2つのクエリが含まれ、テーブル内で異なる `backend_pid` 値によって識別できます。セッションごとに、Index Advisor は2つのインデックスの推奨事項を生成しました。

```

edb=# SELECT * FROM index_advisor_log;
__OUTPUT__
 reloid  | relname | attrs | benefit | index_size | backend_pid | timestamp
-----+-----+-----+-----+-----+-----+-----
-----
 16651   |      t  | {1}   | 1684.72 |      2184   |      3442   | 22-MAR-11
16:44:32.712638 -04:00
 16651   |      t  | {2}   | 1655.52 |      2184   |      3442   | 22-MAR-11
16:44:32.759436 -04:00
 16651   |      t  | {1}   | 1355.9  |      2184   |      3506   | 22-MAR-11
16:48:28.317016 -04:00
 16651   |      t  | {1}   | 1684.72 |      2184   |      3506   | 22-MAR-11
16:51:45.927906 -04:00
(4 rows)

```

Index Advisor は、`pg_advise_index` ユーティリティによって実行された次の 2 つのクエリを分析した後、最初の 2 行をテーブルに追加しました。

```
SELECT * FROM t WHERE a = 500;
SELECT * FROM t WHERE b < 1000;
```

backend_pid 列の 3442 の値は、これらの結果がプロセス ID が 3442 のセッションからのものであることを示します。

最初の行の attrrs 列の 1 の値は、仮想インデックスが表の最初の列（表 t の列 a）にあることを示しています。

2 行目の attrrs 列の 2 の値は、仮想インデックスが表の 2 列目にあることを示しています（表 t の列 b）。

Index Advisor は、psql コマンドラインで実行された次の 2 つのクエリを分析した後、最後の 2 行をテーブルに追加しました。

```
edb=# EXPLAIN SELECT * FROM t WHERE a < 10000;
__OUTPUT__

              QUERY PLAN
-----
Seq Scan on t (cost=0.00..1693.00 rows=10105 width=8)
  Filter: (a < 10000)
Result (cost=0.00..337.10 rows=10105 width=8)
  One-Time Filter: ===[ HYPOTHETICAL PLAN ]===::text
  -> Index Scan using "<hypothetical-index>:1" on t (cost=0.00..337.10
rows=10105 width=8)
      Index Cond: (a < 10000)
(6 rows)
```

```
edb=# EXPLAIN SELECT * FROM t WHERE a = 100;
__OUTPUT__

              QUERY PLAN
-----
Seq Scan on t (cost=0.00..1693.00 rows=1 width=8)
  Filter: (a = 100)
Result (cost=0.00..8.28 rows=1 width=8)
  One-Time Filter: ===[ HYPOTHETICAL PLAN ]===::text
  -> Index Scan using "<hypothetical-index>:3" on t (cost=0.00..8.28
rows=1 width=8)
      Index Cond: (a = 100)
(6 rows)
```

index_advisor_log 表の benefit 列の値は、次の式を使用して計算されます。

$$\text{benefit} = (\text{normal execution cost}) - (\text{execution cost with hypothetical index})$$

例に示されている index_advisor_log テーブルの最終行の benefit 列の値は、次の SQL ステートメントのクエリプランを使用して計算されます。

```
EXPLAIN SELECT * FROM t WHERE a = 100;
```

さまざまな実行計画の実行コストが評価および比較されます。

```
benefit = (Seq Scan on t cost) - (Index Scan using <hypothetical-index>)
```

特典がテーブルに追加されます。

```
benefit = 1693.00 - 8.28
```

```
benefit = 1684.72
```

行に格納されているクエリの結果を確認する必要がなくなったら、`index_advisor_log` テーブルから行を削除できます。

index_recommendations ビューの照会

`index_recommendations` ビューには、計算されたメトリックと、結果が現在 `index_advisor_log` テーブルにあるすべてのセッションの推奨されるインデックスを作成するための `CREATE INDEX` ステートメントが含まれています。`index_recommendations` ビューを照会することにより、保存されているすべての Index Advisor セッションの結果を表示できます。

```
SELECT * FROM index_recommendations;
```

:ref:`index\_advisor\_log` テーブルのクエリ<`index_advisor_log` テーブルのクエリ>` の例を使用すると、 ``index\_recommendations``

ビューに次が表示されます。

```
edb=# SELECT * FROM index_recommendations;
__OUTPUT__
 backend_pid | show_index_recommendations
-----+-----
 3442      | create index idx_t_a on t(a);/* size: 2184 KB, benefit:
1684.72, gain: 0.771392654586624 */
 3442      | create index idx_t_b on t(b);/* size: 2184 KB, benefit:
1655.52, gain: 0.758021539820856 */
 3506      | create index idx_t_a on t(a);/* size: 2184 KB, benefit:
3040.62, gain: 1.39222666981456 */
(3 rows)
```

各セッションで、同じ推奨インデックスを利用するすべてのクエリの結果が結合され、推奨インデックスごとに1セットのメトリックが生成され、`benefit` および `gain` という名前のフィールドに反映されます。

フィールドの式は次のとおりです。

```
size = MAX(index size of all queries)
benefit = SUM(benefit of each query)
gain = SUM(benefit of each query) / MAX(index size of all queries)
```

したがって、たとえば、次のクエリを使用すると、`backend_pid` が 3506 のプロセスから結果が得られます。

```
__OUTPUT__
reloid | relname | attrs | benefit | index_size | backend_pid | timestamp
-----+-----+-----+-----+-----+-----+-----
16651 | t       | {1}   | 1355.9   | 2184       | 3506       | 22-MAR-11
16:48:28.317016 -04:00
16651 | t       | {1}   | 1684.72  | 2184       | 3506       | 22-MAR-11
16:51:45.927906 -04:00
```

`backend_pid 3506` の `index_recommendations` ビューから表示されるメトリックは次のとおりです。

```
__OUTPUT__
backend_pid | show_index_recommendations
-----+-----
3506       | create index idx_t_a on t(a);/* size: 2184 KB, benefit:
3040.62, gain: 1.39222666981456 */
```

ビューのメトリックは次のように計算されます。

```
benefit = (benefit from 1st query) + (benefit from 2nd query)
benefit = 1355.9 + 1684.72
benefit = 3040.62
```

以下に加えて：

```
gain = ((benefit from 1st query) + (benefit from 2nd query)) / MAX(index
size of all queries)
gain = (1355.9 + 1684.72) / MAX(2184, 2184)
gain = 3040.62 / 2184
gain = 1.39223
```

ゲインメトリックは、特定のセッション中に導き出されたさまざまな推奨インデックスの相対的な利点を比較するときに役立ちます。ゲインの値が大きいほど、インデックスが使用する可能性のあるディスク領域と比較して、インデックスから導き出される費用対効果が優れています。

3.2.5 Index Advisor limitations

Index Advisor は Index Only スキャンを考慮しません。推奨事項を作成するときにインデックススキャンを考慮します。

Index Advisor は、`WHERE` 句で見つかった計算を無視します。事実上、レコメンデーションのインデックスフィールドはいかなる種類の式でもありません。フィールドは単純な列名です。

Index Advisor は、仮想インデックスを推奨するときに継承を考慮しません。クエリが親テーブルを参照する場合、Index Advisor は子テーブルに対してインデックスの推奨を行いません。

`index_advisor_log` テーブル、またはインデックス作成の推奨が行われ `index_advisor_log` テーブルに保存されたテーブルを含む `pg_dump` バックアップファイルを復元するとします。オブジェクト識別子 (OID) を変更すると、`index_advisor_log` テーブルと、`index_advisor_log` テーブルの行によって参照される復元されたテーブルとの間の「リンクが切断」される場合があります。

バックアップ前に行われた推奨事項を表示する必要がある場合は、`SQL UPDATE` ステートメントを使用して、`index_advisor_log` テーブルの `reloid` 列の古い OID を参照先のテーブルの新しい OID に置き換えることができます。

```
UPDATE index_advisor_log SET reloid = new_oid WHERE reloid = old_oid;
```

3.3 SQL Profiler

非効率的な SQL コードは、データベースのパフォーマンス問題の主な原因の 1 つです。データベース管理者と開発者の課題は、大規模で複雑なシステムでこのコードを見つけて最適化することです。

- SQL Profiler* は、実行が不十分な SQL コードを見つけて最適化するのに役立ちます。

SQL プロファイラーの特定の機能と利点は次のとおりです。

- オンデマンドのトレース。パラメーターを手動で設定し、トレースを開始することにより、いつでも SQL トレースをキャプチャできます。
- スケジュールされたトレース。都合の悪いときに、トレースパラメーターを指定して、後で実行するようにスケジュールすることもできます。
- トレースを保存します。トレースを実行し、後で確認できるように保存します。
- トレースフィルター。SQL キャプチャをデータベースおよびユーザー別に選択的にフィルタリングするか、すべてのデータベースに対してすべてのユーザーが送信したすべての SQL ステートメントをキャプチャします。
- トレース出力アナライザー。グラフィカルテーブルを使用すると、期間またはステートメントでクエリをすばやく並べ替えてフィルター処理でき、グラフィカルまたはテキストベースの `EXPLAIN` プランはクエリのパスと結合をレイアウトします。
- Index Advisor の統合。遅いクエリを見つけて最適化したら、インデックスアドバイザーに基になるテーブルインデックスの作成を推奨して、パフォーマンスをさらに向上させることもできます。

次に、インストール手順について説明します。

ステップ 1: SQL プロファイラーをインストールする

SQL プロファイラーは、WindowsEDB Postgres Advanced Server インストーラーによって、Linux では `edb-asxx-server-sqlprofiler` RPM パッケージからインストールされます。xx は EDB Postgres Advanced Server のバージョン番号です。

手順 2: SQL プロファイラーライブラリを追加する

インスタンスの `postgresql.conf` パラメーターファイルを変更して、SQL プロファイラーライブラリを `shared_preload_libraries` 構成パラメーターに含めます。

Linux インストールの場合、パラメーター値には次のものが含まれます。

```
$libdir/sql-profiler
```

Windows では、パラメーター値に次を含める必要があります。

```
$libdir\sql-profiler.dll
```

ステップ 3: SQL プロファイラーで使用される関数を作成する

SQL プロファイラーインストールプログラムは、SQL スクリプト (`sql-profiler.sql` という名前) を次の場所に配置します。

Linux の場合：

```
/usr/edb/as14/share/contrib/
```

Windows：

```
C:\Program Files\edb\as14\share\contrib\
```

`psql` コマンドラインインターフェイスを使用して、プロファイルするサーバー上のメンテナンスデータベースとして指定されたデータベースで `sql-profiler.sql` スクリプトを実行します。EDB Postgres Advanced Server を使用している場合、デフォルトのメンテナンスデータベースの名前は `edb` です。PostgreSQL インスタンスを使用している場合、デフォルトのメンテナンスデータベースの名前は `postgres` です。

次のコマンドは、`psql` コマンドラインを使用して、Linux システムで `sql-profiler.sql` スクリプトを呼び出します。

```
$ /usr/edb/as14/bin/psql -U user_name database_name <
/usr/edb/as14/share/contrib/sql-profiler.sql
```

手順 4: サーバーを停止して再起動して、変更を有効にします。

SQL プロファイラーを構成すると、サーバー上にあるすべてのデータベースで使用できるようになります。EDB Postgres Enterprise Manager で SQL プロファイラー機能を利用できます。Postgres Enterprise Manager の詳細については、次の EDB Web サイトを参照してください。

<https://www.enterprisedb.com/docs>

3.3.1 トラブルシューティング

(新しいバージョンの SQL プロファイラーへのアップグレードを実行した後)、次のテキストを含むエラーが発生した場合。

```
An error has occurred:
ERROR: function return row and query-specified return row do not match.
DETAIL: Returned row contains 11 attributes, but the query expects 10.
```

このエラーを修正するには、既存のクエリセットを新しいクエリセットに置き換える必要があります。最初に、`uninstall-sql-profiler.sql` スクリプトを呼び出して SQL Profiler をアンインストールし、`sql-profiler.sql` スクリプトを呼び出して SQL Profiler を再インストールします。

3.4 pgsnmpd

注釈: 'pgsnmpd' は、 EDB Postgres Advanced Server v14 の時点で非推奨です。 EDB Postgres Advanced Server v15 以降には含まれていません。

`pgsnmpd` は、Linux システム上の EDB Postgres Advanced Server の現在の状態に関する階層情報を返すことができる SNMP エージェントです。`pgsnmpd` は、`edb-asxx-pgsnmpd` RPM パッケージを使用して配布およびインストールされます。ここで、`xx` は EDB Postgres Advanced Server のバージョン番号です。`pgsnmpd` エージェントは、スタンドアロン SNMP エージェント、パススルーサブエージェント、または AgentX サブエージェントとして動作できます。

EDB Postgres Advanced Server をインストールした後、`LD_LIBRARY_PATH` 変数を更新する必要があります。次のコマンドを使用します。

```
$ export LD_LIBRARY_PATH=/usr/edb/as14/lib:$LD_LIBRARY_PATH
```

このコマンドは、`LD_LIBRARY_PATH` の値を永続的に変更しません。`LD_LIBRARY_PATH` の値を永続的に設定する方法については、Linux ディストリビューションのドキュメントを参照してください。

次の例は、読み取り専用アクセスを実装する `pgsnmpd` の最も簡単な使用法を示しています。`pgsnmpd` は `net-snmp` ライブラリに基づいています。`net-snmp` の詳細については、次の Web サイトをご覧ください。

<http://net-snmp.sourceforge.net/>

3.4.1 pgsnmpd の設定

`pgsnmpd` 構成ファイルの名前は `snmpd.conf` です。構成ファイルで指定できる指示子については、`snmpd.conf` のマニュアルページ (`man snmpd.conf`) を確認してください。

構成ファイルは手動で作成するか、`snmpconf perl` スクリプトを使用して構成ファイルを作成できます。`perl` スクリプトは `net-snmp` パッケージで配布されています。

`net-snmp` は、次の場所から入手できるオープンソースパッケージです。

<http://www.net-snmp.org/>

`snmpconf` 構成ファイルウィザードを使用するには、`net-snmp` をダウンロードしてインストールします。インストールが完了したら、コマンドラインを開き、次のように入力します。

```
snmpconf
```

構成ファイルウィザードが開くと、既存の構成ファイルを読み取るように求められる場合があります。none と入力して、新しい構成ファイルを生成します（以前に存在した構成ファイルに基づいていません）。

snmpconf は、メニュー駆動型のウィザードです。メニュー項目 1: snmpd.conf を選択して、構成ウィザードを起動します。snmpconf によって表示されるトップレベルの各メニューオプションを選択すると、ウィザードは一連の質問に進み、構成ファイルのビルドに必要な情報の入力を求めます。システムに関連する各カテゴリの情報を提供したら、Finished と入力して snmpd.conf という名前の構成ファイルを生成します。ファイルをコピーします。

```
/usr/edb/as14/share/
```

3.4.2 リスナーアドレスの設定

デフォルトでは、pgsnmpd はポート 161 でリッスンします。リスナーポートが別のサービスによって既に使用されている場合、次のエラーが表示される場合があります。

```
Error opening specified endpoint "udp:161".
```

snmpd.conf ファイルに次の行を追加して、代替リスナーポートを指定できます。

```
agentaddress $host_address:2000
```

この例では、UDP ポート 2000 でリッスンするように pgsnmpd に指示します。ここで、\$host_address はサーバーの IP アドレスです（例、127.0.0.1）。

pgsnmpd を呼び出す前に、次のコマンドを使用して pgsnmpd スキーマを作成し、検索パスを設定して pgsnmpd スキーマにテーブルを作成する必要があります。

```
CREATE SCHEMA pgsnmpd;
SET search_path = pgsnmpd;
\i pgsnmpd.sql
```

3.4.3 pgsnmpd の呼び出し

EDB Postgres Advanced Server のインスタンスが稼働していることを確認します（pgsnmpd がこのサーバーに接続します）。次の形式のコマンドで pgsnmpd を呼び出す前に、コマンドラインを開き、スーパーユーザ権限を引き受けます。

```
POSTGRES_INSTALL_HOME/bin/pgsnmpd -b
-c POSTGRES_INSTALL_HOME/share/snmpd.conf
-C "user=enterprisedb dbname=edb password=safe_password
port=5444"
```

POSTGRES_INSTALL_HOME は、EDB Postgres Advanced Server のインストールディレクトリを指定します。

pgsnmpd をバックグラウンドで実行するように指定するには、-b オプションを含めます。

pgsnmpd 構成ファイルのパスと名前を指定して、`-c` オプションを含めます。

`-C` オプションの後に、EDB Postgres Advanced Server のインストールの接続情報（`libpq` 接続文字列の形式）を含めます。

3.4.4 pgsnmpd ヘルプの表示

pgsnmpd ユーティリティを呼び出すときに`--help` オプションを含めて、他の pgsnmpd コマンドラインオプションを表示します。

```
pgsnmpd --help
Version PGSQL-SNMP-Ver1.0
usage: pgsnmpd [-s] [-b] [-c FILE ] [-x address ] [-g] [-C "Connect String"]
  -s : run as AgentX sub-agent of an existing snmpd process
  -b : run in the background
  -c : configuration file name
  -g : use syslog
  -C : libpq connection string
  -x : address:port of a network interface
  -V : display version strings
```

3.4.5 pgsnmpd に情報を要求する

`net-snmp` コマンドを使用して、pgsnmpd サービスを照会できます。例：

```
snmpgetnext -v 2c -c public localhost .1.3.6.1.4.1.5432.1.4.2.1.1.0
```

上記の例では：

`-v 2c` オプションは、SNMP バージョン 2c 形式で要求を送信するように `snmpgetnext` クライアントに指示します。

`-c public` は、コミュニティ名を指定します。

`localhost` は、pgsnmpd サーバーを実行しているホストマシンを示します。

`.1.3.6.1.4.1.5432.1.4.2.1.1.0` は、要求されたオブジェクトの ID です。すべてのデータベースのリストを表示するには、最後の桁を 1 つ増やします（例.1.1、.1.2、.1.3 など）。

特定のオブジェクトを照会するために必要なエンコーディングは、MIB（管理情報ベース）で定義されています。SNMP クライアントはさまざまなサーバーを監視できます。サーバーの種類によって、特定のサーバーによって公開される情報が決まります。各 SNMP サーバーは、公開されたデータを MIB（管理情報ベース）の形式で説明します。デフォルトでは、pgsnmpd は次の場所で MIB を検索します。

```
/usr/share/snmp/mibs
```

```
$HOME/.snmp/mibs
```

3.5 EDB audit logging

EDB Postgres Advanced Server を使用すると、データベースとセキュリティの管理者、監査人、およびオペレーターは、EDB 監査ログを使用してデータベースアクティビティを追跡および分析できます。EDB 監査ログは、関連するすべての情報を含む監査ログファイルを生成します。次のような情報を記録するように監査ログを構成できます。

- ロールが EDB Postgres Advanced Server データベースへの接続を確立するとき
- EDB Postgres Advanced Server に接続したときにロールが作成、変更、または削除するデータベースオブジェクト
- 認証に失敗した場合

構成ファイル `postgresql.conf` または `postgresql.auto.conf` で指定されたパラメーターは、監査ログに含まれる情報を制御します。

```
audit_logging_configuration_parameters selection_sql_statements_to_audit enabled_audit_logging audit_log_file using_error_codes_to_filter_audit_logs using_command_tags_to_filter_audit_logs redacting_passwords_from_audit_logs audit_log_archiving object_auditing
```

3.5.1 Audit logging configuration parameters

次の構成パラメーターを使用して、データベース監査を制御します。

Summary of configuration parameters を参照して、構成パラメーターが変更されたかどうかを判断します。

- すぐに有効になります
- 構成のリロードが必要
- データベースサーバの再起動が必要

`edb_audit`

データベース監査を有効または無効にします。 `xml` または `csv` の値はデータベース監査を有効にします。これらの値は、監査情報をキャプチャするファイル形式を表します。 `none` はデータベース監査を無効にします。これがデフォルトです。

注釈: `logging_collector` パラメーターを `on` に設定して、`edb_audit` パラメーターを有効にします。

`edb_audit_archiver`

データベース監査アーカイブを有効または無効にします。

`edb_audit_archiver_timeout`

データベースがログファイルをアーカイブしようとするときにタイムアウトを秒単位で適用します。有効な範囲は 30 秒から 1 日です。

`edb_audit_archiver_filename_prefix`

アーカイブが必要な監査ログファイルのファイル名を指定します。ファイル名は、`edb_audit_filename` パラメーターと揃える必要があります。`edb_audit_archiver_filename_prefix` のデフォルト値は `audit-` です。`edb_audit_directory` に `edb_audit_archiver_filename_prefix` を持つ監査ファイルは、圧縮と有効期限切れの対象です。

`edb_audit_archiver_compress_time_limit`

監査ログが圧縮の対象になるまでの時間を秒単位で指定します。可能な値は次のとおりです。ログファイルが現在のファイルでなくなるとすぐに圧縮が開始されます。`-1` — タイムリーなログファイルの圧縮は発生しません。

`edb_audit_archiver_compress_size_limit`

監査ログが圧縮の対象になるまでのファイルサイズのしきい値をメガバイト単位で指定します。パラメーターが `-1` に設定されている場合、サイズベースでのログファイルの圧縮は発生しません。

`edb_audit_archiver_compress_command`

監査ログファイルの圧縮を実行するコマンドを指定します。`edb_audit_archiver_compress_command` のデフォルト値は `gzip %p` です。`gzip` は、ファイルを圧縮する標準の方法を提供します。文字列内の `%p` は、アーカイブするファイルのパス名に置き換えられます。

`edb_audit_archiver_compress_suffix`

圧縮済みのログファイルのファイル名を指定します。ファイル名は `edb_audit_archiver_compress_command` に合わせる必要があります。デフォルトのファイル名は `.gz` です。

`edb_audit_archiver_expire_time_limit`

監査ログが期限切れになるまでの時間を秒単位で指定します。このパラメーターを設定できる値は次のとおりです。有効期限は、ログファイルが現在のファイルでなくなるとすぐに始まります。`-1` — タイムリーなログファイルの有効期限は発生しません。

`edb_audit_archiver_expire_size_limit`

監査ログが期限切れになるまでのファイルサイズのしきい値をメガバイトで指定します。パラメーターが `-1` に設定されている場合、サイズベースでのログファイルの有効期限は発生しません。

`edb_audit_archiver_expire_command`

削除する前に期限切れの監査ログファイルで実行するコマンドを指定します。

`edb_audit_archiver_sort_file`

最も古いログファイルを識別して、アルファベット順または `mtime` に基づいて並べ替えます。`mtime` は、ファイル変更時間に基づいたファイルの並べ替えを示します。`alphabetic` は、ファイル名に基づいてファイルをアルファベット順に並べ替えることを示します。

`edb_audit_directory`

ログファイルが作成されるディレクトリを指定します。ディレクトリのパスは、データフォルダーに対する相対パスまたは絶対パスです。デフォルトは `PGDATA/edb_audit` ディレクトリです。

`edb_audit_filename`

監査情報を格納する監査ファイルのファイル名を指定します。デフォルトのファイル名は `audit-%Y%m%d_%H%M%S` です。エスケープシーケンス `%Y`、`%m` などは、システムの日付と時刻に従って、適切な現在値に置き換えられます。

`edb_audit_rotation_day`

監査ファイルをローテーションする曜日を指定します。有効な値は、`sun`、`mon`、`tue`、`wed`、`thu`、`fri`、`sat`、`every`、および `none` です。回転を無効にするには、値を `none` に設定します。毎日ファイルをローテーションするには、`edb_audit_rotation_day` の値を `every` に設定します。特定の曜日にファイルをローテーションするには、値を目的の曜日に設定します。`every` がデフォルト値です。

`edb_audit_rotation_size`

ファイルのローテーションを強制的に行うときのファイルサイズのしきい値をメガバイト単位で指定します。デフォルト値は `0MB` です。パラメーターがコメントアウトされるか `0` に設定されている場合、サイズベースでのファイルのローテーションは発生しません。

`edb_audit_rotation_seconds`

ログファイルを作成するローテーション時間を秒単位で指定します。この機能が無効にするには、このパラメーターをデフォルトの `0` に設定します。

`edb_audit_connect`

ユーザーによるデータベース接続試行の監査を有効にします。すべての接続試行の監査を無効にするには、`edb_audit_connect` を `none` に設定します。すべての接続試行を監査するには、値を `all` に設定します。失敗した接続試行をすべて監査するには、値をデフォルトの `failed` に設定します。

`edb_audit_disconnect`

接続しているユーザーによるデータベース切断の監査を有効にします。切断の監査を有効にするには、値を `all` に設定します。無効にするには、値をデフォルトの `none` に設定します。

`edb_audit_statement`

さまざまなカテゴリの SQL ステートメントの監査を指定します。You can specify various combinations of the following values: `none` , `dml` , `insert` , `update` , `delete` , `truncate` , `select` , `error` , `rollback` , `ddl` , `create` , `drop` , `alter` , `grant` , `revoke` , `set` , `all` , and `{ select | update | delete | insert }@groupname` . デフォルトは `ddl` および `error` です。このパラメーターの設定については、

Selecting SQL statements to audit を参照してください。

`edb_audit_tag`

各エントリの監査ログファイルに追跡タグとして含める文字列値を指定します。

`edb_log_every_bulk_value`

一括処理は、結果のステートメントを EDB Postgres Advanced Server ログファイルと EDB Audit ログファイルの両方に記録します。ただし、一括処理ですべてのステートメントをログに記録するにはコストがかかります。これは、`edb_log_every_bulk_value` 構成パラメーターで制御できます。`true` に設定すると、一括処理のすべてのステートメントがログに記録されます。一括実行中に `edb_log_every_bulk_value` が `false` に設定されている場合、ログメッセージは処理された行数とともに一括処理ごとに記録されます。さらに、期間は一括処理ごとに 1 回発行されます。デフォルトは `false` です。

`edb_audit_destination`

`edb_audit_directory` パラメーターで指定されたディレクトリに監査ログ情報を記録するか、`syslog` プロセスによって管理されるディレクトリとファイルに記録するかを指定します。デフォルト設定である `edb_audit_directory` で指定されたディレクトリを使用するには、`file` に設定します。

`/etc/syslog.conf` ファイルで構成された `syslog` プロセスとその場所を使用するには、`syslog` に設定します。`syslog` 設定は、Linux ホストで実行されている EDB Postgres Advanced Server で有効であり、Windows システムではサポートされていません。.. **Note::**

最近の Linux バージョンでは、`syslog` は `rsyslog` に置き換えられ、構成ファイルは `/etc/rsyslog.conf` にあります。

注釈: EDB Postgres Advanced Server により、管理権限に関連付けられた管理ユーザーは、任意のユーザー、グループ、またはロールによるステートメントを監査できます。特定のユーザーを監査することにより、生成される監査レコードの数を最小限に抑えることができます。詳細については、[Selecting SQL statements to audit](#) の例を参照してください。

3.5.2 Selecting SQL statements to audit

`edb_audit_statement` には、監査する SQL 文を制御する 1 つ以上のコンマ区切り値を含めることができます。一般的なフォーマットは次のとおりです。

```
edb_audit_statement = value_1[, value_2]...
```

コンマ区切りの値には、コンマに続くスペース文字を含めることも省略することもできます。小文字または大文字を任意に組み合わせて値を指定できます。

基本的なパラメーター値は次のとおりです。

- **all** — ステートメントのエラーメッセージを含むすべてのステートメントを監査してログに記録します。
- **none** — すべての監査とログを無効にします。 **none** の値は、リストに含まれる他の値をオーバーライドします。
- **ddl** — すべてのデータ定義言語 (DDL) ステートメント (**CREATE**、**ALTER**、および **DROP**) と、**GRANT** および **REVOKE** データ制御言語 (DCL) ステートメントを監査します。
- **dml** — すべてのデータ操作言語 (DML) ステートメント (**INSERT**、**UPDATE**、**DELETE**、および **TRUNCATE**) を監査します。
- **select** — **SELECT** ステートメントを監査します。
- **set** — **SET** ステートメントを監査します。
- **rollback** — **ROLLBACK** ステートメントを監査します。
- **error** — 発生したすべてのエラーメッセージをログに記録します。それ以外の場合、**all** が使用されている場合を除き、上記の他のパラメーター値に関連する SQL ステートメントで発生したエラーに関するメッセージはログに記録されません。

- { select | update | delete | insert }@groupname —特定の DML ステートメント (SELECT、UPDATE、DELETE、および INSERT) のオブジェクトを選択的に監査し、特定のテーブルの (@) グループを除外します。詳細については、[Object auditing](#) を参照してください。

DDL ステートメントと DCL ステートメント は、監査する特定の DDL または DCL ステートメントを選択するための追加のパラメーター値を説明します。**DML ステートメント** は、監査用に特定の DML ステートメントを選択するための追加のパラメーター値について説明します。

edb_audit_statement パラメータにサポートされていない値が含まれている場合、データベースサーバに設定を適用するときにエラーが発生します。create materialized vw がエラーになる次の例のようなエラーについては、データベースサーバのログファイルを参照してください。正しい値は create materialized view です。

```
2017-07-16 11:20:39 EDT LOG:  invalid value for parameter
"edb_audit_statement": "create materialized vw, create sequence, grant"
2017-07-16 11:20:39 EDT FATAL:  configuration file "/var/lib/edb/as14/data/
postgresql.conf" contains errors
```

DDL ステートメントと DCL ステートメント

DDL および DCL ステートメントを監査する場合、edb_audit_statement パラメーターに含めることができる値には、次の一般的なルールが適用されます。

- edb_audit_statement パラメーターに ddl または all が含まれる場合、すべての DDL ステートメントが監査されます。さらに、DCL ステートメント GRANT および REVOKE が監査されます。
- edb_audit_statement が none に設定されている場合、DDL ステートメントも DCL ステートメントも監査されません。
- edb_audit_statement パラメーターに値の組み合わせを含めることにより、監査のために特定の種類の DDL および DCL ステートメントを選択できます。

次の構文を使用して、DDL ステートメントの edb_audit_statement パラメーター値を指定します。

```
{ create | alter | drop } [ <object_type> ]
```

object_type は次のいずれかです。

- ACCESS METHOD
- AGGREGATE
- CAST
- COLLATION
- CONVERSION
- DATABASE
- EVENT TRIGGER
- EXTENSION

- FOREIGN TABLE
- FUNCTION
- INDEX
- LANGUAGE
- LARGE OBJECT
- MATERIALIZED VIEW
- OPERATOR
- OPERATOR CLASS
- OPERATOR FAMILY
- POLICY
- PUBLICATION
- ROLE
- RULE
- SCHEMA
- SEQUENCE
- SERVER
- SUBSCRIPTION
- TABLE
- TABLESPACE
- TEXT SEARCH CONFIGURATION
- TEXT SEARCH DICTIONARY
- TEXT SEARCH PARSER
- TEXT SEARCH TEMPLATE
- TRANSFORM
- TRIGGER
- TYPE
- USER MAPPING
- VIEW

SQL コマンドで使用するオブジェクトタイプの説明は、 [PostgreSQL core documentation](#) にあります。

パラメーター値から `object_type` が省略された場合、指定されたすべてのコマンドステートメント (`create`、`alter` または `drop`) が監査されます。

次の構文を使用して、DCL ステートメントの `edb_audit_statement` パラメーター値を指定します。

```
{ grant | revoke }
```

例

この例では、`edb_audit_connect` と `edb_audit_statement` は次の非デフォルト値で設定されます。

```
logging_collector = on
edb_audit_connect = all
edb_audit_statement = create, alter, error
```

したがって、**CREATE** および **ALTER** コマンドによって呼び出された SQL ステートメントのみが監査されます。エラーメッセージも監査ログに含まれます。

発生するデータベースセッションは次のとおりです。

```
$ psql edb enterprisedb
Password for user enterprisedb:
psql.bin (14.0.0)
Type "help" for help.

edb=# SHOW edb_audit_connect;
__OUTPUT__
  edb_audit_connect
-----
all
(1 row)
```

```
edb=# SHOW edb_audit_statement;
__OUTPUT__
  edb_audit_statement
-----
create, alter, error
(1 row)
```

```
edb=# CREATE ROLE adminuser;
CREATE ROLE
edb=# ALTER ROLE adminuser WITH LOGIN, SUPERUSER, PASSWORD password;
ERROR:  syntax error at or near ","
LINE 1: ALTER ROLE adminuser WITH LOGIN, SUPERUSER, PASSWORD passwo...
                                     ^

edb=# ALTER ROLE adminuser WITH LOGIN SUPERUSER PASSWORD password;
ALTER ROLE
edb=# CREATE DATABASE auditdb;
CREATE DATABASE
edb=# ALTER DATABASE auditdb OWNER TO adminuser;
```

(次のページに続く)

(前のページからの続き)

```

ALTER DATABASE
edb=# \c auditdb adminuser
Password for user adminuser:
You are now connected to database "auditdb" as user "adminuser".
auditdb=# CREATE SCHEMA edb;
CREATE SCHEMA
auditdb=# SET search_path TO edb;
SET
auditdb=# CREATE TABLE department (
auditdb(# deptno          NUMBER(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
auditdb(#  dname          VARCHAR2(14) CONSTRAINT dept_dname_uq UNIQUE,
auditdb(#  loc            VARCHAR2(13)
auditdb(# );
CREATE TABLE
auditdb=# DROP TABLE department;
DROP TABLE
auditdb=# CREATE TABLE dept (
auditdb(# deptno          NUMBER(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
auditdb(#  dname          VARCHAR2(14) CONSTRAINT dept_dname_uq UNIQUE,
auditdb(#  loc            VARCHAR2(13)
auditdb(# );
CREATE TABLE

```

生成される監査ログファイルには次が含まれます。(各監査ログエントリは分割され、複数行に表示されます。見やすくするために監査ログエントリの間に空白行が挿入されました。)

```

2020-05-25 12:32:22.799 IST,"enterprisedb","edb",72518,"[local]",5ecb6d7e.
11b46,1,"authentication",2020-05-25 12:32:22 IST,4/19,0,AUDIT,000000,
"connection authorized:user=enterprisedb database=edb",,,,,,,,,,"client
backend",,,,,,"connect"

2020-05-25 12:34:05.843 IST,"enterprisedb","edb",72518,"[local]",5ecb6d7e.
11b46,2,"idle",2020-05-25 12:32:22 IST,4/23,0,AUDIT,000000,"statement: CREATE
ROLE adminuser;",,,,,,,,,,"psql","client backend","CREATE ROLE","create"

2020-05-25 12:34:16.617 IST,"enterprisedb","edb",72518,"[local]",5ecb6d7e.
11b46,3,"idle",2020-05-25 12:32:22 IST,4/24,0,ERROR,42601,"syntax error at or
near """,,,,,,"ALTER ROLE adminuser WITH LOGIN, SUPERUSER, PASSWORD
password;",32,,,"psql","client backend",,,,,,"error"

2020-05-25 12:34:29.954 IST,"enterprisedb","edb",72518,"[local]",5ecb6d7e.
11b46,4,"idle",2020-05-25 12:32:22 IST,4/25,0,AUDIT,000000,"statement: ALTER
ROLE adminuser WITH LOGIN SUPERUSER PASSWORD password;",,,,,,,,,,"psql",
"client backend","ALTER ROLE","alter"

```

(次のページに続く)

(前のページからの続き)

```

2020-05-25 12:34:40.114 IST,"enterprisedb","edb",72518,"[local]",5ecb6d7e.
11b46,5,"idle",2020-05-25 12:32:22 IST,4/26,0,AUDIT,000000,"statement: CREATE
DATABASE auditdb;",,,,,,,,,,"psql","client backend","CREATE DATABASE","create"

2020-05-25 12:34:50.591 IST,"enterprisedb","edb",72518,"[local]",5ecb6d7e.
11b46,6,"idle",2020-05-25 12:32:22 IST,4/27,0,AUDIT,000000,"statement: ALTER
DATABASE auditdb OWNER TO adminuser;",,,,,,,,,,"psql","client backend","ALTER
DATABASE","alter"

2020-05-25 12:35:01.554 IST,"adminuser","auditdb",75531,"[local]",5ecb6e1d.
1270b,1,"authentication",2020-05-25 12:35:01 IST,5/11,0,AUDIT,000000,
"connection authorized:user=adminuser database=auditdb",,,,,,,,,,"client
backend","", "connect"

2020-05-25 12:35:12.931 IST,"adminuser","auditdb",75531,"[local]",5ecb6e1d.
1270b,2,"idle",2020-05-25 12:35:01 IST,5/13,0,AUDIT,000000,"statement: CREATE
SCHEMA edb;",,,,,,,,,,"psql","client backend","CREATE SCHEMA","create"

2020-05-25 12:37:18.547 IST,"adminuser","auditdb",75531,"[local]",5ecb6e1d.
1270b,3,"idle",2020-05-25 12:35:01 IST,5/15,0,AUDIT,000000,"statement: CREATE
TABLE department
(
    deptno    NUMBER(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
    dname     VARCHAR2(14) CONSTRAINT dept_dname_uq UNIQUE,
    loc       VARCHAR2(13)
);",,,,,,,,,,"psql","client backend","CREATE TABLE","create"

2020-05-25 12:39:09.065 IST,"adminuser","auditdb",75531,"[local]",5ecb6e1d.
1270b,4,"idle",2020-05-25 12:35:01 IST,5/17,0,AUDIT,000000,"statement: CREATE
TABLE dept (
    deptno    NUMBER(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
    dname     VARCHAR2(14) CONSTRAINT dept_dname_uq UNIQUE,
    loc       VARCHAR2(13)
);",,,,,,,,,,"psql","client backend","CREATE TABLE","create"

```

adminuser ロールおよび auditdb データベースの CREATE および ALTER ステートメントが監査されます。error は edb_audit_statement パラメーターに含まれているため、ALTER ROLE adminuser ステートメントのエラーもログに記録されます。

同様に、スキーマ edb とテーブル department および dept の CREATE ステートメントが監査されます。

ddl、all、drop などの DROP ステートメントが正常に処理されたことを監査する edb_audit_statement 設定はありません。したがって、DROP TABLE department ステートメントは監査ログにありません。

この例では、edb_audit_connect と edb_audit_statement は次の非デフォルト値で設定されます。

```
logging_collector = on
edb_audit_connect = all
edb_audit_statement = create view,create materialized view,create sequence,grant
```

したがって、CREATE VIEW、CREATE MATERIALIZED VIEW、CREATE SEQUENCE、および GRANT コマンドによって呼び出された SQL ステートメントのみが監査されます。

発生するデータベースセッションは次のとおりです。

```
$ psql auditdb adminuser
Password for user adminuser:
psql.bin (14.0.0)
Type "help" for help.

auditdb=# SHOW edb_audit_connect;
__OUTPUT__
    edb_audit_connect
-----
    all
(1 row)
```

```
auditdb=# SHOW edb_audit_statement;
__OUTPUT__
    edb_audit_statement
-----
    create view,create materialized view,create sequence,grant
(1 row)
```

```
auditdb=# SET search_path TO edb;
SET
auditdb=# CREATE TABLE emp (
auditdb(#   empno           NUMBER(4) NOT NULL CONSTRAINT emp_pk PRIMARY KEY,
auditdb(#   ename           VARCHAR2(10),
auditdb(#   job             VARCHAR2(9),
auditdb(#   mgr             NUMBER(4),
auditdb(#   hiredate        DATE,
auditdb(#   sal             NUMBER(7,2) CONSTRAINT emp_sal_ck CHECK (sal > 0),
auditdb(#   comm            NUMBER(7,2),
auditdb(#   deptno          NUMBER(2) CONSTRAINT emp_ref_dept_fk
auditdb(#                               REFERENCES dept(deptno)
auditdb(# );
CREATE TABLE
auditdb=# CREATE VIEW salesemp AS
auditdb=#   SELECT empno, ename, hiredate, sal, comm FROM emp WHERE job =
SALESMAN;
CREATE VIEW
```

(次のページに続く)

(前のページからの続き)

```

auditdb=# CREATE MATERIALIZED VIEW managers AS
auditdb=# SELECT empno, ename, hiredate, sal, comm FROM emp WHERE job =
MANAGER;
SELECT 0
auditdb=# CREATE SEQUENCE next_empno START WITH 8000 INCREMENT BY 1;
CREATE SEQUENCE
auditdb=# GRANT ALL ON dept TO PUBLIC;
GRANT
auditdb=# GRANT ALL ON emp TO PUBLIC;
GRANT

```

生成される監査ログファイルには次が含まれます。(各監査ログエントリは分割され、複数行に表示されます。見やすくするために監査ログエントリの間に空白行が挿入されました。)

```

2020-05-25 14:05:29.163 IST,"adminuser","auditdb",40810,"[local]",5ecb8351.
9f6a,1,"authentication",2020-05-25 14:05:29 IST,4/28,0,AUDIT,000000,
"connection authorized:user=adminuser database=auditdb",,,,,,,,,,"client
backend",,,,,,"connect"

2020-05-25 14:12:06.318 IST,"adminuser","auditdb",40810,"[local]",5ecb8351.
9f6a,2,"idle",2020-05-25 14:05:29 IST,4/34,0,AUDIT,000000,"statement: CREATE
VIEW salesemp AS SELECT empno, ename, hiredate, sal, comm FROM emp WHERE job
= SALESMAN;",,,,,,,,,,"psql","client backend","CREATE VIEW","create"

2020-05-25 14:13:26.657 IST,"adminuser","auditdb",40810,"[local]",5ecb8351.
9f6a,3,"idle",2020-05-25 14:05:29 IST,4/36,0,AUDIT,000000,"statement: CREATE
MATERIALIZED VIEW managers AS SELECT empno, ename, hiredate, sal, comm FROM
emp WHERE job = MANAGER;",,,,,,,,,,"psql","client backend","CREATE
MATERIALIZED VIEW","create"

2020-05-25 14:13:38.928 IST,"adminuser","auditdb",40810,"[local]",5ecb8351.
9f6a,4,"idle",2020-05-25 14:05:29 IST,4/37,0,AUDIT,000000,"statement: CREATE
SEQUENCE next_empno START WITH 8000 INCREMENT BY 1;",,,,,,,,,,"psql","client
backend","CREATE SEQUENCE","create"

2020-05-25 14:13:51.434 IST,"adminuser","auditdb",40810,"[local]",5ecb8351.
9f6a,5,"idle",2020-05-25 14:05:29 IST,4/38,0,AUDIT,000000,"statement: GRANT
ALL ON dept TO PUBLIC;",,,,,,,,,,"psql","client backend","GRANT","grant"

2020-05-25 14:14:03.737 IST,"adminuser","auditdb",40810,"[local]",5ecb8351.
9f6a,6,"idle",2020-05-25 14:05:29 IST,4/39,0,AUDIT,000000,"statement: GRANT
ALL ON emp TO PUBLIC;",,,,,,,,,,"psql","client backend","GRANT","grant"

```

CREATE VIEW および CREATE MATERIALIZED VIEW ステートメントが監査されます。create、create table、ddl、およびallの値がedb_audit_statementパラメーターに含まれていないため、前のCREATE TABLE emp ステートメントは監査されません。

CREATE SEQUENCE および GRANT ステートメントは、これらの値が `edb_audit_statement` パラメーターに含まれるために監査されます。

この例では、`edb_audit_connect` と `edb_audit_statement` は次の非デフォルト値で設定されます。

```
logging_collector = on
edb_audit_connect = all
edb_audit_statement = none
```

ADMIN または SYSDBA として接続するユーザーのセッションを完全に監査できます。管理ユーザーは、データベース `auditdb` に `ADMINUSER` として接続されています。次の `ALTER USER` コマンドは、`ADMINUSER` を監査することを指定しています。

```
ALTER USER adminuser SET edb_audit_statement = "all";
```

`edb_audit_statement` パラメーターを `all` に設定すると、管理ユーザーのすべての SQL ステートメントを監査できます。

発生するデータベースセッションは次のとおりです。

```
$ psql auditdb adminuser
Password for user adminuser:
psql.bin (14.0.0)
Type "help" for help.

auditdb=# SHOW edb_audit_connect;
__OUTPUT__
  edb_audit_connect
- - - - -
    all
(1 row)
```

```
auditdb=# SHOW edb_audit_statement;
__OUTPUT__
  edb_audit_statement
- - - - -
    all
(1 row)
```

```
auditdb=# SET search_path TO edb;
SET
auditdb=> CREATE TABLE dept
auditdb->      (deptno NUMBER(2),
auditdb(>      dname VARCHAR2(14),
auditdb(>      loc VARCHAR2(13) );
CREATE TABLE
auditdb=> INSERT INTO dept VALUES (10, ACCOUNTING, NEW YORK);
```

(次のページに続く)

(前のページからの続き)

```

INSERT 0 1
auditdb=> INSERT INTO dept VALUES (20, RESEARCH, DALLAS);
INSERT 0 1
auditdb=> INSERT INTO dept VALUES (30, SALES, CHICAGO);
INSERT 0 1
auditdb=> UPDATE dept SET loc = BEDFORD WHERE deptno = 20;
UPDATE 1
auditdb=> SELECT * FROM dept;
__OUTPUT__
 deptno |  dname   |   loc
-----+-----+-----
      10 | ACCOUNTING | NEW YORK
      30 | SALES      | CHICAGO
      20 | RESEARCH   | BEDFORD
(3 rows)

```

```

auditdb=> DELETE FROM emp WHERE deptno = 40;
ERROR:  relation "emp" does not exist
LINE 1: DELETE FROM emp WHERE deptno = 40;
              ^
auditdb=> DELETE FROM dept WHERE deptno = 10;
DELETE 1
auditdb=> SELECT * FROM dept;
__OUTPUT__
 deptno |  dname   |   loc
-----+-----+-----
      30 | SALES      | CHICAGO
      20 | RESEARCH   | BEDFORD
(2 rows)

```

生成される監査ログファイルには次が含まれます。(各監査ログエントリは分割され、複数行に表示されま
す。見やすくするために監査ログエントリの間に空白行が挿入されました。)

```

2021-06-23 06:06:59.027 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,1,"authentication",2021-06-23 06:06:59 IST,4/19,0,AUDIT,000000,"connection_
→authorized:
user=adminuser database=auditdb",,,,,,,,,,"","client backend",,"","connect"

2021-06-23 06:07:33.192 IST,"adminuser","auditdb",66316,"[local]",60daab0c.
1030c,2,"idle",2021-06-23 06:07:33 IST,4/16,0,AUDIT,000000,"statement: SHOW edb_audit_
→connect;
",,,,,,,,,,"psql","client backend",,"SHOW",,"","sql statement"

2021-06-23 06:08:12.474 IST,"adminuser","auditdb",66316,"[local]",60daab0c.
1030c,3,"idle",2021-06-23 06:08:12 IST,4/17,0,AUDIT,000000,"statement: SHOW edb_audit_
→statement;

```

(次のページに続く)

(前のページからの続き)

```

",,,,,,,,,,"psql","client backend",,"SHOW","",,"sql statement"

2021-06-23 06:08:20.519 IST,"adminuser","auditdb",66922,"[local]",60dab036.
1056a,4,"idle",2021-06-23 06:08:20 IST,4/15,0,AUDIT,000000,"statement: SET search_path_
→TO edb;
",,,,,,,,,,"psql","client backend",,"SET","",,"set"

2021-06-23 06:09:27.613 IST,"adminuser","auditdb",60218,"[local]",60dab117.
10602,5,"idle",2021-06-23 06:09:59 IST,4/21,0,AUDIT,000000,"statement: CREATE TABLE_
→dept
      (deptno NUMBER(2),
       dname VARCHAR2(14),
       loc VARCHAR2(13) );",,,,,,,,,,"psql","client backend",,"CREATE TABLE","",
→"create"

2021-06-23 06:09:39.238 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,6,"idle",2021-06-23 06:09:29 IST,4/22,0,AUDIT,000000,"statement: INSERT INTO
dept VALUES (10, ACCOUNTING, NEW YORK);",,,,,,,,,,"psql","client backend",,"INSERT","",
→"insert"

2021-06-23 06:09:39.242 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,7,"idle",2021-06-23 06:09:29 IST,4/23,0,AUDIT,000000,"statement: INSERT INTO
dept VALUES (20, RESEARCH, DALLAS);",,,,,,,,,,"psql","client backend",,"INSERT","",
→"insert"

2021-06-23 06:09:39.247 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,8,"idle",2021-06-23 06:08:35 IST,4/24,0,AUDIT,000000,"statement: INSERT INTO
dept VALUES (30, SALES, CHICAGO);",,,,,,,,,,"psql","client backend",,"INSERT","",
→"insert"

2021-06-23 06:10:04.849 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,9,"idle",2021-06-23 06:08:59 IST,4/25,0,AUDIT,000000,"statement: UPDATE dept SET_
→loc = BEDFORD
WHERE deptno = 20;",,,,,,,,,,"psql","client backend",,"UPDATE","",,"update"

2021-06-23 06:10:16.045 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,10,"idle",2021-06-23 06:08:59 IST,4/26,0,AUDIT,000000,"statement: SELECT * FROM_
→dept;",,,,,,,,,
"psql","client backend",,"SELECT","",,"select"

2021-06-23 06:10:40.593 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,11,"idle",2021-06-23 06:08:59 IST,4/27,0,AUDIT,000000,"statement: DELETE FROM emp_
→WHERE deptno = 40;
",,,,,,,,,,"psql","client backend",,"DELETE","",,"delete"

```

(次のページに続く)

(前のページからの続き)

```

2021-06-23 06:10:40.594 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,12,"DELETE",2021-06-23 06:08:59 IST,4/27,0,ERROR,42P01,"relation ""emp"" does
↳not exist",,,,,,
"DELETE FROM emp WHERE deptno = 40;",13,,,"psql","client backend",,"DELETE","",,"error"

2021-06-23 06:11:02.563 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,13,"idle",2021-06-23 06:08:59 IST,4/28,0,AUDIT,00000,"statement: DELETE FROM
↳dept WHERE deptno = 10;
",,,,,,,,,,"psql","client backend",,"DELETE","",,"delete"

2021-06-23 06:11:14.585 IST,"adminuser","auditdb",60218,"[local]",60d3083b.
eb3a,14,"idle",2021-06-23 06:08:59 IST,4/29,0,AUDIT,00000,"statement: SELECT * FROM
↳dept;",,,,,,,,,,
"psql","client backend",,"SELECT","",,"select"

```

この例では、`edb_audit_connect` と `edb_audit_statement` は次の非デフォルト値で設定されます。

```

logging_collector = on
edb_audit_connect = all
edb_audit_statement = all

```

ユーザー `carol` の監査セッションは、データベース管理者が `ALTER USER` コマンドを使用して完全にブロックできます。

```
ALTER USER carol SET edb_audit_statement = "none";
```

注釈: データベース管理者は、`ALTER USER` コマンドを指定し、`edb_audit_statement` パラメーターを任意の値に設定することにより、特定のユーザーに SQL ステートメントの監査を許可できます。発生するデータベースセッションは次のとおりです。

```

$ psql auditdb carol
Password for user carol:
psql.bin (14.0.0)
Type "help" for help.

auditdb=# SHOW edb_audit_connect;
__OUTPUT__
edb_audit_connect
- - - - -
all
(1 row)

```

```
auditdb=# SHOW edb_audit_statement;
```

(次のページに続く)

(前のページからの続き)

```
__OUTPUT__
edb_audit_statement
-----
none
(1 row)
```

```
auditdb=# SET search_path TO edb;
SET
auditdb=> CREATE TABLE salgrade
auditdb->      (grade NUMBER,
auditdb(>      losal NUMBER,
auditdb(>      hisal NUMBER);
CREATE TABLE
INSERT INTO salgrade VALUES (1, 700, 1200);
INSERT INTO salgrade VALUES (2, 1201, 1400);
INSERT INTO salgrade VALUES (3, 1401, 2000);
```

生成される監査ログファイルには、接続認証情報のみが含まれます。edb_audit_statement パラメーターを none に設定すると、carol の SQL ステートメントの監査が許可されないため、監査ログは生成されません。

(各監査ログエントリは分割され、複数行に表示されます。見やすくするために監査ログエントリの間に空白行が挿入されました。)

```
2021-06-29 02:27:26.240 IST,"carol","auditdb",68072,"[local]",60dabd4e.
109e8,1,"authentication",2021-06-29 02:27:26 IST,4/13,0,AUDIT,00000,"connection_
→authorized: user=carol
database=auditdb",,,,,,,,,,"","client backend",,,,,,"connect"
```

DML ステートメント

DML ステートメントを監査する場合、edb_audit_statement パラメータに含めることができる値には、次の一般的なルールが適用されます。

- edb_audit_statement パラメーターに dml または all が含まれる場合、すべての DML ステートメントが監査されます。
- edb_audit_statement が none に設定されている場合、DML ステートメントは監査されません。
- edb_audit_statement パラメーターに値の組み合わせを含めることにより、監査用に特定の種類の DML ステートメントを選択できます。

次の構文を使用して、SQL INSERT、UPDATE、DELETE、または TRUNCATE ステートメントの edb_audit_statement パラメーター値を指定します。

```
{ insert | update | delete | truncate }
```

例

この例では、`edb_audit_connect` と `edb_audit_statement` は次の非デフォルト値で設定されます。

```
logging_collector = on
edb_audit_connect = all
edb_audit_statement = UPDATE, DELETE, error
```

したがって、UPDATE および DELETE コマンドによって呼び出された SQL ステートメントのみが監査されます。UPDATE および DELETE コマンドに関連しないエラーを含む、すべてのエラーも監査ログに含まれます。

発生するデータベースセッションは次のとおりです。

```
$ psql auditdb adminuser
Password for user adminuser:
psql.bin (14.0.0)
Type "help" for help.

auditdb=# SHOW edb_audit_connect;
__OUTPUT__
edb_audit_connect
- - - - -
all
(1 row)
```

```
auditdb=# SHOW edb_audit_statement;
__OUTPUT__
edb_audit_statement
- - - - -
UPDATE, DELETE, error
(1 row)
```

```
auditdb=# SET search_path TO edb;
SET
auditdb=# INSERT INTO dept VALUES (10,ACCOUNTING,NEW YORK);
INSERT 0 1
auditdb=# INSERT INTO dept VALUES (20,RESEARCH,DALLAS);
INSERT 0 1
auditdb=# INSERT INTO dept VALUES (30,SALES,CHICAGO);
INSERT 0 1
auditdb=# INSERT INTO dept VALUES (40,OPERATIONS,BOSTON);
INSERT 0 1
auditdb=# INSERT INTO emp VALUES (7369,SMITH,CLERK,7902,17-DEC-80,800,NULL,20);
INSERT 0 1
auditdb=# INSERT INTO emp VALUES (7499,ALLEN,SALESMAN,7698,20-FEB-81,1600,300,30);
INSERT 0 1
```

(次のページに続く)

(前のページからの続き)

```

auditdb=# INSERT INTO emp VALUES (7521,WARD,SALESMAN,7698,22-FEB-81,1250,500,30);
INSERT 0 1
      .
      .
      .
auditdb=# INSERT INTO emp VALUES (7934,MILLER,CLERK,7782,23-JAN-82,1300,NULL,10);
INSERT 0 1
auditdb=# UPDATE dept SET loc = BEDFORD WHERE deptno = 40;
UPDATE 1
auditdb=# SELECT * FROM dept;
__OUTPUT__
 deptno |  dname   |   loc
-----+-----+-----
    10 | ACCOUNTING | NEW YORK
    20 | RESEARCH  | DALLAS
    30 | SALES     | CHICAGO
    40 | OPERATIONS | BEDFORD
(4 rows)

```

```

auditdb=# DELETE FROM emp WHERE deptno = 10;
DELETE 3
auditdb=# TRUNCATE employee;
ERROR:  relation "employee" does not exist
auditdb=# TRUNCATE emp;
TRUNCATE TABLE
auditdb=# \q

```

生成される監査ログファイルには次が含まれます。(各監査ログエントリは分割され、複数行に表示されます。見やすくするために監査ログエントリの間に空白行が挿入されました。)

```

2020-05-25 15:17:16.392 IST,"adminuser","auditdb",123420,"[local]",5ecb9424.
1e21c,1,"authentication",2020-05-25 15:17:16 IST,5/3,0,AUDIT,000000,
"connection authorized:user=adminuser database=auditdb",,,,,,,,,,"client
backend",,,,,,connect"

2020-05-25 15:19:18.066 IST,"adminuser","auditdb",123420,"[local]",5ecb9424.
1e21c,2,"idle",2020-05-25 15:17:16 IST,5/14,0,AUDIT,000000,"statement: UPDATE
dept SET loc = BEDFORD WHERE deptno = 40;",,,,,,,,,,"psql","client backend",
"UPDATE","update"

2020-05-25 15:19:38.524 IST,"adminuser","auditdb",123420,"[local]",5ecb9424.
1e21c,3,"idle",2020-05-25 15:17:16 IST,5/16,0,AUDIT,000000,"statement: DELETE
FROM emp WHERE deptno = 10;",,,,,,,,,,"psql","client backend","DELETE","delete"

2020-05-25 15:19:48.149 IST,"adminuser","auditdb",123420,"[local]",5ecb9424.

```

(次のページに続く)

(前のページからの続き)

```
1e21c,4,"TRUNCATE TABLE",2020-05-25 15:17:16 IST,5/17,0,ERROR,42P01,"relation
""employee""does not exist",,,,,,"TRUNCATE employee;",,,,psql,"client backend
","TRUNCATE TABLE","error"
```

UPDATE dept および DELETE FROM emp ステートメントが監査されます。insert、dml、または all の値が edb_audit_statement パラメーターに含まれていないため、以前の INSERT ステートメントはすべて監査されません。

select および all は edb_audit_statement パラメーターに含まれていないため、SELECT * FROM dept ステートメントは監査されません。

error は edb_audit_statement パラメーターで指定されていますが、truncate 値ではないため、TRUNCATE employee ステートメントのエラーが監査ファイルに記録されます。しかし、成功した TRUNCATE emp ステートメントはそうではありません。

3.5.3 Enabling audit logging

すべての接続、切断、DDL ステートメント、DCL ステートメント、DML ステートメント、およびエラーが発生したステートメントをログに記録するように EDB Postgres Advanced Server を構成できます。

- edb_audit パラメーターを xml または csv に設定して、監査を有効にします。
- パラメーター edb_audit_rotation_day を目的の値に設定して、新しいファイルが作成されるファイルのローテーション日を設定します。
- すべての接続を監査するには、パラメーター edb_audit_connect を all に設定します。
- すべての切断を監査するには、パラメーター edb_audit_disconnect を all に設定します。
- DDL、DCL、DML およびその他のステートメントを監査するには、
Selecting SQL statements to audit の指示に従ってパラメーター edb_audit_statement を設定します。

構成ファイルで edb_audit_statement パラメーターを設定すると、データベースクラスター全体に影響します。

監査されるステートメントの種類をさらに絞り込むことができます。このタイプは、使用中のデータベースとセッションを実行しているデータベースロールに従って、edb_audit_statement パラメーターによって制御されます。

- ALTER DATABASE dbname SET edb_audit_statement コマンドで、指定したデータベースの属性として edb_audit_statement パラメーターを設定できます。この設定は、データベース dbname への接続時に実行されるステートメントの構成ファイルの edb_audit_statement パラメーターをオーバーライドします。

- ALTER ROLE rolename SET edb_audit_statement コマンドで、指定したロールの属性として edb_audit_statement パラメーターを設定できます。この設定は、構成ファイルの edb_audit_statement パラメーター、および指定されたロールが現在のセッションを実行しているときに ALTER DATABASE コマンドによってデータベースに割り当てられた設定をオーバーライドします。

・ `ALTER ROLE rolename IN DATABASE dbname SET edb_audit_statement` コマンドで指定したデータベースを利用する場合、指定したロールの属性として `edb_audit_statement` パラメーターを設定できます。この設定は、構成ファイルの `edb_audit_statement` パラメーターをオーバーライドします。また、`ALTER DATABASE` コマンドによってデータベースに割り当てられた設定、および `IN DATABASE` 句なしで `ALTER ROLE` コマンドでロールに割り当てられた設定をオーバーライドします。

例

以下は、この手法の例です。

`postgresql.conf` ファイルに示すように、データベースクラスターは `edb_audit_statement` を `all` に設定して確立されています。

```
logging_collector = on
edb_audit_statement = all      # Statement type to be audited:
                                # none, dml, insert, update, delete, truncate,
                                # select, error, rollback, ddl, create, drop,
                                # alter, grant, revoke, set, all
                                # {select | update | delete | insert}@groupname
```

`edb_audit_statement` パラメーターを次の設定で使用して、データベースとロールが確立されます。

- `ddl`、`insert`、`update`、および `delete` を備えたデータベース `auditdb`
- `select`、`truncate`、および `set` を持つロール `admin`
- `create table`、`insert`、および `update` を持つデータベース `auditdb` のロール `admin`

次に、データベースとロールの作成と変更を示します。

```
$ psql edb enterprisedb
Password for user enterprisedb:
psql.bin (14.0.0)
Type "help" for help.

edb=# SHOW edb_audit_statement;
__OUTPUT__
edb_audit_statement
-----
all
(1 row)
```

```
edb=# CREATE DATABASE auditdb;
CREATE DATABASE
edb=# ALTER DATABASE auditdb SET edb_audit_statement TO ddl, insert, update, delete;
ALTER DATABASE
edb=# CREATE ROLE admin WITH LOGIN SUPERUSER PASSWORD password;
CREATE ROLE
```

(次のページに続く)

(前のページからの続き)

```

edb=# ALTER ROLE admin SET edb_audit_statement TO select, truncate;
ALTER ROLE
edb=# ALTER ROLE admin IN DATABASE auditdb SET edb_audit_statement TO create table,
↪insert, update;
ALTER ROLE

```

次に、3つの場合について行われた変更と結果の監査ログファイルを示します。

ケース 1：ロール `enterprisedb` によってデータベース `auditdb` に加えられた変更

`ddl`、`insert`、`update`、および `delete` ステートメントのみが監査されます。

```

$ psql auditdb enterprisedb
Password for user enterprisedb:
psql.bin (14.0.0)
Type "help" for help.

auditdb=# SHOW edb_audit_statement;
__OUTPUT__
      edb_audit_statement
-----
ddl, insert, update, delete
(1 row)

```

```

auditdb=# CREATE TABLE audit_tbl (f1 INTEGER PRIMARY KEY, f2 TEXT);
CREATE TABLE
auditdb=# INSERT INTO audit_tbl VALUES (1, Row 1);
INSERT 0 1
auditdb=# UPDATE audit_tbl SET f2 = Row A WHERE f1 = 1;
UPDATE 1
auditdb=# SELECT * FROM audit_tbl;
<== Should not be audited
__OUTPUT__
 f1 | f2
----+-----
  1 | Row A
(1 row)

```

```

auditdb=# TRUNCATE audit_tbl;
<== Should not be audited
TRUNCATE TABLE

```

次の監査ログファイルは、`CREATE TABLE`、`INSERT INTO audit_tbl`、および `UPDATE audit_tbl` ステートメントのエントリのみを示しています。`SELECT * FROM audit_tbl` および `TRUNCATE audit_tbl` ステートメントは監査されませんでした。(各監査ログエントリは分割され、複数行に表示されます。見やすくするために監査ログエントリの間に空白行が挿入されました。)

```

2020-05-25 15:59:12.332 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.a36d,2,
"idle",2020-05-25 15:59:02 IST,5/7,0,AUDIT,000000,"statement: CREATE TABLE
audit_tbl(f1 INTEGER PRIMARY KEY, f2 TEXT);",,,,,,,,,,"psql","client backend","CREATE_
→TABLE","", "create"

2020-05-25 15:59:22.419 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.a36d,3,
"idle",2020-05-25 15:59:02 IST,5/8,0,AUDIT,000000,"statement: INSERT INTO
audit_tbl VALUES (1, Row 1);",,,,,,,,,,"psql","client backend","INSERT","", "insert"

2020-05-25 15:59:32.180 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.a36d,4,
"idle",2020-05-25 15:59:02 IST,5/9,0,AUDIT,000000,"statement: UPDATE
audit_tbl SET f2 = Row A WHERE f1 = 1;",,,,,,,,,,"psql","client backend","UPDATE","",
→"update"

```

ケース 2：ロール admin によってデータベース edb に加えられた変更

select、truncate、および set ステートメントのみが監査されます。

```

$ psql edb admin
Password for user admin:
psql.bin (14.0.0)
Type "help" for help.

edb=# SHOW edb_audit_statement;
__OUTPUT__
  edb_audit_statement
-----
select, truncate, set
(1 row)

```

```

edb=# SET default_with_rowids TO TRUE;
SET

edb=# CREATE TABLE edb_tbl (f1 INTEGER PRIMARY KEY, f2 TEXT); <== Should not be_
→audited
CREATE TABLE
edb=# INSERT INTO edb_tbl VALUES (1, Row 1); <== Should not be audited
INSERT 0 1
edb=# SELECT * FROM edb_tbl;
__OUTPUT__
 f1 | f2
----+-----
  1 | Row 1
(1 row)

```

```
edb=# TRUNCATE edb_tbl;
TRUNCATE TABLE
```

監査ログファイルの続きは次のように表示されます。2 番目のケースを表す最後の 2 つのエントリには、SET default_with_rowids TO TRUE、SELECT * FROM edb_tbl、および TRUNCATE edb_tbl ステートメントのみが表示されます。CREATE TABLE edb_tbl および INSERT INTO edb_tbl ステートメントは監査されませんでした。

```
2020-05-25 15:59:12.332 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.a36d,2,
"idle",2020-05-25 15:59:02 IST,5/7,0,AUDIT,000000,"statement: CREATE TABLE
audit_tbl(f1 INTEGER PRIMARY KEY, f2 TEXT);",,,,,,,,,,"psql","client backend","CREATE_
→TABLE",,,,,,"create"

2020-05-25 15:59:22.419 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.a36d,3,
"idle",2020-05-25 15:59:02 IST,5/8,0,AUDIT,000000,"statement: INSERT INTO
audit_tbl VALUES (1, Row 1);",,,,,,,,,,"psql","client backend","INSERT",,,,,,"insert"

2020-05-25 15:59:32.180 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.a36d,4,
"idle",2020-05-25 15:59:02 IST,5/9,0,AUDIT,000000,"statement: UPDATE
audit_tbl SET f2 = Row A WHERE f1 = 1;",,,,,,,,,,"psql","client backend","UPDATE",,,,,
→"update"

2021-02-18 08:04:57.434 IST,"admin","edb",3182,"[local]",602e65dc.c6e,1,
"idle",2021-02-18 08:04:28 IST,4/22,0,AUDIT,000000,"statement: SET default_with_rowids_
→TO TRUE;
",,,,,,,,,,"psql","client backend","SET",,,,,,"set"

2021-02-18 08:06:01.662 IST,"admin","edb",3182,"[local]",602e65dc.c6e,2,
"idle",2021-02-18 08:04:28 IST,4/27,0,AUDIT,000000,"statement: SELECT * FROM edb_tbl;
",,,,,,,,,,"psql","client backend","SELECT",,,,,,"select"

2021-02-18 08:06:11.125 IST,"admin","edb",3182,"[local]",602e65dc.c6e,3,
"idle",2021-02-18 08:04:28 IST,4/28,0,AUDIT,000000,"statement: TRUNCATE edb_tbl;
",,,,,,,,,,"psql","client backend","TRUNCATE TABLE",,,,,,"truncate"
```

ケース 3：ロール admin によってデータベース auditdb に加えられた変更

create table、insert、および update ステートメントのみが監査されます。

```
$ psql auditdb admin
Password for user admin:
psql.bin (14.0.0)
Type "help" for help.

auditdb=# SHOW edb_audit_statement;
```

(次のページに続く)

(前のページからの続き)

```
__OUTPUT__
      edb_audit_statement
-----
create table, insert, update
(1 row)
```

```
auditdb=# CREATE TABLE audit_tbl_2 (f1 INTEGER PRIMARY KEY, f2 TEXT);
CREATE TABLE
auditdb=# INSERT INTO audit_tbl_2 VALUES (1, Row 1);
INSERT 0 1
auditdb=# SELECT * FROM audit_tbl_2;                <== Should not be audited
__OUTPUT__
 f1 | f2
-----+-----
  1 | Row 1
(1 row)
```

```
auditdb=# TRUNCATE audit_tbl_2;                <== Should not be audited
TRUNCATE TABLE
```

監査ログファイルの続きは次のように表示されます。3 番目のケースを表す最後から 2 つのエントリには、CREATE TABLE audit_tbl_2 および INSERT INTO audit_tbl_2 ステートメントのみが表示されます。SELECT * FROM audit_tbl_2 および TRUNCATE audit_tbl_2 ステートメントは監査されませんでした。

```
2020-05-25 15:59:12.332 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.a36d,2,
"idle",2020-05-25 15:59:02 IST,5/7,0,AUDIT,000000,"statement: CREATE TABLE
audit_tbl(f1 INTEGER PRIMARY KEY, f2 TEXT);",,,,,,,,,,"psql","client backend","CREATE_
→TABLE",,,,,,"create"

2020-05-25 15:59:22.419 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.a36d,3,
"idle",2020-05-25 15:59:02 IST,5/8,0,AUDIT,000000,"statement: INSERT INTO
audit_tbl VALUES (1, Row 1);",,,,,,,,,,"psql","client backend","INSERT",,,,,,"insert"

2020-05-25 15:59:32.180 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.a36d,4,
"idle",2020-05-25 15:59:02 IST,5/9,0,AUDIT,000000,"statement: UPDATE
audit_tbl SET f2 = Row A WHERE f1 = 1;",,,,,,,,,,"psql","client backend","UPDATE",,,,,
→"update"

2021-02-18 08:04:57.434 IST,"admin","edb",3182,"[local]",602e65dc.c6e,1,
"idle",2021-02-18 08:04:28 IST,4/22,0,AUDIT,000000,"statement: SET default_with_rowids_
→TO TRUE;
",,,,,,,,,,"psql","client backend","SET",,,,,,"set"
```

(次のページに続く)

(前のページからの続き)

```

2021-02-18 08:06:01.662 IST,"admin","edb",3182,"[local]",602e65dc.c6e,2,
"idle",2021-02-18 08:04:28 IST,4/27,0,AUDIT,000000,"statement: SELECT * FROM edb_tbl;
,,,,,,,,,"psql","client backend","SELECT","", "select"

2021-02-18 08:06:11.125 IST,"admin","edb",3182,"[local]",602e65dc.c6e,3,
"idle",2021-02-18 08:04:28 IST,4/28,0,AUDIT,000000,"statement: TRUNCATE edb_tbl;
,,,,,,,,,"psql","client backend","TRUNCATE TABLE","", "truncate"

2020-05-25 17:30:59.057 IST,"admin","auditdb",122093,"[local]",5ecbb370.1dcde,2,
"idle",2020-05-25 17:30:48 IST,5/11,0,AUDIT,000000,"statement: CREATE TABLE
audit_tbl_2 (f1 INTEGER PRIMARY KEY, f2 TEXT);",,,,,,,,,,"psql","client backend",
→ "CREATE TABLE","", "create"

2020-05-25 17:31:08.866 IST,"admin","auditdb",122093,"[local]",5ecbb370.1dcde,3,
"idle",2020-05-25 17:30:48 IST,5/12,0,AUDIT,000000,"statement: INSERT INTO
audit_tbl_2 VALUES (1, Row 1);",,,,,,,,,,"psql","client backend","INSERT","", "insert"

```

3.5.4 Audit log file

監査ログファイルは CSV または XML 形式で生成できます。形式は、`edb_audit` 構成パラメーターによって決まります。

監査ログの情報は、PostgreSQL core documentation の「エラーの報告とロギング」の「CSV 形式のログ出力の使用」で説明されているように、PostgreSQL によって実行されたロギングに基づいています。

次の表に、CSV 監査ログ形式に表示される順にフィールドをリストします。テーブルには次の情報が含まれています。

- フィールド。PostgreSQL ドキュメントのサンプルテーブル定義に示されているフィールドの名前。
- XML 要素/属性。XML 形式の場合、値を参照する XML 要素の名前とその属性（使用される場合）。
- データ型。PostgreSQL サンプルテーブル定義で指定されたフィールドのデータ型。
- 説明。フィールドの説明。

ログに記録する値がないフィールドは、CSV 形式で連続したコンマ（、）として表示されます。

Field	XML element/attribute	Data type	Description
<i>log_time</i>	<i>event/log_time</i>	<i>timestamp with time zone</i>	ステートメントのログ日付/時刻。
<i>user_name</i>	<i>event/user</i>	<i>text</i>	ステートメントを実行したデータベースユーザー。
<i>database_name</i>	<i>event/database</i>	<i>text</i>	ステートメントが実行されたデータベース。

次のページに続く

表 5 – 前のページからの続き

Field	XML element/attribute	Data type	Description
<i>process_id</i>	<i>event/process_id</i>	<i>integer</i>	ステートメントが実行されたオペレーティングシステムのプロセス ID。
<i>connection_from</i>	<i>event/remote_host</i>	<i>text</i>	ステートメントが実行されたホストとポートの場所。
<i>session_id</i>	<i>event/session_id</i>	<i>text</i>	ステートメントが実行されたセッション ID。
<i>session_line_num</i>	<i>event/session_line_num</i>	<i>bigint</i>	セッション内のステートメントの順序。
<i>process_status</i>	<i>event/process_status</i>	<i>text</i>	処理状況。
<i>session_start_time</i>	<i>event/session_start_time</i>	<i>timestamp with time zone</i>	セッションが開始された日時。
<i>virtual_transaction_id</i>	<i>event/virtual_transaction_id</i>	<i>text</i>	ステートメントの仮想トランザクション ID。
<i>transaction_id</i>	<i>event/transaction_id</i>	<i>bigint</i>	ステートメントの通常のトランザクション ID。
<i>error_severity</i>	<i>error_severity</i>	<i>text</i>	ステートメントの重大度。値は、監査されたステートメントの場合は <i>AUDIT</i> 、結果のエラーメッセージの場合は <i>ERROR</i> です。
<i>sql_state_code</i>	<i>event/sql_state_code</i>	<i>text</i>	ステートメントに対して返された SQL 状態コード。XML ログ形式の値が 00000 の場合、`sql_state_code` はログに記録されません。
<i>message</i>	<i>message</i>	<i>text</i>	実行しようとした SQL ステートメント。
<i>detail</i>	<i>detail</i>	<i>text</i>	エラーメッセージの詳細。
<i>hint</i>	<i>hint</i>	<i>text</i>	エラーのヒント。
<i>internal_query</i>	<i>internal_query</i>	<i>text</i>	エラーの原因となった内部クエリ (ある場合)。
<i>internal_query_pos</i>	<i>internal_query_pos</i>	<i>integer</i>	その中のエラー位置の文字数。
<i>context</i>	<i>context</i>	<i>text</i>	エラーコンテキスト。
<i>query</i>	<i>query</i>	<i>text</i>	エラーの原因となったユーザークエリ。エラー専用。
<i>query_pos</i>	<i>query_pos</i>	<i>integer</i>	その中のエラー位置の文字数。エラー専用。
<i>location</i>	<i>location</i>	<i>text</i>	ソースコード内のエラーの場所。 `log_error_verbosity` が <i>verbose</i> に設定されている場合、 <i>location</i> フィールドに値が設定されます。

次のページに続く

表 5 – 前のページからの続き

Field	XML element/attribute	Data type	Description
<i>application_name</i>	<i>event/application_name</i>	<i>text</i>	ステートメントが実行されたアプリケーションの名前（例、 <i>psql.bin</i> ）。
<i>backend_type</i>	<i>event/backend_type</i>	<i>text</i>	<code>`backend_type`</code> は、 <code>`pg_stat_activity.backend_type`</code> が示すものに対応し、csv ログに列として追加されます。
<i>leader_pid</i>	<i>event/leader_pid</i>	<i>integer</i>	アクティブなパラレルワーカーのリーダーのプロセス ID。
<i>query_id</i>	<i>event/query_id</i>	<i>long</i>	このバックエンドの最新のクエリの識別子。
<i>command_tag</i>	<i>event/command_tag</i>	<i>text</i>	ステートメントの SQL コマンド。
<i>audit_tag</i>	<i>event/audit_tag</i>	<i>text</i>	構成ファイルの <code>`audit_tag`</code> パラメーターで指定された値。
<i>type</i>	<i>event/type</i>	<i>text</i>	ログ内のメッセージを識別する監査 <code>`event_type`</code> を決定します。

次の例は、CSV および XML 形式で生成されます。

`postgresql.conf` ファイルのデフォルト以外の監査設定は次のとおりです。

```
logging_collector = on
edb_audit = csv
edb_audit_connect = all
edb_audit_disconnect = all
edb_audit_statement = ddl, dml, select, error
edb_audit_tag = edbaudit
```

XML フォーマットを生成するときに `edb_audit parameter` は `xml` に変更されます。

監査済みセッションは次のとおりです。

```
$ psql edb enterprisedb
Password for user enterprisedb:
psql.bin (14.0.0)
Type "help" for help.

edb=# CREATE SCHEMA edb;
CREATE SCHEMA
edb=# SET search_path TO edb;
SET
edb=# CREATE TABLE dept (
edb(#      deptno          NUMBER(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
edb(#      dname           VARCHAR2(14) CONSTRAINT dept_dname_uq UNIQUE,
```

(次のページに続く)

(前のページからの続き)

```

edb=#      loc          VARCHAR2(13)
edb=# );
CREATE TABLE
edb=# INSERT INTO dept VALUES (10,ACCOUNTING,NEW YORK);
INSERT 0 1
edb=# UPDATE department SET loc = BOSTON WHERE deptno = 10;
ERROR:  relation "department" does not exist
LINE 1: UPDATE department SET loc = BOSTON WHERE deptno = 10;
          ^
edb=# UPDATE dept SET loc = BOSTON WHERE deptno = 10;
UPDATE 1
edb=# SELECT * FROM dept;
__OUTPUT__
 deptno |  dname   |  loc
-----+-----+-----
      10 | ACCOUNTING | BOSTON
(1 row)

edb=# \q

```

CSV 監査ログファイル

監査ログファイルの CSV 形式は次のとおりです。(各監査ログエントリは分割され、複数行に表示されます。見やすくするために監査ログエントリの上に空白行が挿入されました。)

```

2022-12-14 12:19:01.035 UTC,"enterprisedb","edb",9290,"[local]",
6399bf35.244a,1,"authentication",2022-12-14 12:19:01 UTC,4/19,0,
AUDIT,000000,"connection authorized: user=enterprisedb database=edb",
,,,,,,,"","client backend",,0,"","edbaudit","connect"

2022-12-14 12:19:12.599 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,1,"authentication",2022-12-14 12:19:12 UTC,5/1,0,
AUDIT,000000,"connection authorized: user=enterprisedb database=edb",
,,,,,,,"","client backend",,0,"","edbaudit","connect"

2022-12-14 12:19:21.351 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,2,"idle",2022-12-14 12:19:12 UTC,5/3,0,AUDIT,000000,
"statement: CREATE SCHEMA edb;",,,,,,,,"psql","client backend",,
0,"CREATE SCHEMA","edbaudit","create"

2022-12-14 12:19:27.817 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,3,"idle",2022-12-14 12:19:12 UTC,5/4,0,AUDIT,000000,
"statement: CREATE SCHEMA edb;",,,,,,,,"psql","client backend",,

```

(次のページに続く)

(前のページからの続き)

```

0,"CREATE SCHEMA","edbaudit","create"

2022-12-14 12:19:27.820 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,4,"CREATE SCHEMA",2022-12-14 12:19:12 UTC,5/4,0,ERROR,
42P06,"schema ""edb"" already exists",,,,,,"CREATE SCHEMA edb;",,,
"psql","client backend",,0,"CREATE SCHEMA","edbaudit","error"

2022-12-14 12:20:15.407 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,5,"idle",2022-12-14 12:19:12 UTC,5/6,0,AUDIT,000000,
"statement: CREATE TABLE dept (
    deptno          NUMBER(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
    dname            VARCHAR2(14) CONSTRAINT dept_dname_uq UNIQUE,
    loc              VARCHAR2(13)
);",,,,,,"psql","client backend",,0,"CREATE TABLE","edbaudit",
"create"

2022-12-14 12:20:24.433 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,6,"idle",2022-12-14 12:19:12 UTC,5/7,0,AUDIT,000000,
"statement: INSERT INTO dept VALUES (10,ACCOUNTING,NEW YORK);",
,,,,,"psql","client backend",,0,"INSERT","edbaudit","insert"

2022-12-14 12:20:34.393 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,7,"idle",2022-12-14 12:19:12 UTC,5/8,0,AUDIT,000000,
"statement: UPDATE department SET loc = BOSTON WHERE deptno = 10;",
,,,,,"psql","client backend",,0,"UPDATE","edbaudit","update"

2022-12-14 12:20:34.394 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,8,"UPDATE",2022-12-14 12:19:12 UTC,5/8,0,ERROR,
42P01,"relation ""department"" does not exist",,,,,,
"UPDATE department SET loc = BOSTON WHERE deptno = 10;",8,,
"psql","client backend",,0,"UPDATE","edbaudit","error"

2022-12-14 12:20:43.721 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,9,"idle",2022-12-14 12:19:12 UTC,5/9,0,AUDIT,000000,
"statement: UPDATE dept SET loc = BOSTON WHERE deptno = 10;",
,,,,,"psql","client backend",,0,"UPDATE","edbaudit","update"

2022-12-14 12:20:51.231 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,10,"idle",2022-12-14 12:19:12 UTC,5/10,0,AUDIT,000000,
"statement: SELECT * FROM dept;",,,,,,"psql","client backend",,0,
"SELECT","edbaudit","select"

2022-12-14 12:20:53.940 UTC,"enterprisedb","edb",9293,"[local]",
6399bf40.244d,11,"idle",2022-12-14 12:19:12 UTC,,0,AUDIT,000000,
"disconnection: session time: 0:01:41.344 user=enterprisedb database=edb

```

(次のページに続く)

(前のページからの続き)

```
host=[local]",,,,,,,,,,"psql","client backend",,0,"","edbaudit","disconnect"
```

XML 監査ログファイル

監査ログファイルの XML 形式は次のとおりです。(出力は、視覚的にわかりやすくするためにフォーマットされています。)

```
<event user="amul" database="edb" process_id="110405"
  remote_host="[local]" session_id="63e1f4cf.1af45"
  session_line_num="1" process_status="idle"
  session_start_time="2023-02-07 12:20:56.920 IST"
  log_time="2023-02-07 12:20:55 IST" virtual_transaction_id="4/22"
  type="create" command_tag="CREATE ROLE" application_name="psql"
  backend_type="client backend" query_id="0">
  <error_severity>AUDIT</error_severity>
  <message>statement: create user edb superuser;</message>
</event>
<event user="edb" database="postgres" process_id="110424"
  remote_host="[local]" session_id="63e1f4d4.1af58"
  session_line_num="1" process_status="idle"
  session_start_time="2023-02-07 12:21:00.561 IST"
  log_time="2023-02-07 12:21:00 IST" virtual_transaction_id="4/25"
  type="set" command_tag="SET" application_name="pg_regress"
  backend_type="client backend" query_id="0">
  <error_severity>AUDIT</error_severity>
  <message>statement: SET client_min_messages = warning</message>
</event>
<event user="edb" database="postgres" process_id="110424"
  remote_host="[local]" session_id="63e1f4d4.1af58"
  session_line_num="2" process_status="idle"
  session_start_time="2023-02-07 12:21:00.561 IST"
  log_time="2023-02-07 12:21:00 IST" virtual_transaction_id="4/26"
  type="drop" command_tag="DROP DATABASE" application_name="pg_regress"
  backend_type="client backend" query_id="0">
  <error_severity>AUDIT</error_severity>
  <message>statement: DROP DATABASE IF EXISTS &quot;regression&quot;</message>
</event>
<event user="edb" database="postgres" process_id="110426"
  remote_host="[local]" session_id="63e1f4d4.1af5a"
  session_line_num="1" process_status="idle"
  session_start_time="2023-02-07 12:21:00.568 IST"
  log_time="2023-02-07 12:21:00 IST" virtual_transaction_id="4/29"
  type="create" command_tag="CREATE DATABASE" application_name="pg_regress"
```

(次のページに続く)

(前のページからの続き)

```

        backend_type="client backend" query_id="0">
<error_severity>AUDIT</error_severity>
<message>statement: CREATE DATABASE &quot;regression&quot;; TEMPLATE=template0</
→message>
</event>
<event user="edb" database="postgres" process_id="110426"
        remote_host="[local]" session_id="63e1f4d4.1af5a"
        session_line_num="2" process_status="idle"
        session_start_time="2023-02-07 12:21:00.983 IST"
        log_time="2023-02-07 12:21:00 IST" virtual_transaction_id="4/30"
        type="alter" command_tag="ALTER DATABASE" application_name="pg_regress"
        backend_type="client backend" query_id="0">
<error_severity>AUDIT</error_severity>
<message>statement: ALTER DATABASE &quot;regression&quot;; SET lc_messages TO &
→apos;C&apos;;ALTER DATABASE &quot;regression&quot;; SET lc_monetary TO &apos;C&apos;;
→ALTER DATABASE &quot;regression&quot;; SET lc_numeric TO &apos;C&apos;;ALTER
→DATABASE &quot;regression&quot;; SET lc_time TO &apos;C&apos;;ALTER DATABASE &quot;
→regression&quot;; SET bytea_output TO &apos;hex&apos;;ALTER DATABASE &quot;
→regression&quot;; SET timezone_abbreviations TO &apos;Default&apos;;</message>
</event>
<event user="edb" database="regression" process_id="110429"
        remote_host="[local]" session_id="63e1f4d5.1af5d"
        session_line_num="1" process_status="idle"
        session_start_time="2023-02-07 12:21:01.152 IST"
        log_time="2023-02-07 12:21:01 IST" virtual_transaction_id="4/33"
        type="set" command_tag="SET" application_name="pg_regress/test_setup"
        backend_type="client backend" query_id="0">
<error_severity>AUDIT</error_severity>
<message>statement: SET synchronous_commit = on;</message>
</event>
<event user="enterprisedb" database="edb" process_id="5942" remote_host=
"[local]"
        session_id="5ec7ac4d.1735" session_line_num="8" process_status="idle"
        session_start_time="2023-02-07 16:11:17 IST" log_time="2023-02-07
16:12:45.471 IST" virtual_transaction_id="4/27" type="select"
        command_tag="SELECT" audit_tag="edbaudit" application_name="psql"
        backend_type="parallel worker" leader_pid="5940" query_id="0">
        <error_severity>AUDIT</error_severity>
        <message>statement: SELECT * FROM dept;</message>
</event>
<event process_id="112399" session_id="63e1f4e2.1b70f" session_line_num="1"
        session_start_time="2023-02-07 12:21:14.150 IST"
        log_time="2023-02-07 12:21:14 IST" virtual_transaction_id="5/2012"
        type="error" sql_state_code="42601" command_tag="SELECT"
        application_name="pg_regress/timestamptz" backend_type="parallel worker"

```

(次のページに続く)

```

    leader_pid="112317" query_id="0">
    <error_severity>ERROR</error_severity>
    <message>date format not recognized</message>
    <query>SELECT to_char(d1, E'"HH:MI:SS is"
HH:MI:SS "\"text between quote marks\"&apos;')
    FROM TIMESTAMPTZ_TBL;</query>
</event>
<event process_id="112402" session_id="63e1f4e2.1b712" session_line_num="1"
    session_start_time="2023-02-07 12:21:14.156 IST" log_time="2023-02-07
    12:21:14 IST" virtual_transaction_id="5/2015" type="error"
    sql_state_code="42601" command_tag="SELECT"
    application_name="pg_regress/timestamptz" backend_type="parallel worker"
    leader_pid="112317" query_id="0">
    <error_severity>ERROR</error_severity>
    <message>date format not recognized</message>
    <query>SELECT to_char(d1, &apos;HH24--text--MI--text--SS&apos;')
    FROM TIMESTAMPTZ_TBL;</query>
</event>

```

3.5.5 Using error codes to filter audit logs

EDB Postgres Advanced Server には、ユーザー指定のエラーコードを含む EDB Postgres Advanced Server ログファイルエントリから除外するために使用できる拡張機能が含まれています。監査ログエントリをフィルタリングするには、最初に `postgresql.conf` ファイルを変更して拡張機能を有効にする必要があります。 `shared_preload_libraries` パラメーターで指定された値に次の値を追加します。

```
$libdir/edb_filter_log
```

次に、 `edb_filter_log.errcodes` パラメーターを使用して、ログファイルから除外するエラーコードを指定します。

```
edb_filter_log.errcode = error_code
```

`error_code` は、ログファイルから省略する 1 つ以上のエラーコードを指定します。複数のエラーコードをコンマ区切りのリストで指定します。

たとえば、 `edb_filter_log` が有効で、 `edb_filter_log.errcode` が `'23505,23502,22012'` に設定されている場合、次の SQLSTATE エラーのいずれかを返すログエントリはログファイルから省略されます。

23505 (ユニーク制約に違反するため)

23502 (not-null 制約に違反するため)

22012 (ゼロ除算用)

EDB Postgres Advanced Server 監査ログフィルタリングでサポートされているエラーコードの完全なリストについては、

Postgres core documentation を参照してください。

3.5.6 Using command tags to filter audit logs

ログファイルの各エントリには、エラーメッセージを表示するエントリを除き、**コマンドタグ**が含まれています。コマンドタグは、そのログエントリに対して実行される SQL コマンドです。コマンドタグを使用すると、後でツールを使用してログファイルをスキャンし、SQL コマンドに関連するエントリを見つけることができます。

この例は、確認しやすいようにフォーマットされた XML 形式です。コマンドタグは、値 `CREATE ROLE`、`ALTER ROLE`、および `DROP ROLE` を持つ `event` 要素の `command_tag` 属性として表示されます。

```
<event user="enterprisedb" database="edb" process_id="64234" remote_host=
"[local]"
  session_id="5ecbc7e6.faea" session_line_num="2" process_status="idle"
  session_start_time="2020-05-25 18:58:06 IST" log_time="2020-05-25
18:58:21.147 IST"
  virtual_transaction_id="4/30" type="create" command_tag="CREATE ROLE"
  application_name="psql" backend_type="client backend">
  <error_severity>AUDIT</error_severity>
  <message>statement: CREATE ROLE newuser WITH LOGIN;</message>
</event>
<event user="enterprisedb" database="edb" process_id="64234" remote_host=
"[local]"
  session_id="5ecbc7e6.faea" session_line_num="3" process_status="idle"
  session_start_time="2020-05-25 18:58:06 IST" log_time="2020-05-25
18:58:34.142 IST"
  virtual_transaction_id="4/31" type="error" sql_state_code="42601"
  application_name="psql" backend_type="client backend">
  <error_severity>ERROR</error_severity>
  <message>unrecognized role option &quot;super&quot;</message>
  <query>ALTER ROLE newuser WITH SUPER USER;</query>
  <query_pos>25</query_pos>
</event>
<event user="enterprisedb" database="edb" process_id="64234" remote_host=
"[local]"
  session_id="5ecbc7e6.faea" session_line_num="4" process_status="idle"
  session_start_time="2020-05-25 18:58:06 IST" log_time="2020-05-25
18:58:44.680 IST"
  virtual_transaction_id="4/32" type="alter" command_tag="ALTER ROLE"
  application_name="psql" backend_type="client backend">
  <error_severity>AUDIT</error_severity>
  <message>statement: ALTER ROLE newuser WITH SUPERUSER;</message>
</event>
<event user="enterprisedb" database="edb" process_id="64234" remote_host=
"[local]"
```

(次のページに続く)

(前のページからの続き)

```

session_id="5ecbc7e6.faea" session_line_num="5" process_status="idle"
session_start_time="2020-05-25 18:58:06 IST" log_time="2020-05-25
18:58:58.173 IST"
virtual_transaction_id="4/33" type="drop" command_tag="DROP ROLE"
application_name="psql" backend_type="client backend">
  <error_severity>AUDIT</error_severity>
  <message>statement: DROP ROLE newuser;</message>
</event>

```

以下は CSV 形式の同じ例です。コマンドタグは、各エントリの最後から 2 番目の列です。リストで、空 (「」) と表示される列は、`edb_audit_tag` パラメーターの値です。

(各監査ログエントリは分割され、複数行に表示されます。見やすくするために監査ログエントリの間空白行が挿入されました。)

```

2020-05-25 19:09:32.105 IST,"enterprisedb","edb",77212,"[local]",5ecbca7b.
12d9c,2,"idle",2020-05-25 19:09:07 IST,4/30,0,AUDIT,000000,"statement: CREATE
ROLE newuser WITH LOGIN;",,,,,,,,,,"psql","client backend","CREATE ROLE","",
"create"

2020-05-25 19:09:50.975 IST,"enterprisedb","edb",77212,"[local]",5ecbca7b.
12d9c,3,"idle",2020-05-25 19:09:07 IST,4/31,0,ERROR,42601,"unrecognized role
option ""super""",,,,,,"ALTER ROLE newuser WITH SUPER USER;",25,,,"psql",
"client backend","", "", "error"

2020-05-25 19:10:04.128 IST,"enterprisedb","edb",77212,"[local]",5ecbca7b.
12d9c,4,"idle",2020-05-25 19:09:07 IST,4/32,0,AUDIT,000000,"statement: ALTER
ROLE newuser WITH SUPERUSER;",,,,,,,,,,"psql","client backend","ALTER ROLE","",
"alter"

2020-05-25 19:10:15.959 IST,"enterprisedb","edb",77212,"[local]",5ecbca7b.
12d9c,5,"idle",2020-05-25 19:09:07 IST,4/33,0,AUDIT,000000,"statement: DROP
ROLE newuser;",,,,,,,,,,"psql","client backend","DROP ROLE","", "drop"

```

3.5.7 Redacting passwords in audit logs

`postgresql.conf` ファイルの下の GUC パラメーター `edb_filter_log.redact_password_commands` を使用して、ログファイルに保存されているパスワードをリダクションできます。

構文は次のとおりです。

```

{CREATE|ALTER} {USER|ROLE|GROUP} identifier { [WITH] [ENCRYPTED] PASSWORD
nonempty_string_literal | IDENTIFIED BY { nonempty_string_literal |
bareword } } [ REPLACE { nonempty_string_literal | bareword } ]

```

パスワードリダクションを有効にするには、最初に `postgresql.conf` ファイルを変更してパラメーターを有効にする必要があります。 `shared_preload_libraries` パラメーターで指定された値に次の値を追加します。

```
$libdir/edb_filter_log
```

次に、 `edb_filter_log.redact_password_commands` を `true` に設定します。

```
edb_filter_log.redact_password_commands = true
```

`postgresql.conf` ファイルを変更した後、サーバーを再起動して変更を有効にする必要があります。

例

次のステートメントが `log_statement` によってログに記録されると、サーバーは `x` へのパスワードをリダクションします。たとえば、次のコマンドがログファイルに追加されます。

```
CREATE USER carol with login PASSWORD 1safepwd;
```

次のように表示されます。

```
2021-05-17 05:01:46.841 IST,"enterprisedb","edb",18415,"[local]",60a230f0.47ef,1,"idle
→",2021-05-17 05:01:36 IST,3/3,0,AUDIT,00000,"statement: CREATE USER carol with
→login PASSWORD x;",,,,,,,,,,"psql","client backend","CREATE ROLE","", "create"
```

次のステートメントが `log_statement` によってログに記録されると、サーバーは新しいパスワードを識別し、パスワードを `x` に置き換えます。たとえば、次のコマンドがログファイルに追加されます。

```
ALTER USER carol IDENTIFIED BY new_pass REPLACE 1safepwd;
```

次のように表示されます。

```
2021-05-19 04:41:45.718 IST,"enterprisedb","edb",19354,"[local]",60a23a72.4b9a,1,"idle
→",2021-05-19 04:41:23 IST,3/3,0,AUDIT,00000,"statement: ALTER USER carol IDENTIFIED
→BY x REPLACE x;",,,,,,,,,,"psql","client backend","ALTER ROLE","", "alter"
```

パスワードを含むステートメントがログに記録されます。サーバーは `x` へのパスワードテキストをリダクションします。ステートメントが他のメッセージのコンテキストとしてログに記録される場合、サーバーはコンテキストからステートメントを省略します。

3.5.8 Audit log archiving

EDB 監査ログアーカイブにより、データベース管理者は監査ログディレクトリが消費する領域を制御し、監査ログファイルの管理に役立ちます。Audit Archiver は、audit ディレクトリにある `edb_audit_archiver_filename_prefix` を使用してログファイルの圧縮、実行、および有効期限を処理します。`edb_audit_archiver_timeout` パラメーターは、適切なタイミングで監査ログファイルの圧縮または有効期限をトリガーします。監査アーカイブ構成パラメーターの詳細については、

Audit logging configuration parameters を参照してください。

監査アーカイバーは次のことに役立ちます。

- 圧縮する監査ログディレクトリからログファイルのリストを準備します。
- `edb_audit_archiver_compress_time_limit`、`edb_audit_archiver_compress_size_limit` パラメーターを基に、圧縮するログファイルを決定します。
- ログファイルの圧縮は、`edb_audit_archiver_compress_command` パラメーターを元に圧縮コマンドを指定して行います。
- `edb_audit_archiver_expire_time_limit` および `edb_audit_archiver_expire_size_limit` パラメーターに基づいて、削除するログファイルを決定します。
- `edb_audit_archiver_expire_command` パラメーターで指定した expiration コマンドを実行し、ログファイルを削除します。

古い監査ログファイルをローテーションするには、新しいファイルが作成されるログファイルのローテーション日を設定します。これを行うには、パラメーター `edb_audit_rotation_day` を目的の値に設定します。監査ログレコードは、これ以上の監査ログレコードに使用できる領域がない場合、先入れ先出しベースで上書きされます。

ログファイルの圧縮と有効期限を有効にするように EDB Postgres Advanced Server を構成するには：

1. `postgresql.conf` ファイルで `edb_audit_archiver` パラメーターを `on` に設定して、監査ログのアーカイブを有効にします。
1. ログファイルの圧縮を有効にするには、パラメーター `edb_audit_archiver_compress_size_limit` および `edb_audit_archiver_compress_time_limit` を必要な値に設定します。
1. ログファイルの有効期限を有効にするには、パラメーター `edb_audit_archiver_expire_size_limit` および `edb_audit_archiver_expire_time_limit` を必要な値に設定します。
1. 圧縮と有効期限の両方を有効にするには、パラメーター `edb_audit_archiver_compress_size_limit`、`edb_audit_archiver_compress_time_limit`、`edb_audit_archiver_expire_size_limit`、`edb_audit_archiver_expire_time_limit`、および `edb_audit_archiver_expire_command` を必要な値に設定します。

以下は、`postgresql.conf` ファイルの構成パラメーターの設定の例です。

構成ファイルで `edb_audit_archiver` パラメーターを設定すると、データベースクラスター全体に影響します。`postgresql.conf` ファイルに示すように、データベースクラスターは `edb_audit_archiver` を `on` に設定して確立されています。監査ログファイルは、`edb_audit` 構成パラメーターの設定に基づいて CSV 形式で生成されます。

```
logging_collector = on
edb_audit = csv
edb_audit_archiver = on # (Change requires restart)
```

例

この例では、`edb_audit_archiver`、`edb_audit_archiver_compress_size_limit`、および `edb_audit_archiver_compress_time_limit` は、10MB を超える監査ログファイルの圧縮を有効にするように設定されています。

```
edb_audit_archiver = on
edb_audit_archiver_compress_size_limit = 10
edb_audit_archiver_compress_time_limit = 0
```

圧縮前の監査ログファイルは次のように表示されます。

```
- rw-----. 1 enterprisedb edb 2097276 Jun 4 11:42 audit-20200811_114237.csv <==
→Shows non-compressed audit log files
- rw-----. 1 enterprisedb edb 2097278 Jun 4 11:42 audit-20200811_114243.csv
- rw-----. 1 enterprisedb edb 2097289 Jun 4 11:42 audit-20200811_114245.csv
- rw-----. 1 enterprisedb edb 2097330 Jun 4 11:42 audit-20200811_114246.csv
- rw-----. 1 enterprisedb edb 2097343 Jun 4 11:42 audit-20200811_114248.csv
- rw-----. 1 enterprisedb edb 2097338 Jun 4 11:42 audit-20200811_114249.csv
```

`edb_audit_archiver` パラメーターは、最新のファイルを除くログファイルを確認します。監査ログディレクトリに少なくとも 10MB のログファイルを保持し、残りのログファイルを圧縮します。`.gz` は、既に圧縮されたログファイルの名前を指定します。圧縮後、監査ログファイルは次のように表示されます。

```
- rw-----. 1 enterprisedb edb 64119 Jun 4 11:42 audit-20200811_114237.csv.gz <==
→Shows compressed audit log files
- rw-----. 1 enterprisedb edb 64323 Jun 4 11:42 audit-20200811_114243.csv.gz
- rw-----. 1 enterprisedb edb 64091 Jun 4 11:42 audit-20200811_114245.csv.gz
- rw-----. 1 enterprisedb edb 64152 Jun 4 11:42 audit-20200811_114246.csv.gz
- rw-----. 1 enterprisedb edb 2097343 Jun 4 11:42 audit-20200811_114248.csv <==
→Shows non-compressed audit log files
- rw-----. 1 enterprisedb edb 2097338 Jun 4 11:42 audit-20200811_114249.csv
```

この例では、`edb_audit_archiver`、`edb_audit_archiver_expire_size_limit`、および `edb_audit_archiver_expire_time_limit` は、1 時間より古い監査ログファイルの有効期限を有効にするように設定されています。

```
edb_audit_archiver = on
edb_audit_archiver_expire_size_limit = 0
edb_audit_archiver_expire_time_limit = 3600
```

圧縮前の監査ログファイルは次のように表示されます。

```
- rw-----. 1 enterprisedb edb 2097367 Jun 4 13:23 audit-20200811_132352.csv <==
→Shows non-compressed audit log files scheduled for expiration/removal
- rw-----. 1 enterprisedb edb 2097395 Jun 4 13:24 audit-20200811_132353.csv
- rw-----. 1 enterprisedb edb 2097328 Jun 4 14:35 audit-20200811_143438.csv
- rw-----. 1 enterprisedb edb 2097276 Jun 4 14:35 audit-20200811_143502.csv
- rw-----. 1 enterprisedb edb 2097211 Jun 4 14:35 audit-20200811_143503.csv
- rw-----. 1 enterprisedb edb 2097152 Jun 4 14:35 audit-20200811_143504.csv
```

edb_audit_archiver パラメーターは、最新のファイルを除くログファイルを確認します。1 時間以上前のログファイルを削除します。有効期限が切れると、監査ログファイルは次のように表示されます。

```
- rw-----. 1 enterprisedb edb 2097328 Jun 4 14:35 audit-20200811_143438.csv
- rw-----. 1 enterprisedb edb 2097276 Jun 4 14:35 audit-20200811_143502.csv
- rw-----. 1 enterprisedb edb 2097211 Jun 4 14:35 audit-20200811_143503.csv
- rw-----. 1 enterprisedb edb 2097152 Jun 4 14:35 audit-20200811_143504.csv
```

この例では、`edb_audit_archiver`、`edb_audit_archiver_compress_size_limit`、`edb_audit_archiver_compress_time_limit`、`edb_audit_archiver_expire_size_limit`、`edb_audit_archiver_expire_time_limit`、および `edb_audit_archiver_expire_command` は、監査ログファイルの圧縮と有効期限の両方を有効にするように設定されています。

```
edb_audit_archiver = on
edb_audit_archiver_compress_size_limit = 4
edb_audit_archiver_compress_time_limit = 0
edb_audit_archiver_expire_size_limit = 5
edb_audit_archiver_expire_time_limit = 0
edb_audit_archiver_expire_command = cp %p /home/edb_audit/backup-log/
```

圧縮前の監査ログファイルは次のように表示されます。

```
- rw-----. 1 enterprisedb edb 2097522 Jun 4 13:02 audit-20200811_125816.csv <==
→Shows non-compressed audit log files ready to be removed/expired
- rw-----. 1 enterprisedb edb 2097510 Jun 4 13:02 audit-20200811_130215.csv
- rw-----. 1 enterprisedb edb 2097336 Jun 4 13:10 audit-20200811_130947.csv
- rw-----. 1 enterprisedb edb 2097271 Jun 4 13:10 audit-20200811_131034.csv
- rw-----. 1 enterprisedb edb 2097246 Jun 4 13:10 audit-20200811_131035.csv
- rw-----. 1 enterprisedb edb 2097289 Jun 4 13:10 audit-20200811_131036.csv
```

edb_audit_archiver パラメーターは、最新のファイルを除くログファイルを確認します。監査ログディレクトリに少なくとも 4MB のログファイルを保持し、残りのログファイルを圧縮します。ログファイルの有効期限をチェックしているときに、`edb_audit_archiver` は監査ログディレクトリに 5MB 以上のログファイルを保持し、残りのログファイルを削除します。圧縮と期限切れの後、監査ログファイルは次のように表示されます。

```
- rw-----. 1 enterprisedb edb 63854 Jun 4 13:10 audit-20200811_131034.csv.gz
- rw-----. 1 enterprisedb edb 63513 Jun 4 13:10 audit-20200811_131035.csv.gz
- rw-----. 1 enterprisedb edb 63738 Jun 4 13:10 audit-20200811_131036.csv.gz
```

expire コマンドは、`postgresql.conf` ファイルで `edb_audit_archiver_expire_command = 'cp %p /home/edb_audit/backup-log/'` として指定されます。`edb_audit_archiver` はこのコマンドを実行し、ログファイルを監査ログディレクトリから削除する前にバックアップログディレクトリにコピーします。圧縮と期限切れの後、バックアップログディレクトリは次のように表示されます。

```
- rw-----. 1 enterprisedb edb 2097522 Jun 4 13:02 audit-20200811_125816.csv.gz
- rw-----. 1 enterprisedb edb 2097510 Jun 4 13:02 audit-20200811_130215.csv.gz
- rw-----. 1 enterprisedb edb 2097336 Jun 4 13:10 audit-20200811_130947.csv.gz
```

3.5.9 Object auditing

オブジェクトレベルの監査では、特定のテーブル上の特定のデータ操作言語 (DML) ステートメント (SELECT、UPDATE、DELETE、INSERT など) のオブジェクトを選択的に監査できます。オブジェクトレベルの監査では、`edb_audit_statement` パラメーターで (@) または (-) を指定することにより、特定のグループを含めたり除外したりできます。DML ステートメントの詳細については、[Selecting SQL statements to audit](#) を参照してください。

次の構文を使用して、SELECT、UPDATE、DELETE、または INSERT ステートメントの `edb_audit_statement` パラメーター値を指定します。

```
{ select | update | delete | insert }{@ | -}groupname
```

例

この例では、`edb_audit_connect` と `edb_audit_statement` は次の非デフォルト値で設定されます。

```
logging_collector = on
edb_audit_connect = all
edb_audit = csv
edb_audit_statement = select, insert, update, delete
```

SELECT、INSERT、UPDATE、および DELETE コマンドによって呼び出される SQL ステートメントが監査されます。

発生するデータベースセッションは次のとおりです。

```
$ psql edb enterprisedb
Password for user enterprisedb:
psql.bin (14.0.0)
Type "help" for help.

edb=# SHOW edb_audit_connect;
__OUTPUT__
edb_audit_connect
- - - - -
```

(次のページに続く)

(前のページからの続き)

```
all
(1 row)
```

```
edb=# SHOW edb_audit_statement;
```

```
__OUTPUT__
```

```
    edb_audit_statement
```

```
-----
select, insert, update, delete
(1 row)
```

```
edb=# CREATE TABLE emp
```

```
edb-#      (empno NUMBER(4) NOT NULL,
```

```
edb-#      ename VARCHAR2(10),
```

```
edb-#      job VARCHAR2(9),
```

```
edb-#      mgr NUMBER(4),
```

```
edb-#      hiredate DATE,
```

```
edb-#      sal NUMBER(7, 2),
```

```
edb-#      comm NUMBER(7, 2),
```

```
edb-#      deptno NUMBER(2));
```

```
CREATE TABLE
```

```
edb=# ALTER TABLE emp SET (edb_audit_group = low_security);
```

```
ALTER TABLE
```

```
edb=# CREATE TABLE dept
```

```
edb-#      (deptno NUMBER(2),
```

```
edb-#      dname VARCHAR2(14),
```

```
edb-#      loc VARCHAR2(13) ) with (edb_audit_group = low_security);
```

```
CREATE TABLE
```

```
edb=# CREATE TABLE bonus
```

```
edb-#      (ename VARCHAR2(10),
```

```
edb-#      job VARCHAR2(9),
```

```
edb-#      sal NUMBER,
```

```
edb-#      comm NUMBER) with (edb_audit_group = high_security);
```

```
CREATE TABLE
```

```
edb=# CREATE TABLE sal
```

```
edb-#      (grade NUMBER,
```

```
edb-#      losal NUMBER,
```

```
edb-#      hisal NUMBER) with (edb_audit_group = high_security);
```

```
CREATE TABLE
```

```
edb=# SET edb_audit_statement = select@low_security@high_security, insert@high_
```

```
→security-low_security, update-low_security@high_security, delete@low_security-high_
```

```
→security;
```

(次のページに続く)

(前のページからの続き)

SET

```
edb=# SELECT reloptions FROM pg_class WHERE relname IN(emp, dept, bonus, sal);
```

```
__OUTPUT__
```

```
reloptions
```

```
- - - - -
```

```
{edb_audit_group=low_security}
```

```
{edb_audit_group=low_security}
```

```
{edb_audit_group=high_security}
```

```
{edb_audit_group=high_security}
```

```
(4 rows)
```

```
edb=# SELECT setting FROM pg_settings WHERE name = edb_audit_statement;
```

```
__OUTPUT__
```

```
setting
```

```
- - - - -
```

```
→ - - - - -
```

```
select@low_security@high_security, insert@high_security-low_security, update-low_
```

```
→ security@high_security, delete@low_security-high_security
```

```
(1 row)
```

```
edb=# SELECT * FROM emp;
```

```
__OUTPUT__
```

```
empno | ename | job | mgr | hiredate | sal | comm | deptno
```

```
- - - - -
```

```
(0 rows)
```

```
edb=# SELECT * FROM dept;
```

```
__OUTPUT__
```

```
deptno | dname | loc
```

```
- - - - -
```

```
(0 rows)
```

```
edb=# SELECT * FROM bonus;
```

```
__OUTPUT__
```

```
ename | job | sal | comm
```

```
- - - - -
```

```
(0 rows)
```

```
edb=# SELECT * FROM sal;
```

```
__OUTPUT__
```

```
grade | losal | hisal
```

```
- - - - -
```

```
(0 rows)
```

```

edb=# INSERT INTO emp VALUES (7369, SMITH, CLERK, 7902, TO_DATE(17-DEC-1980, DD-MON-
→YYYY), 800, NULL, 10);
INSERT 0 1
edb=# INSERT INTO emp VALUES (7788, SCOTT, ANALYST, 7566, TO_DATE(09-DEC-1982, DD-MON-
→YYYY), 3000, NULL, 20);
INSERT 0 1
edb=# INSERT INTO emp VALUES (7566, JONES, MANAGER, 7839, TO_DATE(2-APR-1981, DD-MON-
→YYYY), 2975, NULL, 30);
INSERT 0 1

edb=# INSERT INTO dept VALUES (10, ACCOUNTING, NEW YORK);
INSERT 0 1
edb=# INSERT INTO dept VALUES (20, RESEARCH, DALLAS);
INSERT 0 1
edb=# INSERT INTO dept VALUES (30, SALES, CHICAGO);
INSERT 0 1
edb=# INSERT INTO dept VALUES (40, OPERATIONS, BOSTON);
INSERT 0 1

edb=# INSERT INTO bonus VALUES (SMITH, CLERK, 800, NULL);
INSERT 0 1
edb=# INSERT INTO bonus VALUES (SCOTT, ANALYST, 3000, NULL);
INSERT 0 1
edb=# INSERT INTO bonus VALUES (JONES, MANAGER, 2975, 300);
INSERT 0 1

edb=# INSERT INTO sal VALUES (3, 800, 1500);
INSERT 0 1
edb=# INSERT INTO sal VALUES (2, 2975, 4000);
INSERT 0 1
edb=# INSERT INTO sal VALUES (1, 3000, 5000);
INSERT 0 1

edb=# UPDATE emp SET sal = 5500 WHERE deptno = 10;
UPDATE 1
edb=# UPDATE dept SET loc = BEDFORD WHERE deptno = 20;
UPDATE 1
edb=# UPDATE bonus SET sal = 4000 WHERE ename = SMITH;
UPDATE 1
edb=# UPDATE sal SET losal = 5000;
UPDATE 3

edb=# DELETE FROM emp WHERE deptno = 10;
DELETE 1
edb=# DELETE FROM dept WHERE deptno = 20;

```

(次のページに続く)

(前のページからの続き)

```

DELETE 1
edb=# DELETE FROM bonus WHERE sal = 4000;
DELETE 1
edb=# DELETE FROM sal WHERE losal = 5000;
DELETE 3

```

enterprisedb ユーザーと edb データベースの edb_audit_statement パラメーターを 'select@low_security@high_security, insert@high_security-low_security, update-low_security@high_security, delete@low_security-high_security' に設定すると、ファイル内の監査ログに (@) を含み、- を除外する SELECT、INSERT、UPDATE または DELETE ステートメントを監査できます。

ログファイルのテーブルの場合：

- select@low_security@high_security は、low_security および high_security 監査グループの SELECT ステートメントを監査します。
- insert@high_security-low_security は high_security の INSERT ステートメントを監査し、low_security 監査グループの insert ステートメントを除外します。
- update-low_security@high_security は、high_security の UPDATE ステートメントを監査し、low_security 監査グループの update ステートメントを除外します。
- delete@low_security-high_security は low_security の DELETE ステートメントを監査し、ログファイル内のテーブルの high_security 監査グループの delete ステートメントを除外します。

生成される監査ログファイルには次が含まれます。(各監査ログエントリは分割され、複数行に表示されます。見やすくするために監査ログエントリの間に空白行が挿入されました。)

```

2021-07-20 01:45:27.077 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,1,"authentication",2021-07-20 01:45:27 IST,5/1,0,AUDIT,000000,"connection_
→authorized: user=enterprisedb
database=edb",,,,,,,,,,"client backend",,,,,,,,,,"connect"

2021-07-20 01:50:40.125 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,2,"SELECT",2021-07-20 01:45:27 IST,5/13,0,AUDIT,000000,"statement: SELECT * FROM_
→emp;",,,,,,,,,
"psql","client backend",,"SELECT",,,,,,"select"

2021-07-20 01:50:53.778 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,3,"SELECT",2021-07-20 01:45:27 IST,5/14,0,AUDIT,000000,"statement: SELECT * FROM_
→dept;",,,,,,,,,
"psql","client backend",,"SELECT",,,,,,"select"

2021-07-20 01:51:05.306 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,4,"SELECT",2021-07-20 01:45:27 IST,5/15,0,AUDIT,000000,"statement: SELECT * FROM_
→bonus;",,,,,,,,,
"psql","client backend",,"SELECT",,,,,,"select"

```

(次のページに続く)

(前のページからの続き)

```

2021-07-20 01:51:14.874 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,5,"SELECT",2021-07-20 01:45:27 IST,5/16,0,AUDIT,00000,"statement: SELECT * FROM_
→sal;","",,,,,,,
"psql","client backend",,"SELECT","",,"select"

2021-07-20 01:52:53.413 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,6,"INSERT",2021-07-20 01:45:27 IST,5/24,0,AUDIT,00000,"statement: INSERT INTO_
→bonus VALUES (SMITH, CLERK, 800, NULL);
",,,,,,,,"psql","client backend",,"INSERT","",,"insert"

2021-07-20 01:53:02.945 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,7,"INSERT",2021-07-20 01:45:27 IST,5/25,0,AUDIT,00000,"statement: INSERT INTO_
→bonus VALUES (SCOTT, ANALYST, 3000, NULL);
",,,,,,,,"psql","client backend",,"INSERT","",,"insert"

2021-07-20 01:53:12.064 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,8,"INSERT",2021-07-20 01:45:27 IST,5/26,0,AUDIT,00000,"statement: INSERT INTO_
→bonus VALUES (JONES, MANAGER, 2975, 300);
",,,,,,,,"psql","client backend",,"INSERT","",,"insert"

2021-07-20 01:53:23.836 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,9,"INSERT",2021-07-20 01:45:27 IST,5/27,0,AUDIT,00000,"statement: INSERT INTO_
→sal VALUES (3, 800, 1500);
",,,,,,,,"psql","client backend",,"INSERT","",,"insert"

2021-07-20 01:53:33.280 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,10,"INSERT",2021-07-20 01:45:27 IST,5/28,0,AUDIT,00000,"statement: INSERT INTO_
→sal VALUES (2, 2975, 4000);
",,,,,,,,"psql","client backend",,"INSERT","",,"insert"

2021-07-20 01:53:42.388 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,11,"INSERT",2021-07-20 01:45:27 IST,5/29,0,AUDIT,00000,"statement: INSERT INTO_
→sal VALUES (1, 3000, 5000);
",,,,,,,,"psql","client backend",,"INSERT","",,"insert"

2021-07-20 01:54:17.126 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,12,"UPDATE",2021-07-20 01:45:27 IST,5/32,0,AUDIT,00000,"statement: UPDATE bonus_
→SET sal = 4000 WHERE ename = SMITH;
",,,,,,,,"psql","client backend",,"UPDATE","",,"update"

2021-07-20 01:54:34.344 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,13,"UPDATE",2021-07-20 01:45:27 IST,5/33,0,AUDIT,00000,"statement: UPDATE sal_
→SET losal = 5000;
",,,,,,,,"psql","client backend",,"UPDATE","",,"update"

```

(次のページに続く)

(前のページからの続き)

```

2021-07-20 01:54:45.887 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,14,"DELETE",2021-07-20 01:45:27 IST,5/34,0,AUDIT,00000,"statement: DELETE FROM
→emp WHERE deptno = 10;
",,,,,,,,,,"psql","client backend",,"DELETE","",,"delete"

2021-07-20 01:54:54.513 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
155c2,15,"DELETE",2021-07-20 01:45:27 IST,5/35,0,AUDIT,00000,"statement: DELETE FROM
→dept WHERE deptno = 20;
",,,,,,,,,,"psql","client backend",,"DELETE","",,"delete"

```

3.6 Unicode collation algorithm

Unicode Collation Algorithm (UCA) は、Unicode データを照合および比較するカスタマイズ可能な方法を定義する仕様 (*Unicode Technical Report #10*) です。Collation は、SELECT ... ORDER BY 句のようにデータを並べ替える方法を意味します。比較は、より小さい、より大きい、または等しい演算子で範囲を使用する検索に関連します。

カスタマイズ性は、次のようなさまざまな理由で重要な要素です。

- Unicode は膨大な数の言語をサポートしています。いくつかの言語に共通する可能性のある文字は、言語に応じて異なる順序で照合されることが予想される場合があります。
- アクセントやウムラウトなどの特定の言語の文字と共に表示される文字は、言語によっては予想される照合に影響を与えます。
- 言語によっては、複数の連続する文字の組み合わせは、その照合順序に関して単一の文字として扱う必要があります。
- 場合によって文字の照合については特定の好みがある場合があります。たとえば、小文字の文字を同じ文字の大文字の前に照合するか、その逆にする必要があります。
- アンダースコア文字、ハイフン、またはスペース文字などの句読点を照合シーケンスで考慮するか、文字列に存在しないかのように単に無視するかどうかに関する設定がある場合があります。

Unicode がサポートする膨大な数の言語でこれらのバリエーションをすべて考えると、照合順序を決定するための特定の基準を選択する方法が必要です。これは、Unicode 照合アルゴリズムが定義するものです。

注釈: さらに、ICU 照合 (Unicode 照合アルゴリズムの実装) を使用するもう 1 つの利点はパフォーマンスです。B ツリーインデックスの作成を含む並べ替えタスクは、非 ICU 照合にかかる時間の半分未満で完了できます。正確なパフォーマンスの向上は、オペレーティングシステムのバージョン、テキストデータの言語、およびその他の要因によって異なります。

以下のセクションでは、統合照合アルゴリズムの概念を簡単かつ簡単に説明します。アルゴリズムとその使用法は非常に複雑であり、さまざまなバリエーションがあるため、詳細については、これらのセクションで引用されている公式ドキュメントを参照してください。

3.6.1 基本的な Unicode 照合アルゴリズムの概念

Unicode 照合アルゴリズムの公式情報は、*Unicode Technical Report #10* で指定されています。

<http://www.unicode.org/reports/tr10/>

ICU – International Components for Unicode も多くの有用な情報を提供します。照合の概念の説明は、次の Web サイトで見つけることができます。

<http://userguide.icu-project.org/collation/concepts>

Unicode 照合アルゴリズムの基本的な概念は、マルチレベル比較の使用です。これは、次の箇条書きにレベル 1 からレベル 5 としてリストされている多数のレベルが定義されることを意味します。各レベルは、比較のタイプを定義します。文字列は、最初にレベル 1 とも呼ばれるプライマリレベルを使用して比較されます。

プライマリレベルに基づいて順序を決定できる場合、アルゴリズムは完了です。プライマリレベルに基づいて順序を決定できない場合、セカンダリレベルのレベル 2 が適用されます。二次レベルに基づいて順序を決定できる場合はアルゴリズムが実行され、それ以外の場合は三次レベルが適用されます。通常、前のレベルで解決できない場合に順序を決定する最終的なタイブレークレベルがあります。

- **レベル 1–基本文字の初級レベル。** 文字や数字などの基本文字の順序で、 $A < B$ などの違いが決まります。
- ****レベル 2–アクセントの第 2 レベル。**
- **レベル 3–ケースの第 3 レベル。** プライマリレベルまたはセカンダリレベルの違いがない場合、ケースの違いによって $a < A$ などの順序が決まります。
- ****レベル 4–句読点の第 4 レベル。**
- **レベル 5–タイブレークの同一レベル。** 一次、二次、三次、または四次のレベル差がない場合、コードポイント値などの他の差によって順序が決まります。

3.6.2 Unicode の国際コンポーネント

Unicode 照合アルゴリズムは、*International Components for Unicode* (ICU) が提供するオープンソースソフトウェアによって実装されています。このソフトウェアは、C/C++および Java ライブラリのセットです。

EDB Postgres Advanced Server を使用して、ICU コンポーネントを呼び出して照合を作成する場合、結果は *ICU 照合* と呼ばれます。

ロケールの照合

ロケールの照合を作成する場合、通常、指定されたロケールの事前定義された ICU 短縮名が提供されます。

*ICU ショートフォーム*は、照合のプロパティである**照合属性**を指定する方法です。Collation Attributes は、照合属性に関する追加情報を提供します。

ロケールには、事前定義された ICU 短縮形があります。ロケールの ICU 短縮形には、特定のロケールに通常使用される照合属性設定が組み込まれています。これにより、そのロケールの照合属性のリスト全体を指定する必要がなくなるため、照合作成プロセスが簡素化されます。

システムカタログ `pg_catalog.pg_icu_collate_names` には、ロケールの ICU 短縮形の名前のリストが含まれています。ICU の短い形式の名前は、列 `icu_short_form` にリストされています。

```
edb=# SELECT icu_short_form, valid_locale FROM pg_icu_collate_names ORDER BY
valid_locale;
```

__OUTPUT__

icu_short_form	valid_locale
----------------	--------------

-----+-----	
-------------	--

LAF	af
LAR	ar
LAS	as
LAZ	az
LBE	be
LBG	bg
LBN	bn
LBS	bs
LBS_ZCYRL	bs_Cyrl
LROOT	ca
LROOT	chr
LCS	cs
LCY	cy
LDA	da
LROOT	de
LROOT	dz
LEE	ee
LEL	el
LROOT	en
LROOT	en_US
LEN_RUS_VPOSIX	en_US_POSIX
LEO	eo
LES	es
LET	et
LFA	fa
LFA_RAF	fa_AF

.
.
.

必要に応じて、特定のロケールの ICU 短縮形のデフォルトの特性を、そのプロパティをオーバーライドする照合属性を指定することによりオーバーライドできます。これについては、次のセクションで説明します。

照合属性

ICU 照合を作成するときは、照合に必要な特性を指定する必要があります。**Locale Collations** で説明したように、これは通常、目的のロケールの ICU 短い形式で実行できます。ただし、より具体的な情報が必要な場合は、**照合属性**を使用して照合プロパティを指定できます。

照合順序属性は、テキスト文字列の照合順序を決定するために文字を比較する方法のルールを定義します。Unicode は、国、地域、文化に応じてさまざまなバリエーションで膨大な言語セットをカバーするため、これらの照合属性は非常に複雑です。

照合属性の完全で正確な意味と使用法については、次の ICU – International Components for Unicode Web サイトのセクション「Collator Naming Scheme」を参照してください。

<http://userguide.icu-project.org/collation/concepts>

以下は、照合属性と、ICU 短い形式の方法を使用して指定する方法の簡単な概要です

各照合属性は、次の箇条書きにリストされている大文字で表されます。各属性の有効な値は、かっこ内に示されているコードで指定されます。一部のコードは、すべての属性に一般的な意味を持っています。**X** は、属性をオフに設定することを意味します。**O** は、属性をオンに設定することを意味します。**D** は、属性をデフォルト値に設定することを意味します。

- **A-代替 (N, S, D)**。空白、句読点、記号などの**変数**文字の処理を処理します。無視不可 (N) に設定すると、可変文字の違いは文字の違いと同じ重要度で扱われます。シフト (S) に設定すると、可変文字の違いはあまり重要ではありません（つまり、ベース文字を比較するときに可変文字は無視されます）。
- **C-ケースファースト (X, L, U, D)**。小文字を同じ大文字 (L) の前にソートするか、大文字を同じ小文字 (U) の前にソートするかを制御します。オフ (X) は、通常、小文字 (L) が必要な場合に指定されます。
- **E-ケースレベル (X, O, D)**。Strength 属性と組み合わせて設定すると、ケースレベル属性はアクセントを無視する場合に使用されますが、ケース。
- **F-フランス語の照合 (X, O, D)**。on に設定すると、カナダのフランス語ロケールと同様に、二次的な違い（アクセントの存在）が文字列の後ろからソートされます。
- ****H-互換性 4 進法 (X, O, D)**。
- **N-正規化チェック (X, O, D)**。比較のためにテキストを完全に正規化するかどうかを制御します。正規化は、異なるコードポイントシーケンスが同じ文字を表すため、そのような文字を並べ替えまたは比較するときに問題を引き起こす、テキストの標準的な同等性の問題を扱います。アラビア語、古代ギリシャ語、ヘブライ語、ヒンディー語、タイ語、ベトナム語などの言語は、正規化チェックをオンに設定して使用する必要があります。
- **S-強度 (1, 2, 3, 4, I, D)**。比較に使用される最大照合レベル。文字列を照合または比較するときに、アクセントまたは大文字と小文字が考慮されるかどうかに影響します。各数字はレベルを表します。I の設定は、同じ強さを表します（つまり、レベル 5）。

- **T-変数の先頭 (16 進数)**。Alternate 属性が無視不可 (N) に設定されていない場合にのみ適用されます。16 進数は、無視できると見なされる最大の文字シーケンスを指定します。たとえば、空白は無視できるが、可視の可変文字は無視できない場合、0020 に設定された Variable Top は S に設定された Alternate 属性と 3 に設定された Strength 属性とともに指定されます。は 16 進数 0020 です。バックスペース、タブ、ラインフィード、キャリッジリターンなどの他の非表示の可変文字には、0020 未満の値があります。すべての表示可能な句読点には 0020 より大きい値があります。)

一連の照合属性とその値は、照合属性の文字と目的の属性値で構成されるテキスト文字列で表されます。次の例に示すように、各属性/値のペアは、アンダースコア文字で次のペアに結合されます。

AN_CX_EX_FX_HX_NO_S3

ロケールの ICU 短縮名とともに照合属性を指定して、ロケールのデフォルトの属性設定をオーバーライドできます。

以下は、LR00T という名前の ICU の短縮形を他の多数の照合属性/値のペアで変更する例です。

AN_CX_EX_LR00T_NO_S3

前の例では、代替属性 (A) が無視できない (N に設定されています。Case First 属性 (C) は、off (X) に設定されています。ケースレベル属性 (E) はオフ (X) に設定されています。正規化属性 (N) は (0) に設定されています。強度属性 (S) は、3 次レベル 3 に設定されます。LR00T は、これらの他の属性が変更を適用している ICU の短縮形です。

3.6.3 照合の使用

新しく定義された ICU 照合は、COLLATION "collation_name" 句を SQL コマンドで使用できる場所 (CREATE TABLE コマンドの列指定、または SELECT コマンドの ORDER BY 句の式に追加) で使用できます。

次に、米国の英語に基づいた ICU 照合の作成と使用の例をいくつか示します (en_US.UTF8)。

これらの例では、次の特性を持つ ICU 照合が作成されます。

照合 icu_collate_lowercase は、小文字の文字を大文字の前にソートします (CL)。

照合 icu_collate_uppercase は、大文字の文字を対応する小文字の前にソートします。

照合 icu_collate_ignore_punct により、ソート中に可変文字 (空白と句読点) が無視されます (AS)。

照合 icu_collate_ignore_white_sp により、空白やその他の非表示の可変文字はソート中に無視されますが、表示されている可変文字 (句読点) は無視されません (AS、T0020)。

```
CREATE COLLATION icu_collate_lowercase (
    LOCALE = en_US.UTF8,
    ICU_SHORT_FORM = AN_CL_EX_NX_LR00T
);

CREATE COLLATION icu_collate_uppercase (
    LOCALE = en_US.UTF8,
    ICU_SHORT_FORM = AN_CU_EX_NX_LR00T
);
```

(次のページに続く)

(前のページからの続き)

```
CREATE COLLATION icu_collate_ignore_punct (
    LOCALE = en_US.UTF8,
    ICU_SHORT_FORM = AS_CX_EX_NX_LROOT_L3
);

CREATE COLLATION icu_collate_ignore_white_sp (
    LOCALE = en_US.UTF8,
    ICU_SHORT_FORM = AS_CX_EX_NX_LROOT_L3_T0020
);
```

注釈: 照合を作成するときに、*LROOT* 照合を変更する属性が指定されていると、ICU が通知および警告メッセージを生成する場合があります。

次の `psql` コマンドは、照合をリストします。

```
edb=# \d0
__OUTPUT__

                          List of collations
 Schema | Name | Collate | Ctype | ICU
-----+-----+-----+-----+-----
↪-----
enterprisedb | icu_collate_ignore_punct | en_US.UTF8 | en_US.UTF8 | AS_CX_EX_NX_
↪LROOT_L3
enterprisedb | icu_collate_ignore_white_sp | en_US.UTF8 | en_US.UTF8 |
AS_CX_EX_NX_LROOT_L3_T0020
enterprisedb | icu_collate_lowercase | en_US.UTF8 | en_US.UTF8 | AN_CL_EX_NX_
↪LROOT
enterprisedb | icu_collate_uppercase | en_US.UTF8 | en_US.UTF8 | AN_CU_EX_NX_
↪LROOT
(4 rows)
```

次のテーブルが作成され、データが入力されます。

```
CREATE TABLE collate_tbl (
    id          INTEGER,
    c2          VARCHAR(2)
);

INSERT INTO collate_tbl VALUES (1, A);
INSERT INTO collate_tbl VALUES (2, B);
INSERT INTO collate_tbl VALUES (3, C);
INSERT INTO collate_tbl VALUES (4, a);
INSERT INTO collate_tbl VALUES (5, b);
```

(次のページに続く)

(前のページからの続き)

```

INSERT INTO collate_tbl VALUES (6, c);
INSERT INTO collate_tbl VALUES (7, 1);
INSERT INTO collate_tbl VALUES (8, 2);
INSERT INTO collate_tbl VALUES (9, .B);
INSERT INTO collate_tbl VALUES (10, -B);
INSERT INTO collate_tbl VALUES (11, B);

```

次のクエリは、デフォルトの照合を使用して列 c2 で並べ替えます。9、10、および 11 の id 列値を持つ可変文字（空白と句読点）は無視され、文字 B でソートされることに注意してください。

```

edb=# SELECT * FROM collate_tbl ORDER BY c2;
__OUTPUT__
 id | c2
----+----
  7 | 1
  8 | 2
  4 | a
  1 | A
  5 | b
  2 | B
 11 | B
 10 | -B
  9 | .B
  6 | c
  3 | C
(11 rows)

```

次のクエリは、照合 `icu_collate_lowercase` を使用して列 c2 でソートします。これにより、小文字の文字が同じ基本文字の大文字の前にソートされます。また、AN 属性は、ベース文字を比較するときに可変文字を同じレベルでソート順に含めるため、9、10、および 11 の id 値を持つ行がソートリストの先頭に表示されることに注意してください。

```

edb=# SELECT * FROM collate_tbl ORDER BY c2 COLLATE "icu_collate_lowercase";
__OUTPUT__
 id | c2
----+----
 11 | B
 10 | -B
  9 | .B
  7 | 1
  8 | 2
  4 | a
  1 | A
  5 | b
  2 | B

```

(次のページに続く)

(前のページからの続き)

```

6 | c
3 | C
(11 rows)

```

次のクエリは、照合 `icu_collate_uppercase` を使用して列 `c2` でソートします。これは、大文字の文字を同じ基本文字の小文字の前にソートします。

```

edb=# SELECT * FROM collate_tbl ORDER BY c2 COLLATE "icu_collate_uppercase";
__OUTPUT__
 id | c2
-- +--
11 | B
10 | -B
9  | .B
7  | 1
8  | 2
1  | A
4  | a
2  | B
5  | b
3  | C
6  | c
(11 rows)

```

次のクエリは、照合 `icu_collate_ignore_punct` を使用して列 `c2` でソートします。これにより、可変文字が無視されるため、`id` 値が 9、10、および 11 の行は、無視される可変文字の直後にある文字 `B` でソートします。

```

edb=# SELECT * FROM collate_tbl ORDER BY c2 COLLATE
"icu_collate_ignore_punct";
__OUTPUT__
 id | c2
-- +--
7  | 1
8  | 2
4  | a
1  | A
5  | b
11 | B
2  | B
9  | .B
10 | -B
6  | c
3  | C
(11 rows)

```

次のクエリは、照合 `icu_collate_ignore_white_sp` を使用して列 `c2` で並べ替えます。照合の `AS` および `T0020` 属性により、16 進数の `0020` 以下のコードポイントを持つ可変文字は無視されますが、16 進数の `0020` より大きいコードポイントを持つ可変文字はソートに含まれます。

スペース文字（16 進数の `0020`）で始まる `id` の値が 11 である行は、文字 `B` でソートされます。 `id` 値が 9 および 10 であり、16 進数の `0020` より大きい表示可能な句読点で始まる行は、これらの特定の可変文字がベース文字を比較するときに同じレベルでソート順に含まれるため、ソートリストの先頭に表示されます。

```
edb=# SELECT * FROM collate_tbl ORDER BY c2 COLLATE
"icu_collate_ignore_white_sp";
__OUTPUT__
 id | c2
-- +---
 10 | -B
   9 | .B
   7 | 1
   8 | 2
   4 | a
   1 | A
   5 | b
  11 | B
   2 | B
   6 | c
   3 | C
(11 rows)
```


第4章 EDB Resource Manager

- EDB Resource Manager *は、EDB Postgres EDB Postgres Advanced Server プロセスで使用されるオペレーティングシステムリソースの使用を制御する機能を提供する EDB Postgres Advanced Server の機能です。

この機能により、特定のシステムリソースを制御不能に過剰使用および独占する可能性のあるプロセスからシステムを保護できます。

以下は、EDB Resource Manager の使用に関するいくつかの重要なポイントです。

- EDB Resource Manager の基本コンポーネントはリソースグループです。**リソースグループ**は、EDB Postgres Advanced Server インスタンス内のすべてのデータベースで使用可能な名前付きのグローバルグループであり、さまざまなリソース使用制限を定義できます。特定のリソースグループのメンバーとして割り当てられた EDB Postgres Advanced Server プロセスは、グループ内のすべてのプロセスのリソース使用量の合計がグループで定義された制限近くに維持されるように、EDB Resource Manager によって制御されます。

- リソースグループの作成、変更、削除は、データ定義言語のコマンドで行います。これらのコマンドは、スーパーユーザー権限を持つデータベースユーザーのみが使用できます。

- リソースグループに属するすべてのプロセスの目的の合計消費レベルは、**リソースタイプパラメーター**で定義されます。EDB Resource Manager で現在サポートされているさまざまなタイプのシステムリソースには、さまざまなリソースタイプパラメーターがあります。
- それぞれのリソースタイプパラメーターの設定が異なる複数のリソースグループを作成できるため、リソースグループごとに異なる消費レベルを定義できます。
- EDB Resource Manager は、リソースグループ内のプロセスを調整して、リソース消費量をリソースタイプパラメーターで定義された制限近くに保ちます。リソースグループに設定が定義されたリソースタイプパラメーターが複数ある場合、特定のリソースタイプでは、実際のリソース消費が定義されたリソースタイプパラメーター設定よりも大幅に低い場合があります。これは、EDB Resource Manager が**定義されたリソースタイプ設定を持つすべてのリソースを定義された制限内に維持しようとする**プロセスを抑制するためです。
- 使用可能なリソースグループの定義とそのリソースタイプ設定は、共有のグローバルシステムカタログに格納されます。したがって、リソースグループは、特定の EDB Postgres Advanced Server インスタンス内のすべてのデータベースで利用できます。
- `edb_max_resource_groups` 構成パラメーターは、実行中のプロセスと同時にアクティブにできるリソースグループの最大数を設定します。デフォルト設定は 16 個のリソースグループです。このパラメーターへの変更は、データベースサーバの再起動時に有効になります。

- `SET edb_resource_group TO group_name` コマンドで、現在のプロセスを指定したリソースグループに割り当てます。`RESET edb_resource_group` コマンドまたは `SET edb_resource_group TO DEFAULT` を

使用して、リソースグループから現在のプロセスを削除します。

- デフォルトのリソースグループは、`ALTER ROLE ... SET` コマンドを使用してロールに、または `ALTER DATABASE ... SET` コマンドでデータベースに割り当てることができます。`postgresql.conf` ファイルにパラメーターを設定することにより、データベースサーバインスタンス全体にデフォルトのリソースグループを割り当てることができます。
- データベースサーバインスタンスのバックアップファイルにリソースグループを含める場合は、`pg_dumpall` バックアップユーティリティをデフォルト設定で使います（つまり、`--globals-only`、`--roles-only`、または `--tablespaces-only` オプションを指定しません）。

4.1 リソースグループの作成と管理

このセクションで説明するデータ定義言語コマンドは、リソースグループの作成と管理を提供します。

4.1.1 リソースグループを作成する

`CREATE RESOURCE GROUP` コマンドを使用して、新しいリソースグループを作成します。

```
CREATE RESOURCE GROUP <group_name>;
```

4.1.2 説明

`CREATE RESOURCE GROUP` コマンドは、指定された名前のリソースグループを作成します。次に、`ALTER RESOURCE GROUP` コマンドを使用して、グループにリソース制限を定義できます。リソースグループは、EDB Postgres Advanced Server インスタンス内のすべてのデータベースからアクセスできます。

`CREATE RESOURCE GROUP` コマンドを使用するには、スーパーユーザー権限が必要です。

4.1.3 パラメーター

`group_name`

リソースグループの名前。

4.1.4 例

次の例では、`resgrp_a`、`resgrp_b`、および `resgrp_c` という名前の3つのリソースグループが作成されます。

```
edb=# CREATE RESOURCE GROUP resgrp_a;
CREATE RESOURCE GROUP
edb=# CREATE RESOURCE GROUP resgrp_b;
CREATE RESOURCE GROUP
edb=# CREATE RESOURCE GROUP resgrp_c;
CREATE RESOURCE GROUP
```

次のクエリは、edb_resource_group カタログ内のリソースグループのエントリを表示します。

```
edb=# SELECT * FROM edb_resource_group;
__OUTPUT__
 rgrpname | rgrpcpuratelimit | rgrpdirtyratelimit
-----+-----+-----
 resgrp_a |                0 |                0
 resgrp_b |                0 |                0
 resgrp_c |                0 |                0
(3 rows)
```

4.1.5 ALTER リソースグループ

ALTER RESOURCE GROUP コマンドを使用して、既存のリソースグループの属性を変更します。コマンド構文には3つの形式があります。

最初のフォームは、リソースグループの名前を変更します。

```
ALTER RESOURCE GROUP <group_name> RENAME TO <new_name>;
```

2番目のフォームは、リソースグループにリソースタイプを割り当てます。

```
ALTER RESOURCE GROUP <group_name> SET
  <resource_type> { TO | = } { <value> | DEFAULT };
```

3番目のフォームは、リソースタイプの割り当てをグループ内のデフォルトにリセットします。

```
ALTER RESOURCE GROUP <group_name> RESET <resource_type>;
```

4.1.6 説明

ALTER RESOURCE GROUP コマンドは、既存のリソースグループの特定の属性を変更します。

RENAME TO 句を使用した最初の形式は、既存のリソースグループに新しい名前を割り当てます。

SET resource_type TO 句を使用した2番目の形式は、指定されたリテラル値をリソースタイプに割り当てるか、DEFAULT が指定されたときにリソースタイプをリセットします。リソースタイプを DEFAULT にリセットまたは設定すると、リソースグループにはそのリソースタイプに定義された制限がないことを意味します。

RESET resource_type 句を含む 3 番目のフォームは、前述のようにグループのリソースタイプをリセットします。

ALTER RESOURCE GROUP コマンドを使用するには、スーパーユーザー権限が必要です。

4.1.7 パラメーター

group_name

変更するリソースグループの名前。

new_name

リソース グループに割り当てる新しい名前。

resource_type

使用量の値を設定するリソースのタイプを指定するリソースタイプパラメーター。

value | DEFAULT

value を指定した場合、resource_type に代入するリテラル値。DEFAULT を指定した場合、リソースグループに対して resource_type の割り当てをリセットします。

4.1.8 例

以下は、ALTER RESOURCE GROUP コマンドの例です。

```
edb=# ALTER RESOURCE GROUP resgrp_a RENAME TO newgrp;
ALTER RESOURCE GROUP
edb=# ALTER RESOURCE GROUP resgrp_b SET cpu_rate_limit = .5;
ALTER RESOURCE GROUP
edb=# ALTER RESOURCE GROUP resgrp_b SET dirty_rate_limit = 6144;
ALTER RESOURCE GROUP
edb=# ALTER RESOURCE GROUP resgrp_c RESET cpu_rate_limit;
ALTER RESOURCE GROUP
```

次のクエリは、edb_resource_group カタログのエントリに対する ALTER RESOURCE GROUP コマンドの結果を示しています。

```
edb=# SELECT * FROM edb_resource_group;
__OUTPUT__
 rgrpname | rgrpcpuratelimit | rgrpdirtyratelimit
-----+-----+-----
 newgrp   | 0                 | 0
 resgrp_b | 0.5               | 6144
 resgrp_c | 0                 | 0
(3 rows)
```

4.1.9 リソースグループをドロップ

DROP RESOURCE GROUP コマンドを使用して、リソースグループを削除します。

```
DROP RESOURCE GROUP [ IF EXISTS ] <group_name>;
```

4.1.10 説明

DROP RESOURCE GROUP コマンドは、指定された名前のリソースグループを削除します。

DROP RESOURCE GROUP コマンドを使用するには、スーパーユーザー権限が必要です。

4.1.11 パラメーター

group_name

削除するリソースグループの名前。

IF EXISTS

リソースグループが存在しない場合でもエラーをスローしません。この場合、通知が発行されます。

4.1.12 例

次の例では、リソースグループ **newgrp** を削除します。

```
edb=# DROP RESOURCE GROUP newgrp;
DROP RESOURCE GROUP
```

4.1.13 リソースグループへのプロセスの割り当て

次に示すように、SET edb_resource_group TO group_name コマンドを使用して、指定したリソースグループに現在のプロセスを割り当てます。

```
edb=# SET edb_resource_group TO resgrp_b;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
- - - - -
resgrp_b
(1 row)
```

グループのリソースタイプ設定は、現在のプロセスですぐに有効になります。このコマンドを使用して、現在のプロセスに割り当てられているリソースグループを変更すると、新しく割り当てられたグループのリソースタイプ設定がすぐに有効になります。

既定のリソースグループをロール、データベース、またはデータベースサーバーインスタンス全体に割り当てることにより、プロセスを既定でリソースグループに含めることができます。

`ALTER ROLE ... SET` コマンドを使用して、デフォルトのリソースグループをロールに割り当てることができます。`ALTER ROLE` コマンドの詳細については、次の場所にある PostgreSQL コアドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/sql-alterrole.html>

デフォルトのリソースグループは、`ALTER DATABASE ... SET` コマンドでデータベースに割り当てることができます。`ALTER DATABASE` コマンドの詳細については、次の場所にある PostgreSQL コアドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/sql-alterdatabase.html>

次に示すように、`postgresql.conf` ファイルで `edb_resource_group` 構成パラメーターを設定することにより、データベースサーバーインスタンス全体にデフォルトのリソースグループを割り当てることができます。

```
# - EDB Resource Manager -

# edb_max_resource_groups = 16           # 0-65536 (change requires restart)
edb_resource_group = resgrp_b
```

`postgresql.conf` ファイルの `edb_resource_group` への変更は、データベースサーバーインスタンスで有効になる前に構成ファイルをリロードする必要があります。

4.1.14 リソースグループからのプロセスの削除

`edb_resource_group` を `DEFAULT` に設定するか、`RESET edb_resource_group` を使用して、次のようにリソースグループから現在のプロセスを削除します。

```
edb=# SET edb_resource_group TO DEFAULT;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
  edb_resource_group
-----
(1 row)
```

ロールからデフォルトのリソースグループを削除するには、`ALTER ROLE` コマンドの `ALTER ROLE ... RESET` 形式を使用します。

データベースからデフォルトのリソースグループを削除するには、`ALTER DATABASE` コマンドの `ALTER DATABASE ... RESET` 形式を使用します。

データベースサーバインスタンスからデフォルトのリソースグループを削除するには、`postgresql.conf` ファイルで `edb_resource_group` 構成パラメーターを空の文字列に設定し、構成ファイルをリロードします。

4.1.15 リソースグループのプロセスの監視

リソースグループを作成すると、これらのリソースグループをアクティブに使用しているプロセスの数をビュー `edb_all_resource_groups` から取得できます。

`edb_all_resource_groups` の列は次のとおりです。

- **group_name.** リソースグループの名前。
- **active_processes.** リソースグループ内のアクティブなプロセスの数。
- **cpu_rate_limit.** リソースグループに割り当てられた CPU レート制限リソースタイプの値。
- **per_process_cpu_rate_limit.** リソースグループ内の個々のアクティブなプロセスに適用される CPU レート制限。
- **dirty_rate_limit.** リソースグループに割り当てられたダーティレート制限リソースタイプの値。
- **per_process_dirty_rate_limit.** リソースグループ内の個々のアクティブなプロセスに適用されるダーティレート制限。

注釈: 列 `per_process_cpu_rate_limit` および `per_process_dirty_rate_limit` は、プロセスによって使用される*実際の*リソース消費を示しませんが、*EDB Resource Manager* がリソースグループ内のアクティブなプロセスの数に基づいて個々のプロセスのリソース制限を設定する方法を示します。

以下は、リソースグループ `resgrp_a` に活性プロセスが含まれておらず、リソースグループ `resgrp_b` に 2 つの活性プロセスが含まれており、リソースグループ `resgrp_c` に 1 つの活性プロセスがある場合を示しています。

```
edb=# SELECT * FROM edb_all_resource_groups ORDER BY group_name;
__OUTPUT__
- [ RECORD 1 ]-----+-----
group_name          | resgrp_a
active_processes    | 0
cpu_rate_limit      | 0.5
per_process_cpu_rate_limit |
dirty_rate_limit    | 12288
per_process_dirty_rate_limit |
- [ RECORD 2 ]-----+-----
group_name          | resgrp_b
active_processes    | 2
cpu_rate_limit      | 0.4
per_process_cpu_rate_limit | 0.195694289022895
```

(次のページに続く)

(前のページからの続き)

```

dirty_rate_limit          | 6144
per_process_dirty_rate_limit | 3785.92924684337
- [ RECORD 3 ]-----+-----
group_name                | resgrp_c
active_processes          | 1
cpu_rate_limit            | 0.3
per_process_cpu_rate_limit | 0.292342129631091
dirty_rate_limit          | 3072
per_process_dirty_rate_limit | 3072

```

これらのリソースグループに割り当てられる CPU レートリミットおよびダーティレートリミットの設定は次のとおりです。

```

edb=# SELECT * FROM edb_resource_group;
__OUTPUT__
 rgrpname | rgrpcpuratelimit | rgrpdirtyratelimit
-----+-----+-----
 resgrp_a | 0.5              | 12288
 resgrp_b | 0.4              | 6144
 resgrp_c | 0.3              | 3072
(3 rows)

```

edb_all_resource_groups ビューで、per_process_cpu_rate_limit および per_process_dirty_rate_limit の値は、ほぼ対応する CPU レート制限とダーティレート制限をアクティブなプロセスの数で割ったものであることに注意してください。

4.2 CPU 使用率の調整

リソースグループの CPU 使用率は、cpu_rate_limit リソースタイプパラメーターを設定することにより制御されます。

cpu_rate_limit パラメーターを、グループ内のすべてのプロセスの同時 CPU 使用率が超えない、実時間に対する CPU 時間の割合に設定します。したがって、cpu_rate_limit に割り当てられる値は通常 1 以下である必要があります。

マルチコアシステムでは、cpu_rate_limit を 1 より大きい値に設定することにより、複数の CPU コアに適用できます。たとえば、cpu_rate_limit が 2.0 に設定されている場合、2 つの CPU を 100 % 使用します。

cpu_rate_limit パラメーターの有効範囲は 0 ～ 1.67772e+07 です。0 に設定すると、リソースグループに CPU レート制限が設定されていないことを意味します。

cpu_rate_limit は、100 を掛けると、リソースグループの CPU 使用率と解釈することもできます。

EDB Resource Manager は CPU スロットリングを利用して、グループ内のすべてのプロセスの合計 CPU 使用率を cpu_rate_limit パラメーターで指定された制限内に保ちます。グループ内のプロセスは、定義さ

れた制限を維持するために中断され、短時間スリープモードになる場合があります。このような中断がいつどのように発生するかは、EDB Resource Manager が使用する独自のアルゴリズムによって定義されます。

4.2.1 リソースグループの CPU レート制限の設定

SET `cpu_rate_limit` 句を指定した ALTER RESOURCE GROUP コマンドを使用して、リソースグループの CPU レート制限を設定します。

次の例では、CPU 使用率の制限が `resgrp_a` が 50%、`resgrp_b` が 40%、`resgrp_c` が 30% に設定されています。これは、`resgrp_a` に割り当てられたすべてのプロセスの合計 CPU 使用率が約 50 % に維持されることを意味します。同様に、`resgrp_b` のすべてのプロセスでは、合計した CPU 使用率が約 40 % に保たれます。

```
edb=# ALTER RESOURCE GROUP resgrp_a SET cpu_rate_limit TO .5;
ALTER RESOURCE GROUP
edb=# ALTER RESOURCE GROUP resgrp_b SET cpu_rate_limit TO .4;
ALTER RESOURCE GROUP
edb=# ALTER RESOURCE GROUP resgrp_c SET cpu_rate_limit TO .3;
ALTER RESOURCE GROUP
```

次のクエリは、カタログ内の `cpu_rate_limit` の設定を示しています。

```
edb=# SELECT rgrpname, rgrpcpuratelimit FROM edb_resource_group;
__OUTPUT__
 rgrpname | rgrpcpuratelimit
-----+-----
 resgrp_a |          0.5
 resgrp_b |          0.4
 resgrp_c |          0.3
(3 rows)
```

リソースグループの `cpu_rate_limit` を変更すると、グループに割り当てられた新しいプロセスに影響するだけでなく、グループのメンバーである現在実行中のプロセスも変更の影響を受けます。つまり、`cpu_rate_limit` が .5 から .3 に変更された場合、グループで現在実行中のプロセスが調整され、グループの合計 CPU 使用率が 50 % ではなく 30 % 近くになります。

リソースグループの CPU レート制限を設定する効果を示すために、次の例では、クエリ `SELECT 20000!` で実行される 20000 階乗 ($20000 * 19999 * 19998$ の乗算) の計算を使用します。psql コマンドラインユーティリティで実行します。

これらの例では、前のクエリに示されている CPU レート制限設定のリソースグループが使用されます。

4.2.2 例：単一グループ内の単一プロセス

以下は、現在のプロセスがリソースグループ `resgrp_b` を使用するように設定されていることを示しています。階乗計算が開始されます。

```
edb=# SET edb_resource_group TO resgrp_b;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
resgrp_b
(1 row)
edb=# SELECT 20000!;
```

2 番目のセッションでは、Linux の `top` コマンドを使用して、%CPU 列に示されている CPU 使用率を表示します。以下は、`top` コマンドの出力が定期的に変更されるための任意の時点でのスナップショットです。

```
$ top
top - 16:37:03 up 4:15, 7 users, load average: 0.49, 0.20, 0.38
Tasks: 202 total, 1 running, 201 sleeping, 0 stopped, 0 zombie
Cpu(s): 42.7%us, 2.3%sy, 0.0%ni, 55.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 1025624k total, 791160k used, 234464k free, 23400k buffers
Swap: 103420k total, 13404k used, 90016k free, 373504k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
28915 enterpri  20   0 195m 5900 4212 S 39.9   0.6   3:36.98 edb-postgres
1033  root       20   0 171m  77m 2960 S   1.0   7.8   3:43.96 Xorg
3040  user       20   0 278m  22m 14m S   1.0   2.2   3:41.72 knotify4
.
.
.
```

階乗計算を実行する `psql` セッションは、`COMMAND` 列の下に `edb-postgres` が表示されます。%CPU 列の下に表示されるセッションの CPU 使用率は 39.9 を示しています。これは、リソースグループ `resgrp_b` に設定されている 40% の CPU 制限に近いです。

対照的に、`psql` セッションをリソースグループから削除して、階乗計算を再度実行すると、CPU 使用率がはるかに高くなります。

```
edb=# SET edb_resource_group TO DEFAULT;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
```

(次のページに続く)

(前のページからの続き)

(1 row)

edb=# SELECT 20000!;

edb-postgres の%CPU 列の下 の CPU 使用率は 93.6 になり、プロセスがリソースグループの一部であったときの 39.9 を大幅に上回っています。

```
$ top
top - 16:43:03 up 4:21, 7 users, load average: 0.66, 0.33, 0.37
Tasks: 202 total, 5 running, 197 sleeping, 0 stopped, 0 zombie
Cpu(s): 96.7%us, 3.3%sy, 0.0%ni, 0.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 1025624k total, 791228k used, 234396k free, 23560k buffers
Swap: 103420k total, 13404k used, 90016k free, 373508k cached

  PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
28915 enterpri 20 0 195m 5900 4212 R 93.6 0.6 5:01.56 edb-postgres
1033 root 20 0 171m 77m 2960 S 1.0 7.8 3:48.15 Xorg
2907 user 20 0 98.7m 11m 9100 S 0.3 1.2 0:46.51 vmware-user-lo
.
```

4.2.3 例：単一グループ内の複数のプロセス

前述のように、CPU レート制限はリソースグループ内のすべてのプロセスの集計に適用されます。この概念を次の例で説明します。

階乗計算は、2 つの個別の psql セッションで同時に実行されます。各セッションは、`cpu_rate_limit` が 4 (CPU 使用率 40 %) に設定されているリソースグループ `resgrp_b` に追加されています。

セッション 1：

```
edb=# SET edb_resource_group TO resgrp_b;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
resgrp_b
(1 row)

edb=# SELECT 20000!;
```

セッション 2：

```

edb=# SET edb_resource_group TO resgrp_b;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
resgrp_b
(1 row)

edb=# SELECT 20000!;
```

3 番目のセッションは CPU 使用率を監視します。

```

$ top
top - 16:53:03 up 4:31, 7 users, load average: 0.31, 0.19, 0.27
Tasks: 202 total, 1 running, 201 sleeping, 0 stopped, 0 zombie
Cpu(s): 41.2%us, 3.0%sy, 0.0%ni, 55.8%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 1025624k total, 792020k used, 233604k free, 23844k buffers
Swap: 103420k total, 13404k used, 90016k free, 373508k cached

  PID USER PR NI VIRT RES  SHR S %CPU %MEM TIME+  COMMAND
29857 enterpri 20  0 195m 4708 3312 S 19.9  0.5  0:57.35 edb-postgres
28915 enterpri 20  0 195m 5900 4212 S 19.6  0.6  5:35.49 edb-postgres
 3040 user 20  0 278m 22m 14m S 1.0  2.2  3:54.99 knotify4
 1033 root 20  0 171m 78m 2960 S 0.3  7.8  3:55.71 Xorg
.
.
.
```

%CPU の値が 19.9 と 19.6 である edb-postgres という名前の 2 つのプロセスがあり、それらの合計はリソースグループ resgrp_b に設定された 40% の CPU 使用率に近いです。

次のコマンドシーケンスは、0.5 秒の時間間隔でサンプリングされたすべての edb-postgres プロセスの合計を表示します。これは、EDB Resource Manager がプロセスを調整してリソースグループの合計 CPU 使用率を 40 % 近くに保つときに、リソースグループ内のプロセスの合計 CPU 使用率がどのように変化するかを示しています。

```

$ while [[ 1 -eq 1 ]]; do top -d0.5 -b -n2 | grep edb-postgres | awk { SUM
+= $9 } END { print SUM / 2 }; done
37.2
39.1
38.9
38.3
44.7
39.2
42.5
```

(次のページに続く)

(前のページからの続き)

```

39.1
39.2
39.2
41
42.85
46.1
.
.
.

```

4.2.4 例：複数のグループの複数のプロセス

この例では、前の2つのセッションとともに2つの追加の `psql` セッションが使用されます。3番目と4番目のセッションは、リソースグループ `resgrp_c` 内で `cpu_rate_limit` を.3（30 %のCPU使用率）で実行します。

セッション3：

```

edb=# SET edb_resource_group TO resgrp_c;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
resgrp_c
(1 row)

edb=# SELECT 200000!;

```

セッション4：

```

edb=# SET edb_resource_group TO resgrp_c;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
resgrp_c
(1 row)

edb=# SELECT 200000!;

```

`top` コマンドは、次の出力を表示します。

```
$ top
top - 17:45:09 up 5:23, 8 users, load average: 0.47, 0.17, 0.26
Tasks: 203 total, 4 running, 199 sleeping, 0 stopped, 0 zombie
Cpu(s): 70.2%us, 0.0%sy, 0.0%ni, 29.8%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 1025624k total, 806140k used, 219484k free, 25296k buffers
Swap: 103420k total, 13404k used, 90016k free, 374092k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
29857	enterpri	20	0	195m	4820	3324	S	19.9	0.5	4:25.02	edb-postgres
28915	enterpri	20	0	195m	5900	4212	R	19.6	0.6	9:07.50	edb-postgres
29023	enterpri	20	0	195m	4744	3248	R	16.3	0.5	4:01.73	edb-postgres
11019	enterpri	20	0	195m	4120	2764	R	15.3	0.4	0:04.92	edb-postgres
2907	user	20	0	98.7m	12m	9112	S	1.3	1.2	0:56.54	vmware-user-lo
3040	user	20	0	278m	22m	14m	S	1.3	2.2	4:38.73	knotify4

使用中の2つのリソースグループのCPU使用率は40%と30%です。最初の2つの edb-postgres プロセスの%CPU 列の合計は39.5（resgrp_b の制限である約40%）であり、3番目と4番目の edb-postgres プロセスの%CPU 列の合計は31.6（約30%です） resgrp_c の制限）。

これらのプロセスが属する2つのリソースグループのCPU使用率の制限の合計は70%です。次の出力は、4つのプロセスの合計が約70%であることを示しています。

```
$ while [[ 1 -eq 1 ]]; do top -d0.5 -b -n2 | grep edb-postgres | awk { SUM
+= $9 } END { print SUM / 2 }; done
61.8
76.4
72.6
69.55
64.55
79.95
68.55
71.25
74.85
62
74.85
76.9
72.4
65.9
74.9
68.25
```

これに対して、3つのセッションが処理中で、2つのセッションが resgrp_b に残っているが、3番目のセッションがどのリソースグループにも属していない場合、top コマンドは次の出力を表示します。

```
$ top
top - 17:24:55 up 5:03, 7 users, load average: 1.00, 0.41, 0.38
```

(次のページに続く)

(前のページからの続き)

```
Tasks: 199 total, 3 running, 196 sleeping, 0 stopped, 0 zombie
Cpu(s): 99.7%us, 0.3%sy, 0.0%ni, 0.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0
Mem: 1025624k total, 797692k used, 227932k free, 24724k buffers
Swap: 103420k total, 13404k used, 90016k free, 374068k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
29023	enterpri	20	0	195m	4744	3248	R	58.6	0.5	2:53.75	edb-postgres
28915	enterpri	20	0	195m	5900	4212	S	18.9	0.6	7:58.45	edb-postgres
29857	enterpri	20	0	195m	4820	3324	S	18.9	0.5	3:14.85	edb-postgres
1033	root	20	0	174m	81m	2960	S	1.7	8.2	4:26.50	Xorg
3040	user	20	0	278m	22m	14m	S	1.0	2.2	4:21.20	knotify4

CPU 使用率が 40 %に制限されているリソースグループに属する 2 番目と 3 番目の edb-postgres プロセスの合計 CPU 使用率は 37.8 です。ただし、最初の edb-postgres プロセスはリソースグループ内がないため 58.6 %の CPU 使用率であり、基本的にシステム上の残りの使用可能な CPU リソースを利用します。

同様に、次の出力は、セッションの 1 つで CPU 使用率に制限が設定されていないため、3 つすべてのセッションの合計が約 95%であることを示しています。

```
$ while [[ 1 -eq 1 ]]; do top -d0.5 -b -n2 | grep edb-postgres | awk { SUM
+= $9 } END { print SUM / 2 }; done
96
90.35
92.55
96.4
94.1
90.7
95.7
95.45
93.65
87.95
96.75
94.25
95.45
97.35
92.9
96.05
96.25
94.95
.
.
.
```

4.3 ダーティバッファスロットリング

共有バッファへの書き込みは、`dirty_rate_limit` リソースタイプパラメーターを設定することによって制御されます。

`dirty_rate_limit` パラメーターを、グループ内のすべてのプロセスが共有バッファに書き込むまたは「ダーティー」する合計レートの 1 秒あたりのキロバイト数に設定します。設定例は、毎秒 3072 キロバイトです。

`dirty_rate_limit` パラメーターの有効範囲は 0 ～ 1.67772e+07 です。0 に設定すると、リソースグループにダーティレート制限が設定されていないことを意味します。

EDB Resource Manager はダーティバッファスロットリングを利用して、グループ内のすべてのプロセスの共有バッファ書き込み速度を `dirty_rate_limit` パラメーターで指定された制限近くに保ちます。グループ内のプロセスは、定義された制限を維持するために中断され、短時間スリープモードになる場合があります。このような中断がいつどのように発生するかは、EDB Resource Manager が使用する独自のアルゴリズムによって定義されます。

4.3.1 リソースグループのダーティレート制限の設定

`SET dirty_rate_limit` 句を指定した `ALTER RESOURCE GROUP` コマンドを使用して、リソースグループのダーティレート制限を設定します。

次の例では、ダーティレート制限は、`resgrp_a` が毎秒 12288 キロバイト、`resgrp_b` が毎秒 6144 キロバイト、および `resgrp_c` が毎秒 3072 キロバイトに設定されています。これは、`resgrp_a` に割り当てられたすべてのプロセスの共有バッファへの合計書き込み速度が約 12288 キロバイトに維持されることを意味します。同様に、`resgrp_b` のすべてのプロセスでは、共有バッファへの合計書き込み速度は毎秒約 6144 キロバイトに維持されます。

```
edb=# ALTER RESOURCE GROUP resgrp_a SET dirty_rate_limit TO 12288;
ALTER RESOURCE GROUP
edb=# ALTER RESOURCE GROUP resgrp_b SET dirty_rate_limit TO 6144;
ALTER RESOURCE GROUP
edb=# ALTER RESOURCE GROUP resgrp_c SET dirty_rate_limit TO 3072;
ALTER RESOURCE GROUP
```

次のクエリは、カタログ内の `dirty_rate_limit` の設定を示しています。

```
edb=# SELECT rgrpname, rgrpdirtyratelimit FROM edb_resource_group;
__OUTPUT__
 rgrpname | rgrpdirtyratelimit
-----+-----
 resgrp_a |          12288
 resgrp_b |           6144
 resgrp_c |           3072
(3 rows)
```

リソースグループの `dirty_rate_limit` を変更すると、グループに割り当てられた新しいプロセスに影響するだけでなく、グループのメンバーである現在実行中のプロセスも変更の影響を受けます。つまり、`dirty_rate_limit` が 12288 から 3072 に変更された場合、グループで現在実行中のプロセスは下方に調整されるため、グループの総ダーティレートは 12288 キロバイト/秒ではなく 3072 キロバイト近くになります。

リソースグループにダーティレート制限を設定する効果を示すために、次の例では、集中的な I/O 操作について次の表を使用します。

```
CREATE TABLE t1 (c1 INTEGER, c2 CHARACTER(500)) WITH (FILLFACTOR = 10);
```

`FILLFACTOR = 10` 句を指定すると、`INSERT` コマンドでページごとに 10%のみ行がパックされます。これにより、これらの例では、ダーティな共有ブロックのサンプリングが多くなります。

`pg_stat_statements` モジュールは、SQL コマンドによって汚された共有バッファブロックの数と、コマンドの実行にかかった時間を表示するために使用されます。これにより、SQL コマンドの実際の 1 秒あたりの書き込み速度を計算し、リソースグループに設定されたダーティレート制限と比較するための情報が提供されます。

`pg_stat_statements` モジュールを使用するには、次の手順を実行します。

ステップ 1: `postgresql.conf` ファイルで、次のように `$libdir/pg_stat_statements` を `shared_preload_libraries` 構成パラメーターに追加します。

```
shared_preload_libraries = $libdir/dbms_pipe,$libdir/edb_gen,$libdir/pg_stat_
→statements
```

ステップ 2: データベースサーバーを再起動します。

ステップ 3: `CREATE EXTENSION` コマンドを使用して、`pg_stat_statements` モジュールの作成を完了します。

```
edb=# CREATE EXTENSION pg_stat_statements SCHEMA public;
CREATE EXTENSION
```

各例を明確にするために、`pg_stat_statements_reset()` 関数を使用して `pg_stat_statements` ビューをクリアします。

これらの例では、前のクエリに示されているダーティレート制限設定のリソースグループが使用されます。

4.3.2 例：単一グループ内の単一プロセス

次のコマンドシーケンスは、テーブル `t1` の作成を示しています。現在のプロセスはリソースグループ `resgrp_b` を使用するよう設定されています。`pg_stat_statements` ビューは、`pg_stat_statements_reset()` 関数を実行してクリアされます。

最後に、`INSERT` コマンドは、テーブルにデータを入力し、約 10,000 ブロックをダーティにするための一連の整数を生成します。

```

edb=# CREATE TABLE t1 (c1 INTEGER, c2 CHARACTER(500)) WITH (FILLFACTOR = 10);
CREATE TABLE
edb=# SET edb_resource_group TO resgrp_b;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
resgrp_b
(1 row)

```

```

edb=# SELECT pg_stat_statements_reset();
__OUTPUT__
pg_stat_statements_reset
-----

(1 row)

edb=# INSERT INTO t1 VALUES (generate_series (1,10000), aaa);
INSERT 0 10000

```

次に、INSERT コマンドの結果を示します。

```

edb=# SELECT query, rows, total_time, shared_blks_dirtied FROM
pg_stat_statements;
__OUTPUT__
- [ RECORD 1 ]-----+-----
query              | INSERT INTO t1 VALUES (generate_series (?,?), ?);
rows                | 10000
total_time          | 13496.184
shared_blks_dirtied | 10003

```

実際のダーティ率は次のように計算されます。

- ミリ秒（ミリ秒）あたりにダーティされたブロックの数は 10003 ブロック / 13496.184 ミリ秒で、* 0.74117247 ミリ秒*になります。
- 結果に 1000 を掛けて、1 秒あたりにダーティな共有ブロックの数を算出します（1 秒= 1000 ミリ秒）。これは、* 1 秒あたり 741.17247 ブロック*になります。
- 結果に 8.192 を掛けて、1 秒あたりにダーティになったキロバイト数（1 ブロック= 8.192 キロバイト）を出します。これは、約 6072 キロバイト/秒になります。

6072 キロバイト/秒の実際のダーティレートは、リソースグループのダーティレート制限である 6144 キロバイトに近いことに注意してください。

対照的に、プロセスがリソースグループに属せずに再び手順が繰り返される場合、ダーティバッファの割合ははるかに高くなります。

```
edb=# CREATE TABLE t1 (c1 INTEGER, c2 CHARACTER(500)) WITH (FILLFACTOR = 10);
CREATE TABLE
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
(1 row)
```

```
edb=# SELECT pg_stat_statements_reset();
__OUTPUT__
pg_stat_statements_reset
-----
(1 row)

edb=# INSERT INTO t1 VALUES (generate_series (1,10000), aaa);
INSERT 0 10000
```

リソースグループを使用しない場合の INSERT コマンドの結果を次に示します。

```
edb=# SELECT query, rows, total_time, shared_blks_dirtied FROM
pg_stat_statements;
__OUTPUT__
- [ RECORD 1 ]-----+-----
query              | INSERT INTO t1 VALUES (generate_series (?,?), ?);
rows                | 10000
total_time          | 2432.165
shared_blks_dirtied | 10003
```

まず、合計時間は、ダーティレート制限が 6144 キロバイト/秒に設定されたリソースグループを使用した場合の 13496.184 ミリ秒と比較して、わずか 2432.165 ミリ秒であることに注意してください。

リソースグループを使用しない場合の実際のダーティ率は、次のように計算されます。

- ミリ秒（ミリ秒）あたりのダーティブロック数は 10003 ブロック/2432.165 ミリ秒で、* 1 ミリ秒あたり 4.112797 ブロック*になります。
- 結果に 1000 を掛けて、1 秒あたりにダーティな共有ブロックの数を算出します（1 秒= 1000 ミリ秒）。これは、* 1 秒あたり 4112.797 ブロック*になります。
- 結果に 8.192 を掛けて、1 秒あたりにダーティになったキロバイト数（1 ブロック= 8.192 キロバイト）を出します。これは、約 33692 キロバイト/秒になります。

33692 キロバイト/秒の実際のダーティレートは、6144 キロバイト/秒のダーティレート制限を持つリソースグループを使用した場合よりも大幅に高いことに注意してください。

4.3.3 例：単一グループ内の複数のプロセス

前述のように、ダーティレート制限はリソースグループ内のすべてのプロセスの集計に適用されます。この概念を次の例で説明します。

この例では、挿入は2つの個別の psql セッションで2つの異なるテーブルで同時に実行されます。各セッションは、dirty_rate_limit が 6144 キロバイト/秒に設定されているリソースグループ resgrp_b に追加されています。

セッション 1：

```
edb=# CREATE TABLE t1 (c1 INTEGER, c2 CHARACTER(500)) WITH (FILLFACTOR = 10);
CREATE TABLE
edb=# SET edb_resource_group TO resgrp_b;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
resgrp_b
(1 row)

edb=# INSERT INTO t1 VALUES (generate_series (1,10000), aaa);
INSERT 0 10000
```

セッション 2：

```
edb=# CREATE TABLE t2 (c1 INTEGER, c2 CHARACTER(500)) WITH (FILLFACTOR = 10);
CREATE TABLE
edb=# SET edb_resource_group TO resgrp_b;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----
resgrp_b
(1 row)
```

```
edb=# SELECT pg_stat_statements_reset();
__OUTPUT__
pg_stat_statements_reset
-----
(1 row)

edb=# INSERT INTO t2 VALUES (generate_series (1,10000), aaa);
INSERT 0 10000
```

注釈: セッション 1 およびセッション 2 の `INSERT` コマンドは、セッション 2 の `SELECT pg\_stat\_statements\_reset()` コマンドが実行された後に開始されました。

次に、2つのセッションでの INSERT コマンドの結果を示します。RECORD 3 は、セッション 1 の結果を示しています。RECORD 2 は、セッション 2 の結果を示しています。

```
edb=# SELECT query, rows, total_time, shared_blks_dirtied FROM
pg_stat_statements;
__OUTPUT__
- [ RECORD 1 ]-----+-----
query          | SELECT pg_stat_statements_reset();
rows           | 1
total_time     | 0.43
shared_blks_dirtied | 0
- [ RECORD 2 ]-----+-----
query          | INSERT INTO t2 VALUES (generate_series (?,?), ?);
rows           | 10000
total_time     | 30591.551
shared_blks_dirtied | 10003
- [ RECORD 3 ]-----+-----
query          | INSERT INTO t1 VALUES (generate_series (?,?), ?);
rows           | 10000
total_time     | 33215.334
shared_blks_dirtied | 10003
```

まず、セッション 1 の合計時間は 33215.334 ミリ秒、セッション 2 は 30591.551 ミリ秒でした。最初の例に示すように、同じリソースグループで 1つのセッションのみがアクティブだった場合、時間は 13496.184 ミリ秒でした。したがって、リソースグループ内のアクティブなプロセスが多いほど、グループ内のアクティブなプロセスごとにダーティレートが低下します。これは、以下の計算に示されています。

セッション 1 の実際のダーティ率は次のように計算されます。

- ミリ秒（ミリ秒）あたりにダーティされたブロックの数は 10003 ブロック / 33215.334 ミリ秒で、* 1 ミリ秒あたり 0.30115609 ブロック*になります。
- 結果に 1000 を掛けて、1 秒あたりにダーティな共有ブロックの数（1 秒 = 1000 ミリ秒）を取得します。これは、* 301.15609 ブロック/秒*になります。
- 結果に 8.192 を掛けて、1 秒あたりにダーティになったキロバイト数（1 ブロック = 8.192 キロバイト）を出します。これは、約 2467 キロバイト/秒になります。

セッション 2 の実際のダーティ率は次のように計算されます。

- ミリ秒（ミリ秒）あたりにダーティされたブロックの数は 10003 ブロック / 30591.551 ミリ秒で、* 0.32698571 ミリ秒*になります。
- 結果に 1000 を掛けて、1 秒あたりにダーティな共有ブロックの数を算出します（1 秒 = 1000 ミリ秒）。これは、* 1 秒あたり 326.98571 ブロック*になります。

- 結果に 8.192 を掛けて、1 秒あたりにダーティになったキロバイト数（1 ブロック= 8.192 キロバイト）を出します。これは、約 2679 キロバイト/秒になります。

セッション 1（2467 キロバイト/秒）とセッション 2（2679 キロバイト/秒）のダーティレートを合計すると、毎秒 5146 キロバイトになり、リソースグループに設定されたダーティレート制限（6144 キロバイト/秒）を下回ります。

4.3.4 例：複数のグループの複数のプロセス

この例では、前の 2 つのセッションとともに 2 つの追加の `psql` セッションが使用されます。3 番目と 4 番目のセッションは、毎秒 3072 キロバイトの `dirty_rate_limit` を使用して、リソースグループ `resgrp_c` で同じ `INSERT` コマンドを実行します。

セッション 1 と 2 は、前の例に示すように、毎秒 6144 キロバイトのリソースグループ `resgrp_b` を使用して繰り返されます。

セッション 3：

```
edb=# CREATE TABLE t3 (c1 INTEGER, c2 CHARACTER(500)) WITH (FILLFACTOR = 10);
CREATE TABLE
edb=# SET edb_resource_group TO resgrp_c;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----

resgrp_c
(1 row)

edb=# INSERT INTO t3 VALUES (generate_series (1,10000), aaa);
INSERT 0 10000
```

セッション 4：

```
edb=# CREATE TABLE t4 (c1 INTEGER, c2 CHARACTER(500)) WITH (FILLFACTOR = 10);
CREATE TABLE
edb=# SET edb_resource_group TO resgrp_c;
SET
edb=# SHOW edb_resource_group;
__OUTPUT__
edb_resource_group
-----

resgrp_c
(1 row)
```

```
edb=# SELECT pg_stat_statements_reset();
__OUTPUT__
pg_stat_statements_reset
-----

(1 row)

edb=# INSERT INTO t4 VALUES (generate_series (1,10000), aaa);
INSERT 0 10000
```

注釈: 4つのセッションすべての`INSERT` コマンドは、セッション4の`SELECT pg\_stat\_statements\_reset()` コマンドが実行された後に開始されました。

次に、4つのセッションでの INSERT コマンドの結果を示します。

RECORD 3 は、セッション1の結果を示しています。RECORD 2 は、セッション2の結果を示しています。

RECORD 4 は、セッション3の結果を示しています。RECORD 5 は、セッション4の結果を示しています。

```
edb=# SELECT query, rows, total_time, shared_blks_dirtied FROM
pg_stat_statements;
__OUTPUT__
- [ RECORD 1 ]-----+-----
query          | SELECT pg_stat_statements_reset();
rows           | 1
total_time     | 0.467
shared_blks_dirtied | 0
- [ RECORD 2 ]-----+-----
query          | INSERT INTO t2 VALUES (generate_series (?,?), ?);
rows           | 10000
total_time     | 31343.458
shared_blks_dirtied | 10003
- [ RECORD 3 ]-----+-----
query          | INSERT INTO t1 VALUES (generate_series (?,?), ?);
rows           | 10000
total_time     | 28407.435
shared_blks_dirtied | 10003
- [ RECORD 4 ]-----+-----
query          | INSERT INTO t3 VALUES (generate_series (?,?), ?);
rows           | 10000
total_time     | 52727.846
shared_blks_dirtied | 10003
- [ RECORD 5 ]-----+-----
query          | INSERT INTO t4 VALUES (generate_series (?,?), ?);
rows           | 10000
```

(次のページに続く)

(前のページからの続き)

total_time	56063.697
shared_blks_dirtied	10003

最初に、セッション 1 (28407.435) とセッション 2 (31343.458) の時間は、セッション 3 (52727.846) とセッションの時間と比較して、dirty_rate_limit が 6144 に設定されている同じリソースグループにあることに注意してください 4 (56063.697)、これらは dirty_rate_limit が 3072 に設定されたリソースグループにあります。後者のグループにはより遅いダーティレート制限があるため、セッション 3 および 4 の場合と同様に、予想される処理時間が長くなります。

セッション 1 の実際のダーティ率は次のように計算されます。

- ミリ秒 (ミリ秒) あたりにダーティされたブロックの数は 10003 ブロック/28407.435 ミリ秒で、* 1 ミリ秒あたり 0.35212612 ブロック*になります。
- 結果に 1000 を掛けて、1 秒あたりにダーティになる共有ブロックの数 (1 秒 = 1000 ミリ秒) を取得します。これは、* 352.12612 ブロック/秒*になります。
- 結果に 8.192 を掛けて、1 秒あたりにダーティになったキロバイト数 (1 ブロック = 8.192 キロバイト) を出します。これは、約 2885 キロバイト/秒になります。

セッション 2 の実際のダーティ率は次のように計算されます。

- ミリ秒 (ミリ秒) あたりにダーティされたブロックの数は 10003 ブロック/ 31343.458 ミリ秒で、* 0.31914156 ブロック/ミリ秒*。
- 結果に 1000 を掛けて、1 秒あたりにダーティになる共有ブロックの数 (1 秒 = 1000 ミリ秒) を取得します。これは、* 319.14156 ブロック/秒*になります。
- 結果に 8.192 を掛けて、1 秒あたりにダーティになったキロバイト数 (1 ブロック = 8.192 キロバイト) を出します。これは、約 2614 キロバイト/秒になります。

セッション 1 (2885 キロバイト/秒) とセッション 2 (2614 キロバイト/秒) のダーティレートを合わせた結果は、毎秒 5499 キロバイトであり、リソースグループに設定されたダーティレート制限 (6144 キロバイト/秒) に近いです。

セッション 3 の実際のダーティ率は次のように計算されます。

- ミリ秒 (ミリ秒) あたりのダーティブロック数は 10003 ブロック/ 52727.846 ミリ秒で、* 1 ミリ秒あたり 0.18971001 ブロック*になります。
- 結果に 1000 を掛けて、1 秒あたりにダーティになる共有ブロックの数 (1 秒 = 1000 ミリ秒) を取得します。これは、* 189.71001 ブロック/秒*になります。
- 結果に 8.192 を掛けて、1 秒あたりにダーティになったキロバイト数 (1 ブロック = 8.192 キロバイト) を出します。これは、約* 1554 キロバイト/秒*になります。

セッション 4 の実際のダーティ率は次のように計算されます。

- ミリ秒 (ミリ秒) あたりにダーティされたブロックの数は 10003 ブロック/ 56063.697 ミリ秒で、* 0.17842205 ブロック/ミリ秒*。
- 結果に 1000 を掛けて、1 秒あたりにダーティな共有ブロックの数を算出します (1 秒 = 1000 ミリ秒)。これは、* 178.42205 ブロック/秒*になります。

- 結果に 8.192 を掛けて、1 秒あたりにダーティになったキロバイト数 (1 ブロック= 8.192 キロバイト) を出します。これは、約 **1462 キロバイト/秒**になります。

セッション 3 (1554 キロバイト/秒) とセッション 4 (1462 キロバイト/秒) の合計ダーティレートは 3016 キロバイト/秒であり、これはリソースグループに設定されたダーティレート制限 (3072 キロバイト/秒) に近いです。

したがって、これは、EDB Resource Manager がそのグループ内のアクティブなプロセスの合計ダーティレートを各グループに設定されたダーティレート制限に近づける方法を示しています。

4.4 システムカタログ

このセクションでは、EDB Resource Manager が使用するリソースグループ情報を格納するシステムカタログについて説明します。

4.4.1 edb_all_resource_groups

次の表に、`edb_all_resource_groups` カタログで使用可能な情報をリストします。

Column	Type	Description
<code>group_name</code>	<code>name</code>	リソース グループの名前。
<code>active_processes</code>	<code>integer</code>	リソースグループ内の現在アクティブなプロセスの数。
<code>cpu_rate_limit</code>	<code>float8</code>	リソースグループの最大 CPU レート制限。`0`は無制限を意味します。
<code>per_process_cpu_rate_limit</code>	<code>float8</code>	リソースグループ内の現在アクティブなプロセスごとの最大 CPU レート制限。
<code>dirty_rate_limit</code>	<code>float8</code>	リソースグループの最大ダーティレート制限。`0`は無制限を意味します。
<code>per_process_dirty_rate_limit</code>	<code>float8</code>	リソースグループ内の現在アクティブなプロセスごとの最大ダーティレート制限。

4.4.2 edb_resource_group

次の表に、 edb_resource_group カタログで使用可能な情報をリストします。

Column	Type	Description
<i>rgrpname</i>	<i>name</i>	リソース グループの名前。
<i>rgrpcpuratelimt</i>	<i>float8</i>	リソースグループの最大 CPU レート制限。`0`は無制限を意味します。
<i>rgrpdirtyratelimt</i>	<i>float8</i>	リソースグループの最大ダーティレート制限。`0`は無制限を意味します。

第5章 libpq C library

libpq は、EDB Postgres Advanced Server への C アプリケーションプログラマ用インターフェイスです。libpq は、クライアントプログラムが EDB Postgres Advanced Server にクエリを渡し、これらのクエリの結果を受信できるようにするライブラリ関数のセットです。

libpq は、C++、Perl、Python、Tcl、ECPG 用に作成されたものを含む、他のいくつかの EDB アプリケーションインターフェイスの基になるエンジンでもあります。したがって、これらのパッケージのいずれかを使用する場合、libpq の動作のいくつかの側面はユーザーにとって重要です。

libpq を使用するクライアントプログラムは、ヘッダーファイル `libpq-fe.h` をインクルードし、libpq ライブラリとリンクする必要があります。

5.1 EDB SPL で libpq を使用する

EDB SPL 言語は libpq インターフェイスライブラリで使用でき、以下をサポートします。

- プロシージャ、ファンクション、パッケージ
- 準備されたステートメント
- REFCURSORS
- 静的カーソル
- structs および typedefs
- 配列
- DML および DDL 操作
- IN /OUT /IN OUT パラメーター

5.2 REFCURSOR のサポート

以前のリリースでは、EDB Postgres Advanced Server は次の libpq 関数を介して REFCURSOR のサポートを提供しました。これらの関数は非推奨と見なす必要があります。

- PQCursorResult()
- PQgetCursorResult()
- PQnCursor()

PQexec() および PQgetvalue() を使用して、SPL（または PL/pgSQL）関数によって返された REFCURSOR を取得できるようになりました。REFCURSOR は、カーソルの名前を示すスル終了文字列の形式で返されます。カーソルの名前を取得したら、1 つ以上の FETCH ステートメントを実行して、カーソルを介して公開された値を取得できます。

次のサンプルには、実際のクライアントアプリケーションに必要なエラー処理コードが含まれていません。

5.3 単一の REFCURSOR を返す

次の例は、REFCURSOR 型の値を返す SPL ファンクションを示しています。

```
CREATE OR REPLACE FUNCTION getEmployees(p_deptno NUMERIC)
RETURN REFCURSOR AS
    result REFCURSOR;
BEGIN
    OPEN result FOR SELECT * FROM emp WHERE deptno = p_deptno;

    RETURN result;
END;
```

この関数は、単一のパラメーター p_deptno を想定し、OPEN ステートメントに示されている SELECT クエリの結果セットを保持する REFCURSOR を返します。OPEN ステートメントは、クエリを実行し、結果セットをカーソルに保存します。サーバーはそのカーソルの名前を作成し、名前を変数（result という名前）に保存します。次に、関数はカーソルの名前を呼び出し元に返します。

libpq を使用して C クライアントからこの関数を呼び出すには、PQexec() および PQgetvalue() を使用できます。

```
# include <stdio.h>
# include <stdlib.h>
# include "libpq-fe.h"

static void fetchAllRows(PGconn *conn,
                        const char *cursorName,
                        const char *description);
static void fail(PGconn *conn, const char *msg);

int
main(int argc, char *argv[])
{
    PGconn    *conn = PQconnectdb(argv[1]);
    PGresult  *result;

    if (PQstatus(conn) != CONNECTION_OK)
        fail(conn, PQerrorMessage(conn));
```

(次のページに続く)

(前のページからの続き)

```

result = PQexec(conn, "BEGIN TRANSACTION");

if (PQresultStatus(result) != PGRES_COMMAND_OK)
    fail(conn, PQerrorMessage(conn));

PQclear(result);

result = PQexec(conn, "SELECT * FROM getEmployees(10)");

if (PQresultStatus(result) != PGRES_TUPLES_OK)
    fail(conn, PQerrorMessage(conn));

fetchAllRows(conn, PQgetvalue(result, 0, 0), "employees");

PQclear(result);

PQexec(conn, "COMMIT");

PQfinish(conn);

exit(0);
}

static void
fetchAllRows(PGconn *conn,
             const char *cursorName,
             const char *description)
{
    size_t commandLength = strlen("FETCH ALL FROM ") +
                          strlen(cursorName) + 3;

    char *commandText = malloc(commandLength);
    PGresult *result;
    int row;

    sprintf(commandText, "FETCH ALL FROM \"%s\"", cursorName);

    result = PQexec(conn, commandText);

    if (PQresultStatus(result) != PGRES_TUPLES_OK)
        fail(conn, PQerrorMessage(conn));

    printf("-- %s --\n", description);

    for (row = 0; row < PQntuples(result); row++)

```

(次のページに続く)

```

{
    const char *delimiter = "\t";
    int col;

    for (col = 0; col < PQnfields(result); col++)
    {
        printf("%s%s", delimiter, PQgetvalue(result, row, col));
        delimiter = ",";
    }

    printf("\n");
}

PQclear(result);
free(commandText);
}

static void
fail(PGconn *conn, const char *msg)
{
    fprintf(stderr, "%s\n", msg);

    if (conn != NULL)
        PQfinish(conn);

    exit(-1);
}

```

コードサンプルには、`getEmployees()` ファンクションを呼び出し、部門 10 のすべての従業員を含む結果セットを返すコード行が含まれています。

```
result = PQexec(conn, "SELECT * FROM getEmployees(10)");
```

`PQexec()` 関数は、C プログラムに結果セットハンドルを返します。結果セットには値が 1 つだけ含まれます。その値は、`getEmployees()` によって返されるカーソルの名前です。

カーソルの名前を取得したら、SQL `FETCH` ステートメントを使用してそのカーソル内の行を取得できます。関数 `fetchAllRows()` は、`FETCH ALL` ステートメントを構築し、そのステートメントを実行して、`FETCH ALL` ステートメントの結果セットを出力します。

このプログラムの出力を以下に示します。

```

- - employees --
7782,CLARK,MANAGER,7839,09-JUN-81 00:00:00,2450.00,,10
7839,KING,PRESIDENT,,17-NOV-81 00:00:00,5000.00,,10
7934,MILLER,CLERK,7782,23-JAN-82 00:00:00,1300.00,,10

```

5.3.1 複数の REFCURSOR を返す

次の例は、2 つの REFCURSORs を返します。

- 最初の REFCURSOR には、呼び出し元が指定した範囲内の部門で働くすべての従業員を含むカーソル (employees) の名前が含まれています。
- 2 番目の REFCURSOR には、呼び出し元が指定した範囲内のすべての部門を含むカーソル (departments) の名前が含まれています。

この例では、関数は単一の REFCURSOR を返す代わりに、SETOF REFCURSOR (これは、0 以上の REFCURSORs を意味します) を返します。もう 1 つの重要な違いは、libpq プログラムは結果セットに単一の REFCURSOR ではなく、それぞれに単一の値が含まれる 2 つの行を想定することです (最初の行には employees カーソルの名前が含まれ、2 番目の行には departments カーソルの名前)。

```
CREATE OR REPLACE FUNCTION getEmpsAndDepts(p_min NUMERIC,
                                           p_max NUMERIC)
RETURN SETOF REFCURSOR AS
    employees REFCURSOR;
    departments REFCURSOR;
BEGIN
    OPEN employees FOR
        SELECT * FROM emp WHERE deptno BETWEEN p_min AND p_max;
    RETURN NEXT employees;

    OPEN departments FOR
        SELECT * FROM dept WHERE deptno BETWEEN p_min AND p_max;
    RETURN NEXT departments;
END;
```

前の例のように、PQexec() および PQgetvalue() を使用して SPL ファンクションを呼び出すことができます。

```
# include <stdio.h>
# include <stdlib.h>
# include "libpq-fe.h"

static void fetchAllRows(PGconn *conn,
                        const char *cursorName,
                        const char *description);
static void fail(PGconn *conn, const char *msg);

int
main(int argc, char *argv[])
{
    PGconn *conn = PQconnectdb(argv[1]);
    PGresult *result;
```

(次のページに続く)

(前のページからの続き)

```

if (PQstatus(conn) != CONNECTION_OK)
    fail(conn, PQerrorMessage(conn));

result = PQexec(conn, "BEGIN TRANSACTION");

if (PQresultStatus(result) != PGRES_COMMAND_OK)
    fail(conn, PQerrorMessage(conn));

PQclear(result);

result = PQexec(conn, "SELECT * FROM getEmpsAndDepts(20, 30)");

if (PQresultStatus(result) != PGRES_TUPLES_OK)
    fail(conn, PQerrorMessage(conn));

fetchAllRows(conn, PQgetvalue(result, 0, 0), "employees");
fetchAllRows(conn, PQgetvalue(result, 1, 0), "departments");

PQclear(result);

PQexec(conn, "COMMIT");

PQfinish(conn);

exit(0);
}

static void
fetchAllRows(PGconn *conn,
             const char *cursorName,
             const char *description)
{
    size_t    commandLength = strlen("FETCH ALL FROM ") +
                             strlen(cursorName) + 3;
    char      *commandText    = malloc(commandLength);
    PGresult  *result;
    int       row;

    sprintf(commandText, "FETCH ALL FROM \"%s\"", cursorName);

    result = PQexec(conn, commandText);

    if (PQresultStatus(result) != PGRES_TUPLES_OK)
        fail(conn, PQerrorMessage(conn));

```

(次のページに続く)

(前のページからの続き)

```

printf("-- %s --\n", description);

for (row = 0; row < PQntuples(result); row++)
{
    const char *delimiter = "\t";
    int col;

    for (col = 0; col < PQnfields(result); col++)
    {
        printf("%s%s", delimiter, PQgetvalue(result, row, col));
        delimiter = ",";
    }

    printf("\n");
}

PQclear(result);
free(commandText);
}

static void
fail(PGconn *conn, const char *msg)
{
    fprintf(stderr, "%s\n", msg);

    if (conn != NULL)
        PQfinish(conn);

    exit(-1);
}

```

getEmpsAndDepts(20, 30) を呼び出すと、サーバーは部門 20 または 30 で働くすべての従業員を含むカーソルと、部門 20 および 30 の説明を含む 2 番目のカーソルを返します。

```

-- employees --
7369,SMITH,CLERK,7902,17-DEC-80 00:00:00,800.00,,20
7499,ALLEN,SALESMAN,7698,20-FEB-81 00:00:00,1600.00,300.00,30
7521,WARD,SALESMAN,7698,22-FEB-81 00:00:00,1250.00,500.00,30
7566,JONES,MANAGER,7839,02-APR-81 00:00:00,2975.00,,20
7654,MARTIN,SALESMAN,7698,28-SEP-81 00:00:00,1250.00,1400.00,30
7698,BLAKE,MANAGER,7839,01-MAY-81 00:00:00,2850.00,,30
7788,SCOTT,ANALYST,7566,19-APR-87 00:00:00,3000.00,,20
7844,TURNER,SALESMAN,7698,08-SEP-81 00:00:00,1500.00,0.00,30
7876,ADAMS,CLERK,7788,23-MAY-87 00:00:00,1100.00,,20
7900,JAMES,CLERK,7698,03-DEC-81 00:00:00,950.00,,30

```

(次のページに続く)

```

7902,FORD,ANALYST,7566,03-DEC-81 00:00:00,3000.00,,20
- - departments --
20,RESEARCH,DALLAS
30,SALES,CHICAGO

```

5.4 配列バインディング

EDB Postgres Advanced Server の配列バインド機能を使用すると、1 回の往復でネットワーク全体にデータの配列を送信できます。バックエンドはバルクデータを受信すると、そのデータを使用して挿入または更新操作を実行できます。

準備されたステートメントで一括操作を実行します。次の関数を使用してステートメントを準備します。

```

PGresult *PQprepare(PGconn *conn,
                    const char *stmtName,
                    const char *query,
                    int nParams,
                    const Oid *paramTypes);

```

PQprepare() の詳細は、準備されたステートメントのセクションにあります。

次の関数を使用して、一括操作を実行できます。

- PQBulkStart
- PQexecBulk
- PQBulkFinish
- PQexecBulkPrepared

5.4.1 PQBulkStart

PQBulkStart() は、サーバーでの一括操作を初期化します。サーバーにバルクデータを送信する前に、この関数を呼び出す必要があります。PQBulkStart() は、stmtName で指定されたプリペアドステートメントを初期化して、paramFmts で指定された形式でデータを受け取ります。

5.4.2 API 定義

```
PGresult *PQBulkStart(PGconn *conn,
                      const char *Stmt_Name,
                      unsigned int nCol,
                      const int *paramFmts);
```

5.4.3 PQexecBulk

PQexecBulk() は、以前に PQBulkStart() を使用してバルク操作用に初期化されたステートメントにデータ (paramValues) を提供するために使用されます。

この関数は、PQBulkStart() の後に複数回使用して、複数のデータブロックを送信できます。詳細については、例を参照してください。

5.4.4 API 定義

```
PGresult *PQexecBulk(PGconn *conn,
                     unsigned int nRows,
                     const char *const * paramValues,
                     const int *paramLengths);
```

5.4.5 PQBulkFinish

この関数は、現在の一括操作を完了します。準備されたステートメントは、再準備せずに再度使用できます。

5.4.6 API 定義

```
PGresult *PQBulkFinish(PGconn *conn);
```

5.4.7 PQexecBulk 準備済み

または、PQexecBulkPrepared() ファンクションを使用して、1 回のファンクションコールで一括操作を実行できます。PQexecBulkPrepared() は、指定されたパラメーターを使用して準備されたステートメントを実行する要求を送信し、結果を待ちます。この関数は、PQbulkStart()、PQexecBulk()、および PQBulkFinish() の機能を組み合わせたものです。この関数を使用する場合、一括操作を初期化または終了する必要はありません。このファンクションは、一括操作を開始し、データをサーバーに渡して、一括操作を終了します。

stmtName の場所には、事前に準備されたステートメントを指定します。繰り返し使用されるコマンドは、実行されるたびにではなく、1 回だけ解析および計画されます。

5.4.8 API 定義

```
PGresult *PQexecBulkPrepared(PGconn *conn,
                             const char *stmtName,
                             unsigned int nCols,
                             unsigned int nRows,
                             const char *const *paramValues,
                             const int *paramLengths,
                             const int *paramFormats);
```

5.4.9 コード例 (PQBulkStart、PQexecBulk、PQBulkFinish を使用)

次の例では、PQBulkStart、PQexecBulk、および PQBulkFinish を使用しています。

```
void InsertDataUsingBulkStyle( PGconn *conn )
{
    PGresult      *res;
    Oid            paramTypes[2];
    char          *paramVals[5][2];
    int            paramLens[5][2];
    int            paramFmts[2];
    int            i;

    int            a[5] = { 10, 20, 30, 40, 50 };
    char           b[5][10] = { "Test_1", "Test_2", "Test_3",
    "Test_4", "Test_5" };

    paramTypes[0] = 23;
    paramTypes[1] = 1043;
    res = PQprepare( conn, "stmt_1", "INSERT INTO testtable1 values( $1, $2
)", 2, paramTypes );
    PQclear( res );

    paramFmts[0] = 1;    /* Binary format */
    paramFmts[1] = 0;

    for( i = 0; i < 5; i++ )
    {
        a[i] = htonl( a[i] );
        paramVals[i][0] = &(a[i]);
        paramVals[i][1] = b[i];

        paramLens[i][0] = 4;
        paramLens[i][1] = strlen( b[i] );
    }
}
```

(次のページに続く)

(前のページからの続き)

```

}

res = PQBulkStart(conn, "stmt_1", 2, paramFmts);
PQclear( res );
printf( "< -- PQBulkStart -- >\n" );

res = PQexecBulk(conn, 5, (const char *const *)paramVals, (const
int*)paramLens);
PQclear( res );
printf( "< -- PQexecBulk -- >\n" );

res = PQexecBulk(conn, 5, (const char *const *)paramVals, (const
int*)paramLens);
PQclear( res );
printf( "< -- PQexecBulk -- >\n" );

res = PQBulkFinish(conn);
PQclear( res );
printf( "< -- PQBulkFinish -- >\n" );
}

```

5.4.10 コード例 (PQexecBulkPrepared を使用)

次の例では、PQexecBulkPrepared を使用しています。

```

void InsertDataUsingBulkStyleCombinedVersion( PGconn *conn )
{
    PGresult      *res;
    Oid            paramTypes[2];
    char          *paramVals[5][2];
    int            paramLens[5][2];
    int            paramFmts[2];
    int            i;

    int            a[5] = { 10, 20, 30, 40, 50 };
    char           b[5][10] = { "Test_1", "Test_2", "Test_3",
"Test_4", "Test_5" };

    paramTypes[0] = 23;
    paramTypes[1] = 1043;
    res = PQprepare( conn, "stmt_2", "INSERT INTO testtable1 values(
$1, $2)", 2, paramTypes );
    PQclear( res );

```

(次のページに続く)

```
paramFmts[0] = 1;  /* Binary format */
paramFmts[1] = 0;

for( i = 0; i < 5; i++ )
{
    a[i] = htonl( a[i] );
    paramVals[i][0] = &(a[i]);
    paramVals[i][1] = b[i];

    paramLens[i][0] = 4;
    paramLens[i][1] = strlen( b[i] );
}

res = PQexecBulkPrepared(conn, "stmt_2", 2, 5, (const char *const
*)paramVals, (const int *)paramLens, (const int *)paramFmts);
PQclear( res );
}
```

第6章 Debugger

デバッガーを使用すると、開発者と DBA は、グラフィカルな動的環境を使用してサーバーサイドプログラムをテストおよびデバッグできます。デバッグできるプログラムの種類は、SPL のストアドプロシージャ、ファンクション、トリガー、パッケージのほか、PL/pgSQL のファンクションとトリガーです。

デバッガーは *pgAdmin 4* および* EDB Postgres Enterprise Manager*と統合されています。Windows ホストに EDB Postgres Advanced Server をインストールしている場合、pgAdmin 4 が自動的にインストールされます。pgAdmin 4 アイコンは Windows Start メニューにあります。EDB Postgres Advanced Server ホストが CentOS または Linux システム上にある場合、yum を使用して pgAdmin4 をインストールできます。コマンドラインを開き、スーパーユーザ権限を引き受けて、次のように入力します。

```
yum install edb-pgadmin4*
```

Linux では、xx が EDB Postgres Advanced Server のバージョン番号である *edb-asxx-server-pldebugger* RPM パッケージもインストールする必要があります。pgAdmin 4 に関する情報は次の場所で入手できます。

<https://www.pgadmin.org/>

RPM インストールにより、[アプリケーション] メニューに pgAdmin4 アイコンが追加されます。

デバッガーを使用してプログラムをテストするには、2 つの基本的な方法があります。

- **スタンドアロンデバッグ。**デバッガーは、テストするプログラムを起動するために使用されます。プログラムに必要な入力パラメーター値を指定すると、すぐにプログラムのコードを観察してステップ実行できます。スタンドアロンデバッグは、プログラムの新規作成や初期の問題調査に使用される一般的な方法です。
- **インコンテキストデバッグ。**テスト対象のプログラムは、デバッガー以外のアプリケーションによって開始されます。まず、テストするプログラムに**グローバルブレークポイント**を設定します。プログラムを最初に呼び出すアプリケーションは、グローバルブレークポイントに遭遇します。アプリケーションは実行を中断し、デバッガーが呼び出されたプログラムの制御を取得します。次に、呼び出されたプログラムが呼び出し元のアプリケーションのコンテキスト内で実行されるときに、呼び出されたプログラムのコードを観察し、ステップ実行できます。デバッガーで呼び出されたプログラムのコードを完全にステップ実行すると、中断されたアプリケーションは実行を再開します。インコンテキストデバッグは、呼び出し元のアプリケーションとの複雑な相互作用が原因でスタンドアロンデバッグを使用して問題を再現することが難しい場合に役立ちます。

デバッグツールと操作は、スタンドアロンとインコンテキストのどちらを使用しても同じです。違いは、デバッグするプログラムの呼び出し方法です。

以下のセクションでは、スタンドアロンのデバッグ方法を使用したデバッガーの特徴と機能について説明します。インコンテキストデバッグ用にデバッガーを起動する方法については、

Setting Global Breakpoint for In-Context Debugging で説明されています。

6.1 デバッガーの構成

デバッガーを使用する前に、`postgresql.conf` ファイル（EDB Postgres Advanced Server ホームディレクトリの `data` サブディレクトリにあります）を編集し、`shared_preload_libraries` 構成パラメーターにリストされているライブラリに `$libdir/plugin_debugger` を追加します。

```
shared_preload_libraries = $libdir/dbms_pipe,$libdir/edb_gen,$libdir/plugin_debugger
```

- Linux では、`postgresql.conf` ファイルは次の場所にあります：`/var/lib/edb/asxx/data`
- Windows では、`postgresql.conf` ファイルは次の場所にあります：`C:\Program Files\edb\asxx\data`

xx は EDB Postgres Advanced Server のバージョンです。

`shared_preload_libraries` パラメータを変更した後、データベースサーバを再起動する必要があります。

6.2 デバッガーの起動

pgAdmin 4 を使用して、スタンドアロンデバッグ用のデバッガーにアクセスします。デバッガーを開くには、pgAdmin 4 Browser パネルでデバッグするストアドプロシージャまたはファンクションの名前を強調表示します。次に、Object メニューから Debugging メニューに移動し、サブメニューから Debug を選択します。

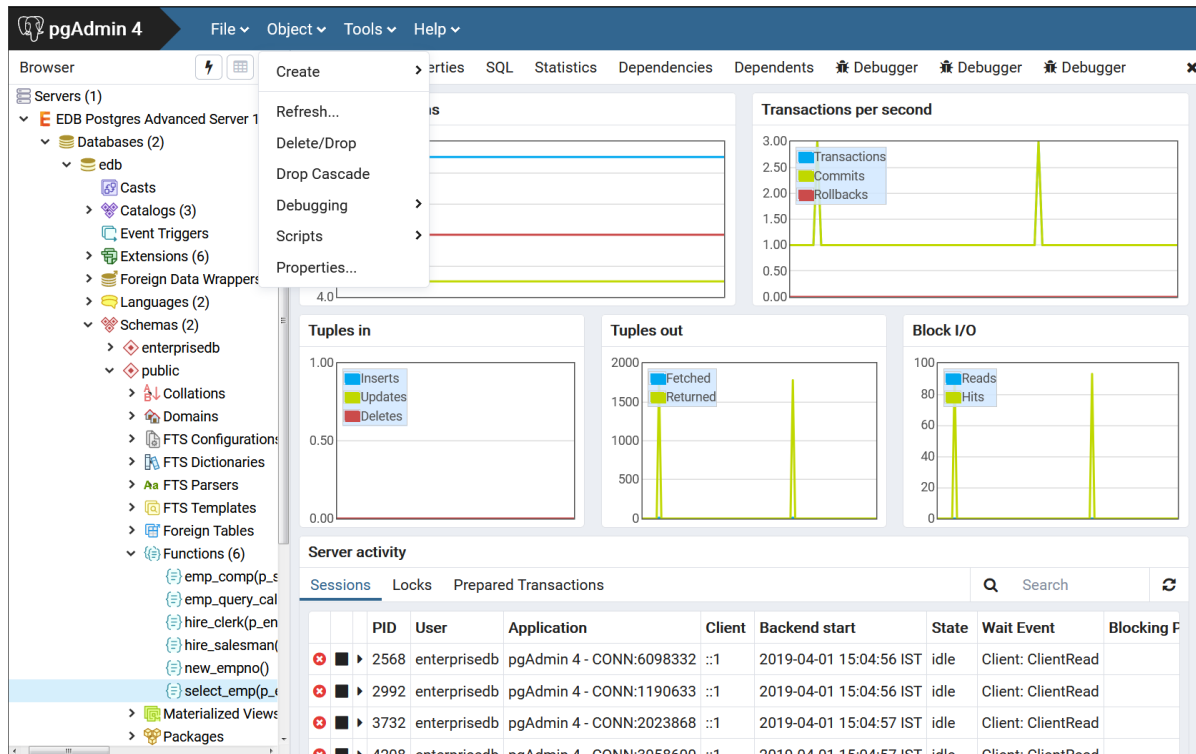


図 1: Starting the Debugger from the Object menu

図 1: [オブジェクト] メニューからのデバッガーの起動

pgAdmin 4 Browser でストアードプロシージャまたはファンクションの名前を右クリックし、コンテキストメニューから Debugging、および Debug を選択することもできます。

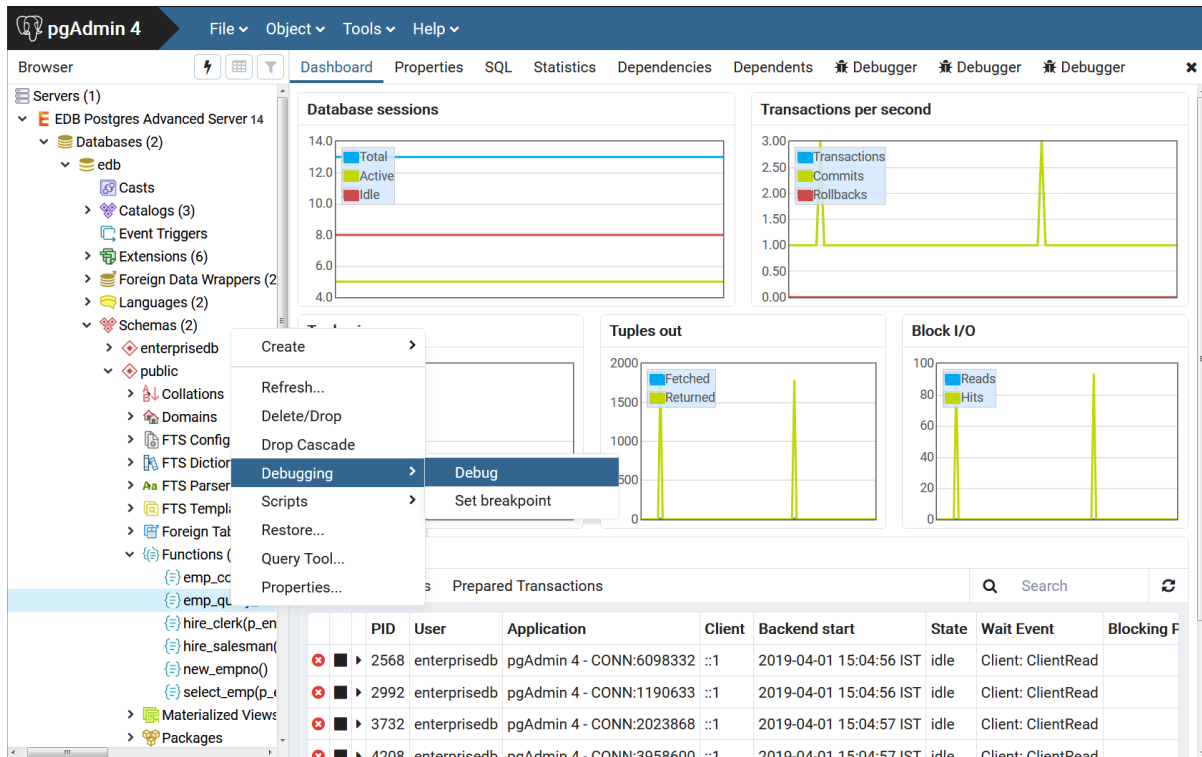


図 2: Starting the Debugger from the object's context menu

図 2: オブジェクトのコンテキストメニューからのデバッガーの起動

スタンドアロンデバッグを使用してトリガーをデバッグすることはできないことに注意してください。トリガーは、インコンテキストデバッグを使用してデバッグする必要があります。インコンテキストデバッグのグローバルブレイクポイントの設定については、

Setting Global Breakpoint for In-Context Debugging を参照してください。

パッケージをデバッグするには、デバッグするパッケージのパッケージノードの下にある特定のプロシージャまたはファンクションを強調表示し、ストアードプロシージャとファンクションと同じ指示に従います。

6.3 デバッガーウィンドウ

パラメーターを必要とするプログラムをスタンドアロンでデバッグしているときに、Debugger ウィンドウを使用してパラメーター値を渡すことができます。デバッガーを起動すると、Debugger ウィンドウが自動的に開き、プログラムが必要とする IN または IN OUT パラメーターが表示されます。プログラムが IN または IN OUT パラメーターを宣言していない場合、Debugger ウィンドウは開きません。

図 3: デバッガーウィンドウ

Debugger ウィンドウのフィールドを使用して、各パラメーターの値を指定します。

- Name フィールドには、仮パラメーター名が含まれます。
- Type フィールドには、パラメーターのデータ型が含まれます。

Debugger						
Name	Type	Null?	Expression?	Value	Use Default?	Default value
p_sal	numeric	☒	☒	NaN	☐	<No default value>
p_comm	numeric	☒	☒	NaN	☐	<No default value>

✕ Cancel
🐞 Debug

図 3: The Debugger window

- **Null?** チェックボックスをオンにして、パラメーターが NULL 値であることを示します。
- **Value** フィールドに式が含まれる場合は、**Expression?** チェックボックスをオンにします。
- **Value** フィールドには、プログラムに渡されるパラメーター値が含まれています。
- **Use Default?** チェックボックスをオンにして、プログラムが **Default Value** フィールドの値を使用する必要があることを示します。
- **Default Value** フィールドには、パラメーターのデフォルト値が含まれます。

Tab キーを押して、データ入力用のリスト内の次のパラメーターを選択するか、**Value** フィールドをクリックしてデータ入力用のパラメーターを選択します。

初期化セクションを持つパッケージのメンバーであるプロシージャまたはファンクションをデバッグする場合は、**Debug Package Initializer** チェックボックスをオンにして、パッケージの初期化セクションにステップインするように指示し、プロシージャをデバッグする前に初期化セクションのコードをデバッグできますまたは機能。このチェックボックスを選択しない場合、デバッガーはパッケージの初期化セクションを実行します。実行時にコードの個々の行を表示したりステップ実行したりすることはできません。

必要なパラメーター値を入力したら、**Debug** ボタンをクリックしてデバッグプロセスを開始します。**Cancel** ボタンをクリックして、デバッガーを終了します。

注釈: インコンテキストデバッグ中に `Debugger` ウィンドウが開きません。代わりに、デバッグするプログラムを呼び出すアプリケーションは、必要な入力パラメーター値を指定する必要があります。

プログラムコードをステップ実行してデバッグサイクル全体を完了すると、**Debugger** ウィンドウが再度開き、新しいパラメーター値を入力してデバッグサイクルを繰り返したり、デバッグセッションを終了したりできます。

6.4 メインデバッガーウィンドウ

メインデバッガーウィンドウには 2 つのパネルがあります。

- 上部の **Program Body** パネルには、プログラムのソースコードが表示されます。
- 下部の **Tabs** パネルには、さまざまな情報のタブセットがあります。

トップパネルにある **Tool Bar** アイコンを使用して、デバッグ機能にアクセスします。

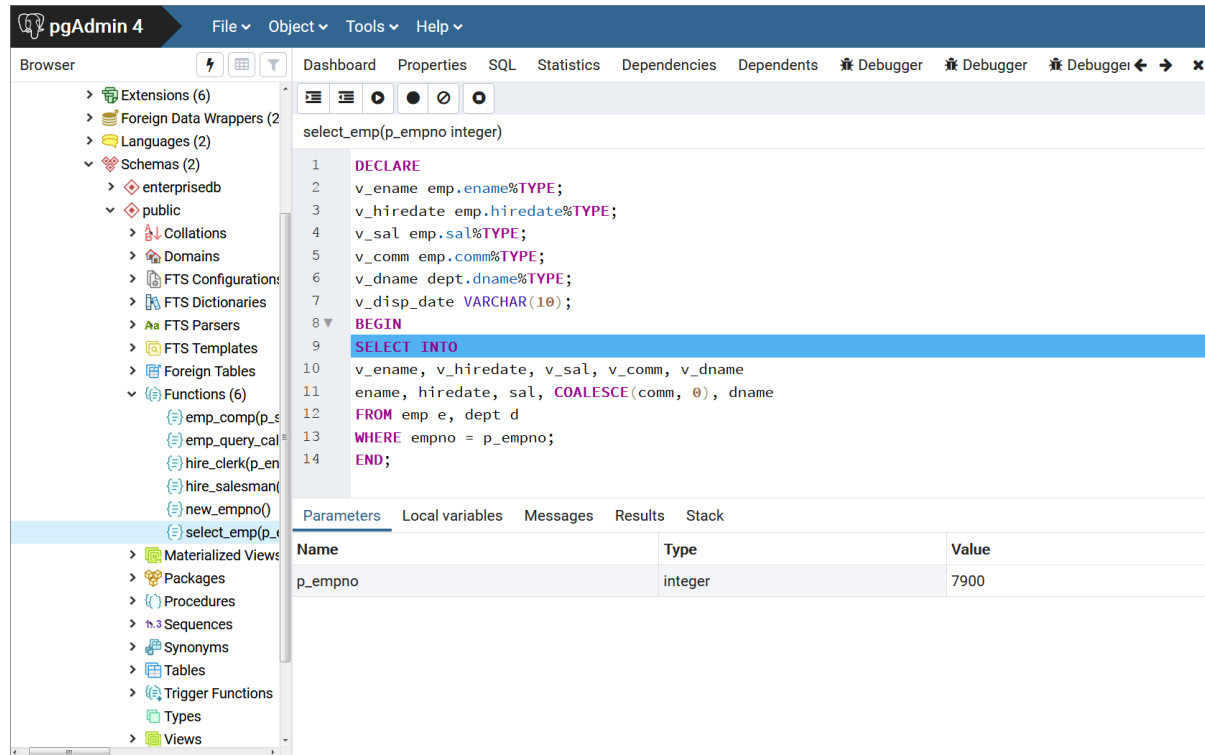


図 4: The Main Debugger window

図 4：メインデバッガーウィンドウ

2 つのパネルについては、次のセクションで説明します。

6.4.1 プログラム本体パネル

Program Body パネルには、デバッグ中のプログラムのソースコードが表示されます。

図 5：プログラム本体

この図は、デバッガーが **SELECT** ステートメントを実行しようとしていることを示しています。プログラム本体の青色のインジケーターは、実行する次のステートメントを強調表示します。

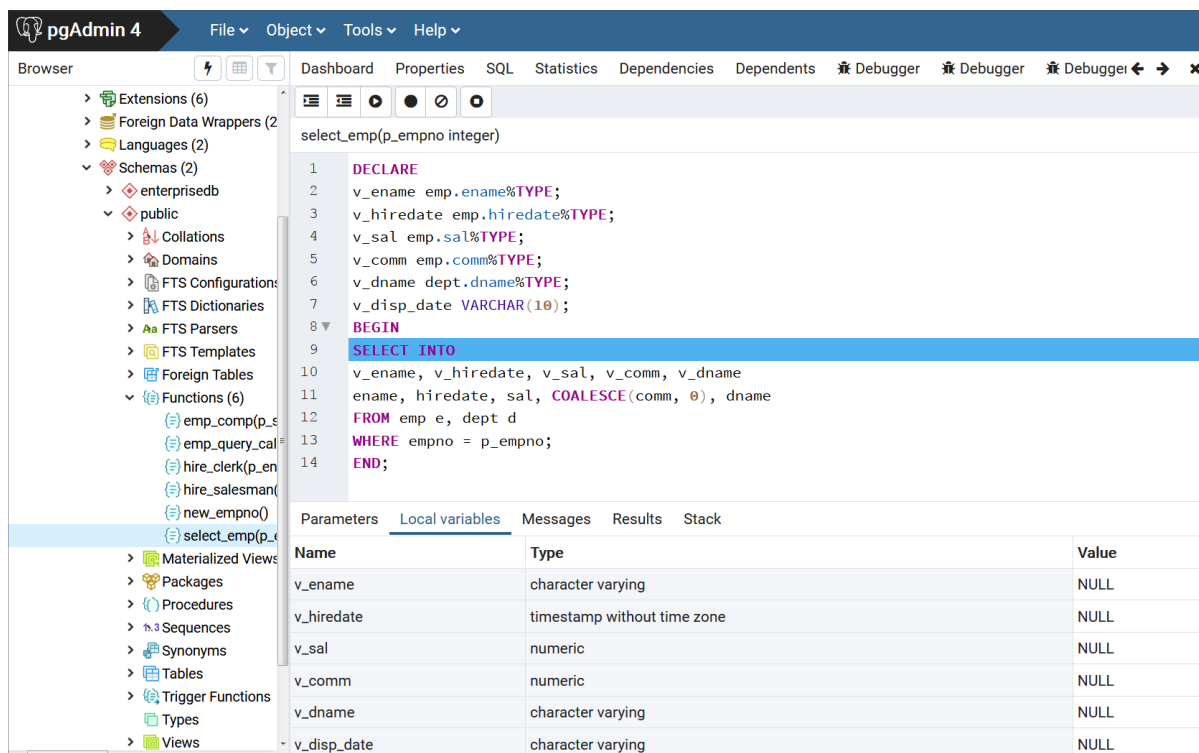


図 5: The Program Body

6.4.2 タブパネル

下部の Tabs パネルを使用して、パラメーター値またはローカル変数を表示または変更したり、RAISE INFO によって生成されたメッセージと関数の結果を表示したりできます。

パネルのタブに表示される情報は次のとおりです。

- **Parameters** タブには、現在のパラメーター値が表示されます。
- **Local variables** タブには、プログラム内で宣言された変数の値が表示されます。
- **Messages** タブには、プログラムの実行時に返された結果が表示されます。
- **Results** タブには、ファンクションの RETURN ステートメントの値など、プログラムの結果（該当する場合）が表示されます。
- **Stack** tab はコールスタックを表示します。

次の図は、さまざまなタブの結果を示しています。

図 6: [パラメーター] タブ

図 7: ローカル変数タブ

図 8: [メッセージ] タブ

図 9: [結果] タブ

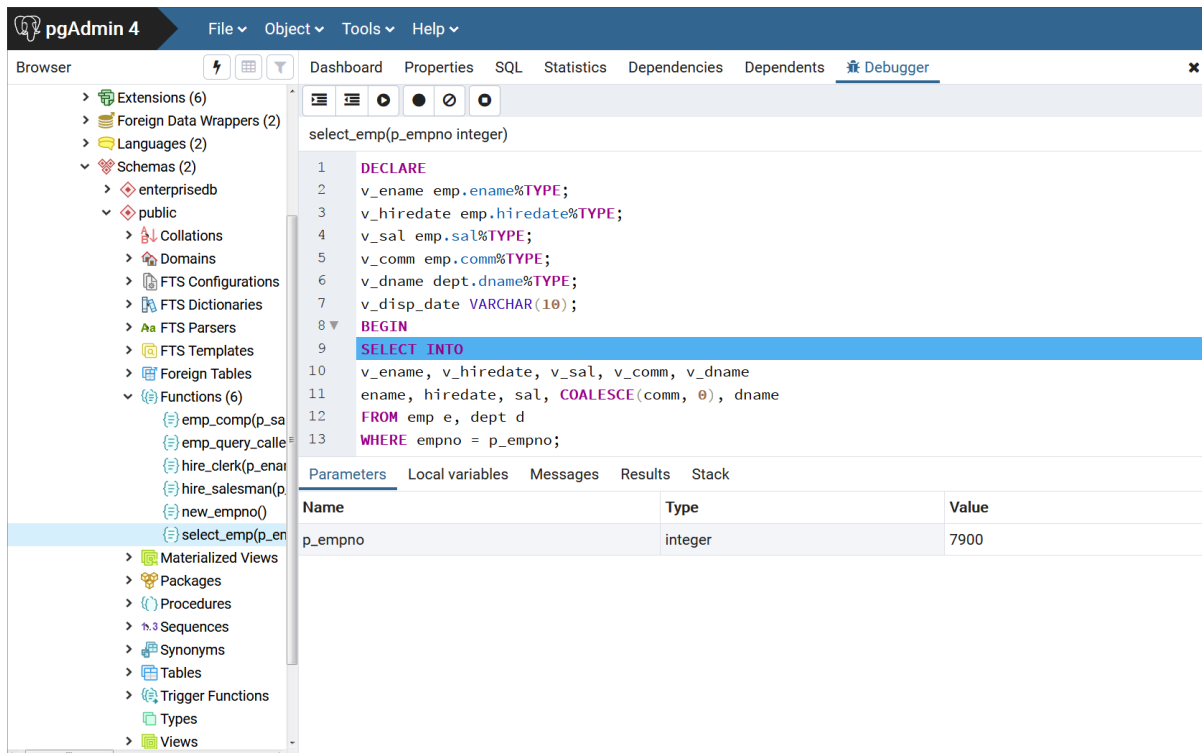


図 6: The Parameters tab

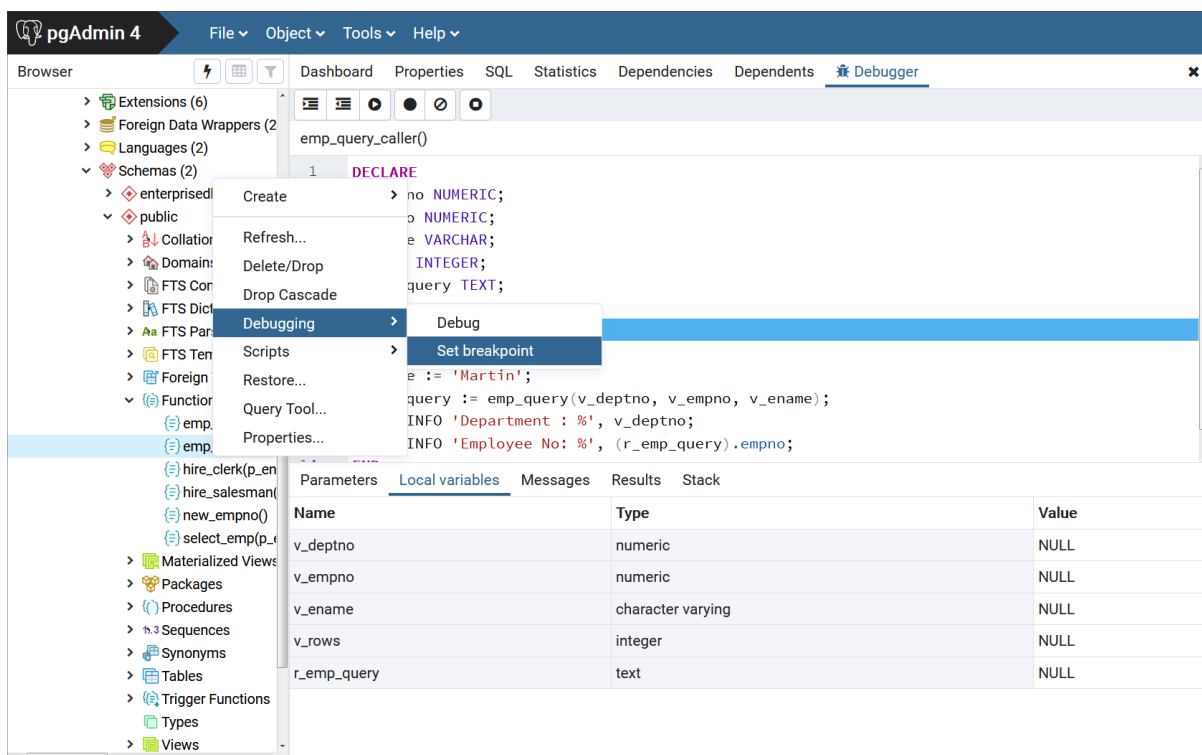


図 7: The Local variables tab

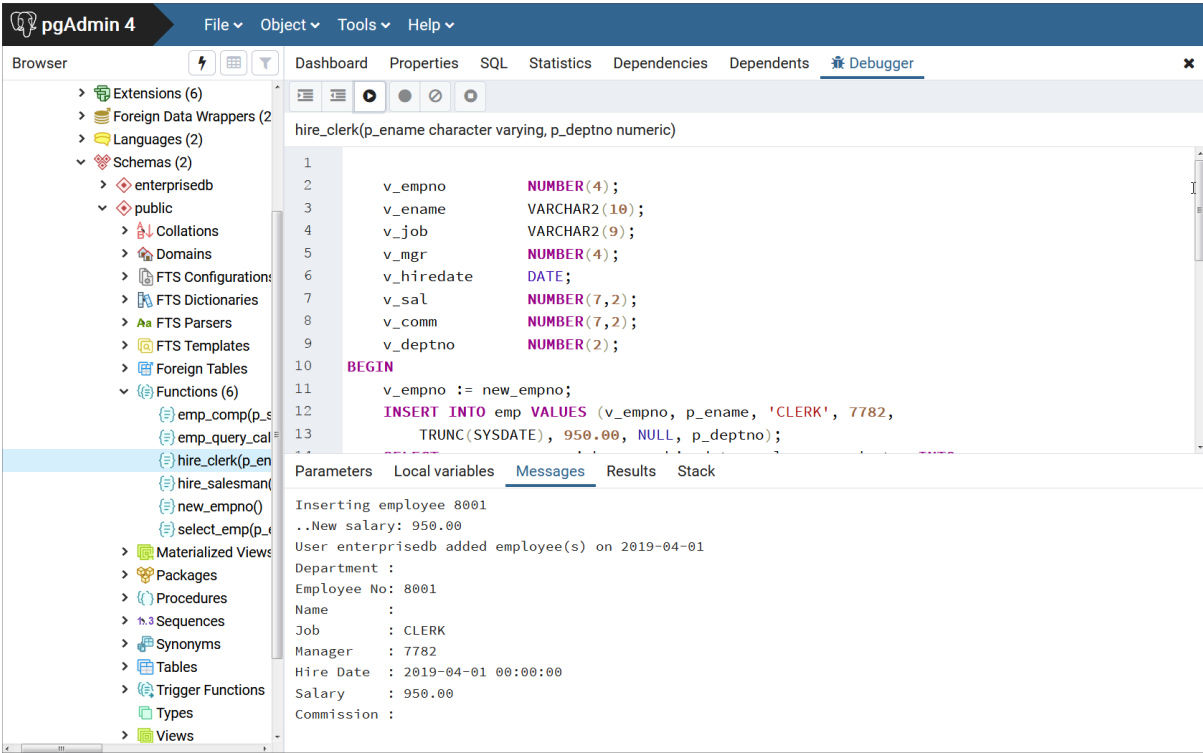


図 8: The Messages tab

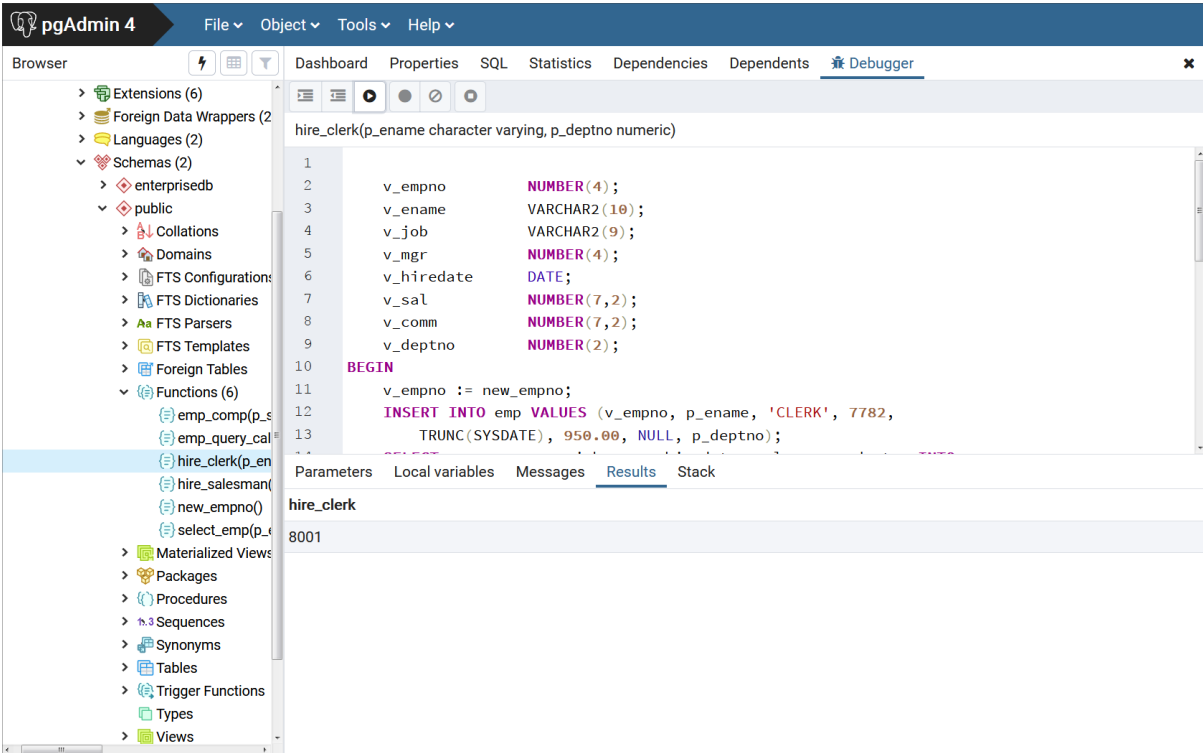


図 9: The Results tab

6.4.3 スタックタブ

Stack タブには、現在コールスタックにあるプログラム（呼び出されたがまだ完了していないプログラム）のリストが表示されます。プログラムを呼び出すと、[Stack] タブに表示されるリストの先頭にプログラムの名前が追加されます。プログラムが終了すると、その名前はリストから削除されます。

Stack タブには、プログラム呼び出しに関する情報も表示されます。情報には次のものが含まれます。

- プログラム内の呼び出しの場所
- 呼び出し引数
- 呼び出されているプログラムの名前

コールスタックを確認すると、一連のネストされたプログラムの実行過程をトレースできます。

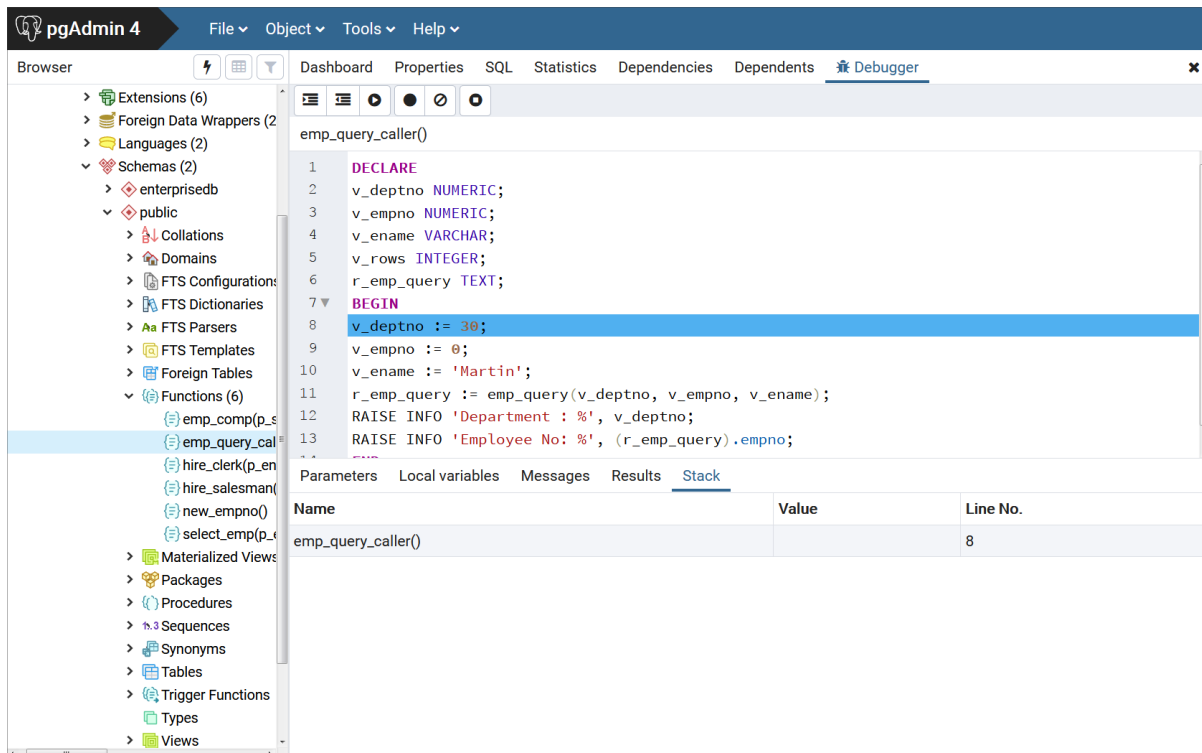


図 10: A debugged program calling a subprogram

図 10：サブプログラムを呼び出すデバッグ済みプログラム

上の図は、`emp_query_caller` が `emp_query` という名前のサブプログラムを呼び出そうとしていることを示しています。`emp_query_caller` は現在、コールスタックの先頭にあります。

`emp_query` への呼び出しが実行されると、`emp_query` が Stack タブの上部に表示され、そのコードが Program Body パネルに表示されます。

図 11：呼び出されたサブプログラムのデバッグ

サブプログラムの実行が完了すると、制御は呼び出し元プログラム (`emp_query_caller`) に戻り、現在は Stack タブの上部に表示されています。

図 12：デバッグされたサブプログラムから制御が戻る

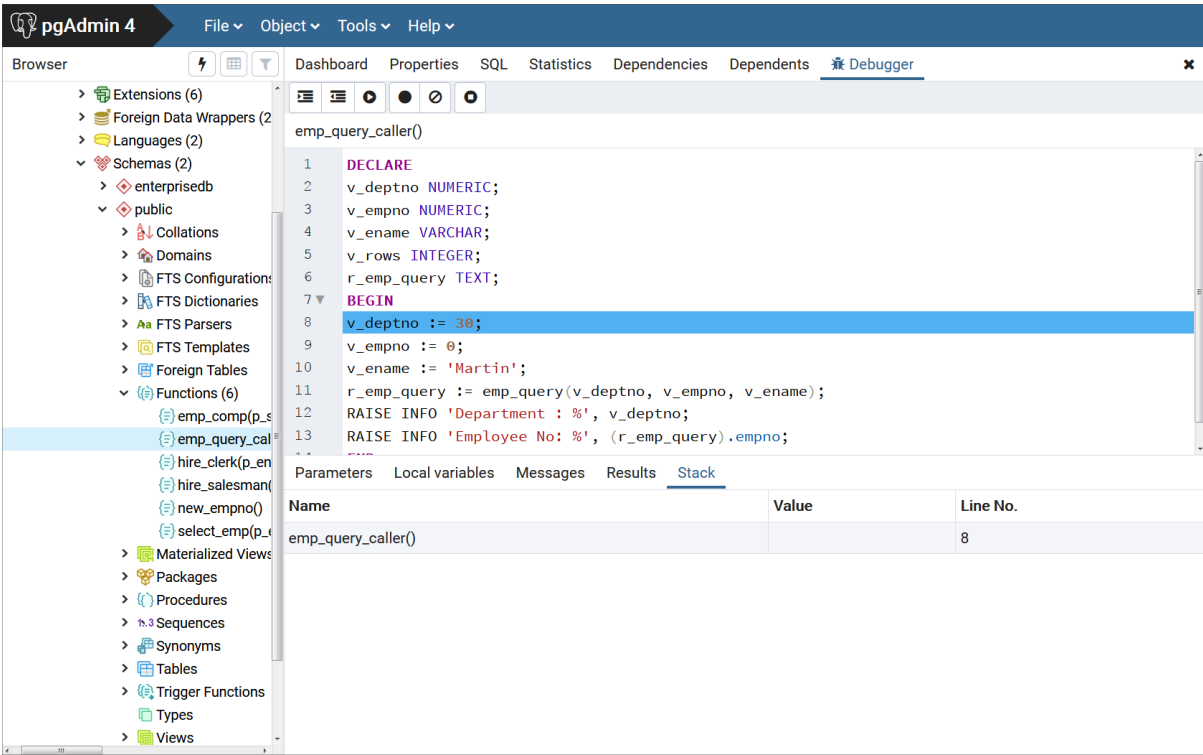


図 11: Debugging the called subprogram

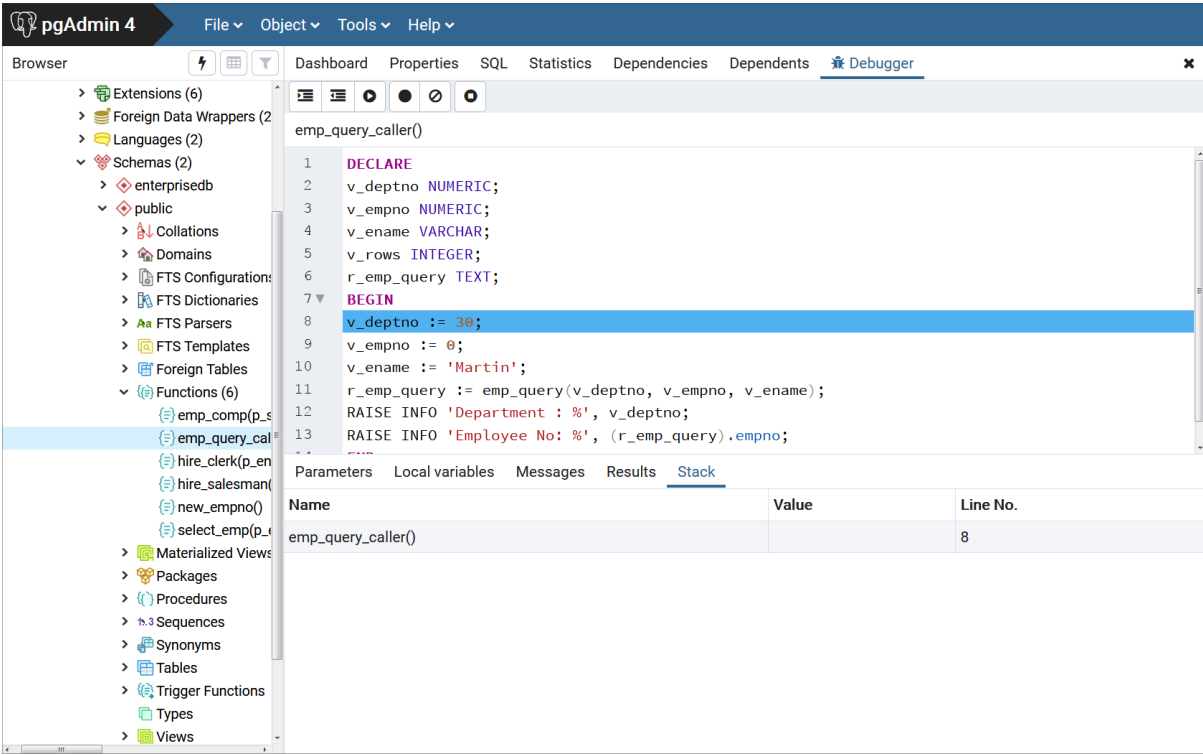


図 12: Control returns from debugged subprogram

6.5 プログラムのデバッグ

次の操作を実行して、プログラムをデバッグできます。

- プログラムを一度に 1 行ずつ実行します
- ブレークポイントに到達するまでプログラムを実行します
- プログラム内のローカル変数値を表示および変更する

6.5.1 コードのステップ実行

ツールバーのアイコンを使用して、デバッガーでプログラムをステップ実行します。



図 13: The Tool bar icons

図 13：ツールバーのアイコン

アイコンの目的は次のとおりです。

- **ステップイン。** Step into アイコンをクリックして、現在強調表示されているコード行を実行します。
- **ステップオーバー。** Step over アイコンをクリックしてコード行を実行し、コードによって呼び出されるサブファンクションをステップオーバーします。サブファンクションは実行されますが、ブレークポイントが含まれていない限りデバッグされません。
- **続行/開始。** Continue/Start アイコンをクリックして強調表示されたコードを実行し、プログラムがブレークポイントに到達するか完了するまで続行します。
- **停止。** Stop アイコンをクリックして、プログラムの実行を停止します。

6.5.2 ブレークポイントの使用

デバッガーはプログラムを実行すると、ブレークポイントに到達するたびに一時停止します。デバッガーが一時停止したら、ローカル変数を監視または変更したり、コールスタックのエントリに移動して、変数を監視したり、他のブレークポイントを設定したりできます。次のステップイン、ステップオーバー、または続行操作により、デバッガーはブレークポイントに続くコードの次の行から実行を再開します。ブレークポイントには次の 2 種類があります。

ローカル ブレークポイント - ローカルブレークポイントは、プログラム内の実行可能なコード行に設定できます。デバッガーは、ローカルブレークポイントが設定されている行に到達すると実行を一時停止します。

グローバルブレークポイント - 任意のセッションがそのブレークポイントに到達すると、グローバルブレークポイントがトリガーされます。プログラムのコンテキストデバッグを実行する場合は、グローバルブレークポイントを設定します。プログラムにグローバルブレークポイントが設定されている場合、グローバルブレークポイントを設定したデバッグセッションは、そのプログラムが別のセッションで呼び出されるまで待機します。グローバルブレークポイントはスーパーユーザーのみが設定できます。

ローカルブレークポイントを作成するには、ローカルブレークポイントを設定するコード行の左側の灰色の網掛けの余白内を左クリックします。グレーの網掛けの余白をクリックする場所は、ソースコードの 12 行目にブレークポイントドットが表示される場所のように、余白の右側にある必要があります。

デバッガーは作成されると、余白に黒い点を表示し、選択したコード行にブレークポイントが設定されたことを示します。

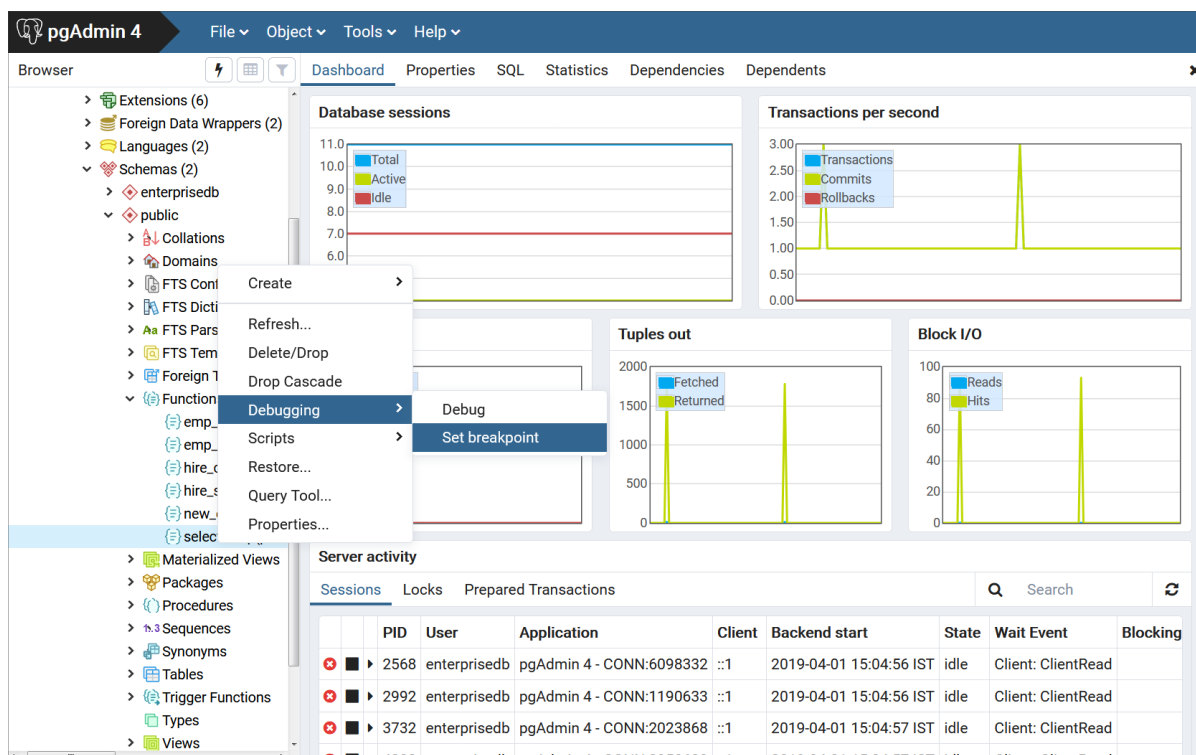


図 14: Set a breakpoint by clicking in left-hand margin

図 14: 左側のマージンをクリックしてブレークポイントを設定する

ローカルブレークポイントは必要な数だけ設定できます。ローカルブレークポイントは、デバッグセッションの間、削除されるまで有効です。

6.5.3 ローカルブレイクポイントの削除

ローカルブレイクポイントを削除するには、**Program Body** パネルの灰色の網掛けの余白にあるブレイクポイントのドットを左クリックします。ドットが消え、ブレイクポイントが削除されたことを示します。

Clear all breakpoints アイコンをクリックすると、現在 **Program Body** フレームに表示されているプログラムからすべてのブレイクポイントを削除できます。

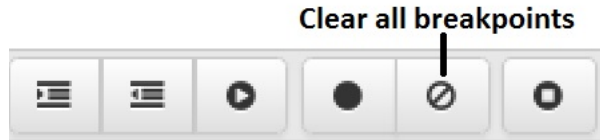


図 15: Clear all breakpoints icon

図 15：すべてのブレイクポイントをクリアするアイコン

注釈： 上記のアクションのいずれかを実行すると、現在 *Program Body* パネルに表示されているプログラム内のブレイクポイントのみが削除されます。呼び出されたサブプログラム内のブレイクポイントまたは現在 *Program Body* パネルに表示されているプログラムを呼び出すプログラム内のブレイクポイントは削除されません。

6.5.4 インコンテキストデバッグのグローバルブレイクポイントを設定する

インコンテキストデバッグ用にグローバルブレイクポイントを設定するには、**Browser** パネルでブレイクポイントを設定するストアードプロシージャ、ファンクション、またはトリガーを強調表示します。**Object** メニューをナビゲートして、**Debugging**、次に **Set Breakpoint** を選択します。

図 16：[オブジェクト] メニューからのグローバルブレイクポイントの設定

または、グローバルブレイクポイントを設定するストアードプロシージャ、ファンクション、またはトリガーの名前を右クリックし、次のようにコンテキストメニューから **Set Breakpoint** を選択します。

図 17：オブジェクトのコンテキストメニューからのグローバルブレイクポイントの設定

トリガーにグローバルブレイクポイントを設定するには、トリガーを含むテーブルノードを展開し、デバッグする特定のトリガーを強調表示し、ストアードプロシージャとファンクションの場合と同じ指示に従います。

パッケージにグローバルブレイクポイントを設定するには、デバッグするパッケージのパッケージノードの下の特定のプロシージャまたはファンクションを強調表示し、ストアードプロシージャおよびファンクションの場合と同じ指示に従います。

Set Breakpoint を選択すると、デバッガーウィンドウが開き、アプリケーションがデバッグするプログラムを呼び出すのを待ちます。

図 18：プログラムの呼び出しがデバッグされるのを待っています

PSQL クライアントが `select_emp` 関数（グローバルブレイクポイントが設定されています）を呼び出します。

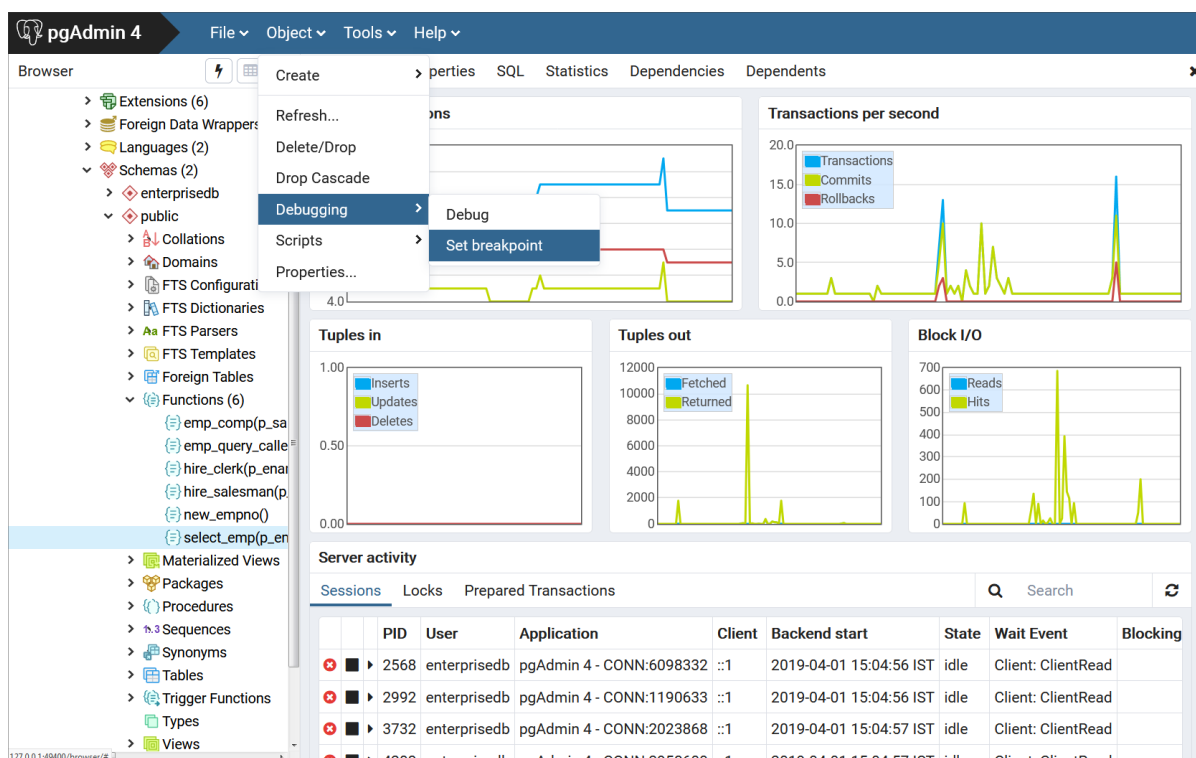


図 16: Setting a global breakpoint from the Object menu

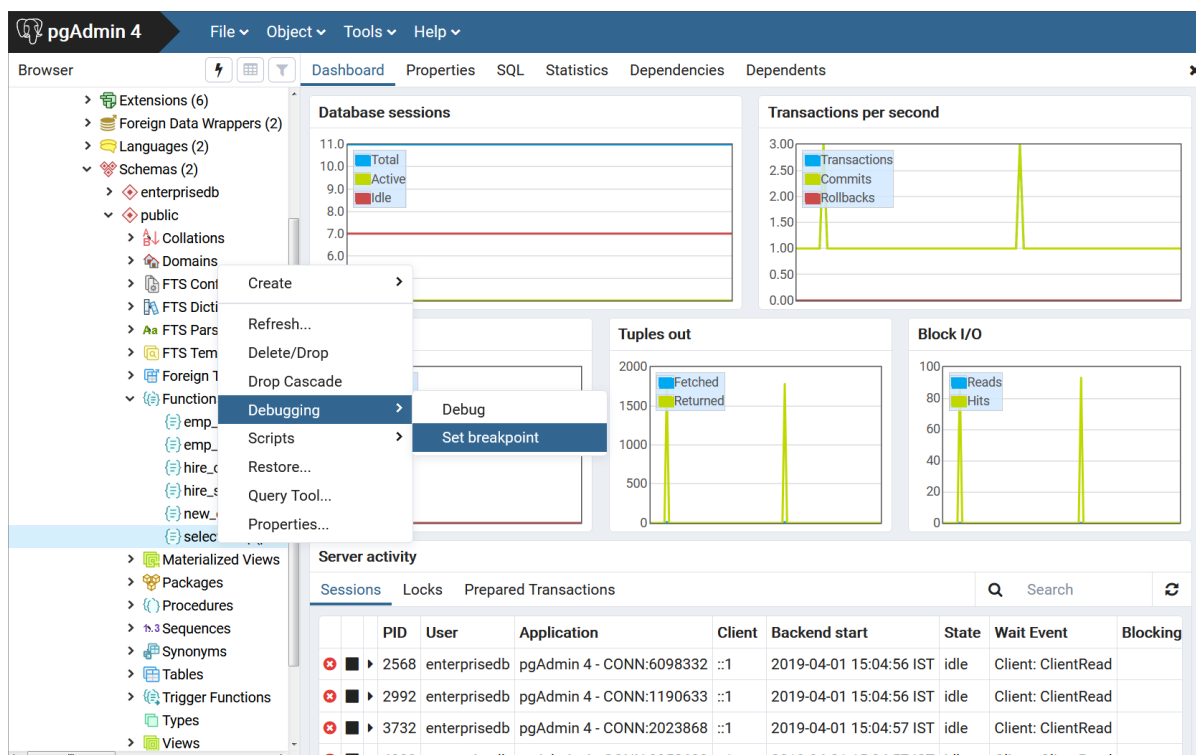


図 17: Setting a global breakpoint from the object's context menu

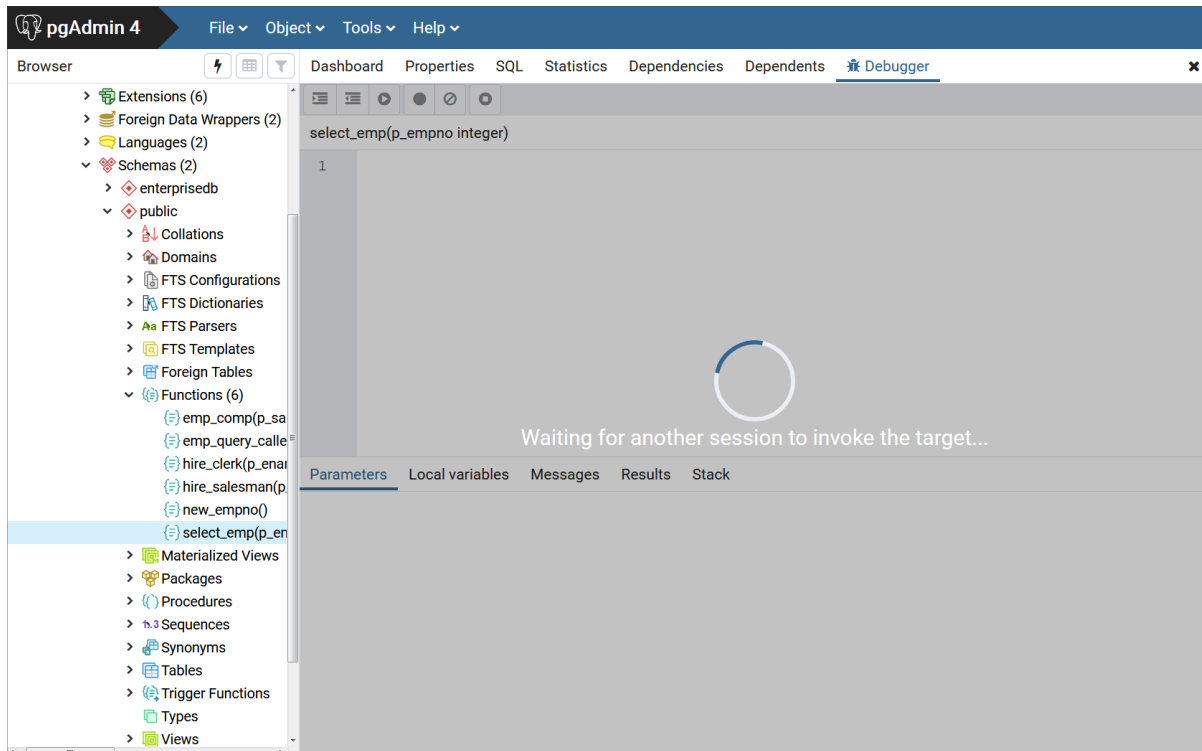


図 18: Waiting for invocation of program to be debugged

```
$ psql edb enterprisedb
psql.bin (14.0.0, server 14.0.0)
Type "help" for help.

edb=# SELECT select_emp(7900);
```

`select_emp` ファンクションは、デバッガーでプログラムをステップ実行するまで完了しません。

図 19: グローバルブレークポイントが設定されたプログラム

ステップイン、ステップオーバー、続行、またはローカルブレークポイントの設定など、前述の操作を使用してプログラムをデバッグできます。プログラムの実行をステップスルーすると、呼び出し元のアプリケーション (PSQL) が制御を取り戻し、`select_emp` ファンクションが実行を完了し、その出力が表示されます。

```
$ psql edb enterprisedb
psql.bin (14.0.0, server 14.0.0)
Type "help" for help.

edb=# SELECT select_emp(7900);
__OUTPUT__
INFO: Number      : 7900
INFO: Name        : JAMES
INFO: Hire Date   : 12/03/1981
INFO: Salary      : 950.00
```

(次のページに続く)

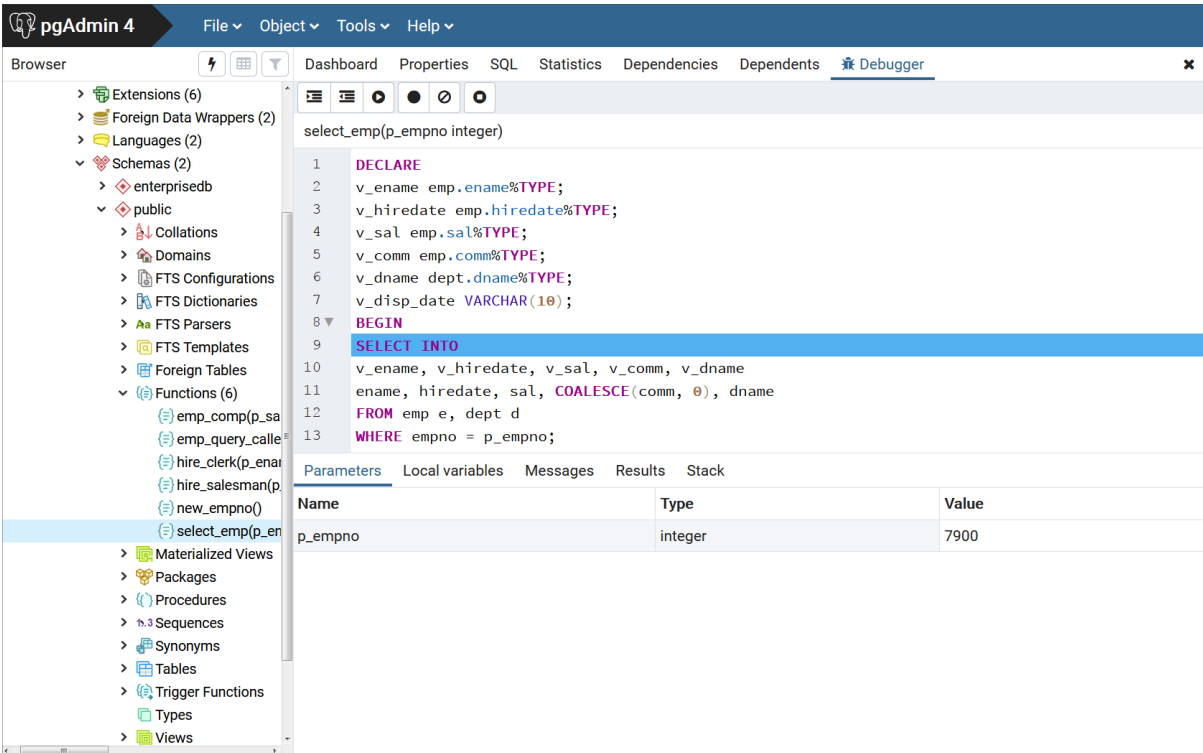


図 19: Program on which a global breakpoint has been set

(前のページからの続き)

```
INFO: Commission: 0.00
INFO: Department: SALES
select_emp
-----
(1 row)
```

この時点で、デバッガーセッションを終了できます。デバッガーセッションを終了しない場合、プログラムを呼び出す次のアプリケーションがグローバルブレークポイントに遭遇し、デバッグサイクルが再び開始されます。

6.5.5 デバッガーの終了

デバッガーセッションを終了してデバッガーを終了するには、右上隅にある閉じるアイコン (x) をクリックしてタブを閉じます。

図 20: デバッガーの終了

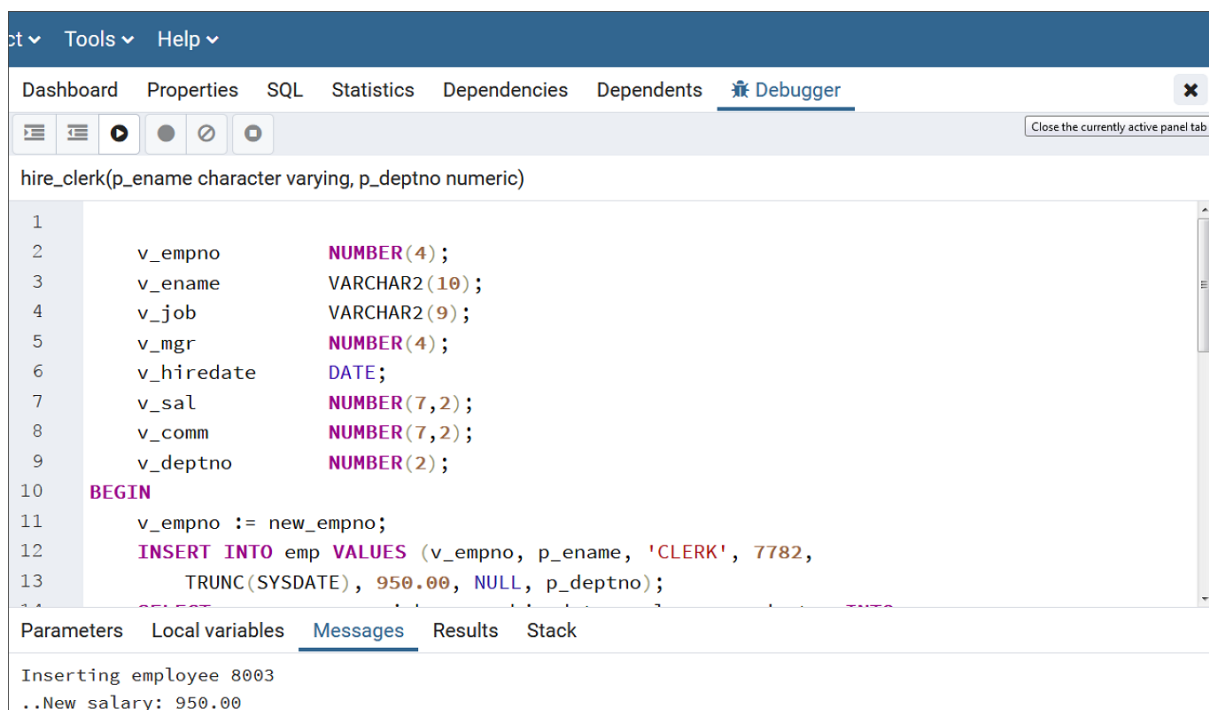


図 20: Exiting from the Debugger

第7章 Performance analysis and tuning

EDB Postgres Advanced Server は、パフォーマンス分析とチューニングのためのさまざまなツールを提供します。このセクションでは、これらの機能について説明します。

7.1 ダイナチューン

EDB Postgres Advanced Server は、データベースサーバーの動的チューニングをサポートして、インストールされているホストマシンで使用可能なシステムリソースを最適化します。この機能を制御する2つのパラメーターは、`postgresql.conf` ファイルにあります。これらのパラメーターは次のとおりです。

- `edb_dynatune`
- `edb_dynatune_profile`

7.1.1 `edb_dynatune`

`edb_dynatune` は、ホストマシンで使用可能なリソースの合計とホストマシンの使用目的に基づいて、データベースサーバーが使用するホストシステムのリソースの量を決定します。

EDB Postgres Advanced Server を最初にインストールするとき、`edb_dynatune` パラメーターは、インストールされたホストマシンの選択した用途（開発マシン、混合使用マシン、または専用サーバー）に従って設定されます。ほとんどの場合、データベース管理者がパフォーマンスを向上させるために `postgresql.conf` ファイルのさまざまな構成パラメーターを調整する必要はありません。

`postgresql.conf` ファイルを編集することにより、EDB Postgres Advanced Server の初期インストール後に `edb_dynatune` パラメーターの値を変更できます。新しい設定を有効にするには、`postmaster` を再起動する必要があります。

`edb_dynatune` パラメーターは、0 以上 100 以下の整数値に設定できます。0 の値は動的チューニング機能をオフにするため、データベースサーバーリソースの使用量は `postgresql.conf` ファイルの他の構成パラメーターによって完全に制御されます。

ゼロ以外の低い値（例、1 から 33）は、ホストマシンのリソースの最小量をデータベースサーバーに割り当てます。この設定は、他の多くのアプリケーションが使用されている開発マシンに使用されます。

34 から 66 の範囲の値は、適度な量のリソースをデータベースサーバーに割り当てます。この設定は、EDB Postgres Advanced Server と同じマシンで固定数の他のアプリケーションが実行されている専用のアプリケーションサーバーに使用される場合があります。

最高の値（例、67 - 100）は、サーバーのリソースの大部分をデータベースサーバーに割り当てます。この設定は、EDB Postgres Advanced Server の実行専用のホストマシンに使用されます。

`edb_dynatune` の値を選択したら、`postgresql.conf` ファイルの他の構成パラメーターを調整して、データベースサーバーのパフォーマンスをさらに調整できます。調整された設定は、`edb_dynatune` によって選択された対応する値をオーバーライドします。構成パラメーターのコメントを外し、目的の値を指定して、データベースサーバを再起動することにより、パラメーターの値を変更できます。

7.1.2 edb_dynatune_profile

`edb_dynatune_profile` パラメーターは、データベースサーバーで予想されるワークロードプロファイルに基づいてチューニングを制御するために使用されます。このパラメーターは、データベースサーバの起動時に有効になります。

`edb_dynatune_profile` の可能な値は次のとおりです。

Value	Usage
<i>oltp</i>	データベースサーバが重いオンライントランザクション処理ワークロードを処理している場合にお勧めします。
<i>reporting</i>	大量のデータのレポートに使用されるデータベースサーバに推奨されます。
<i>mixed</i>	トランザクション処理とデータレポートを組み合わせたサーバに推奨されます。

7.2 EDB の待機状態

EDB wait states contrib モジュールには 2 つの主要コンポーネントが含まれています。

7.2.1 EDB 待機状態バックグラウンドワーカー (EWSBW)

待機状態のバックグラウンドワーカーが共有プリロードライブラリの 1 つとして登録されている場合、EWSBW は実行中の各セッションを定期的にプローブします。

すべてのセッションについて、接続しているデータベース、セッションにログインしているユーザー、そのセッションで実行されているクエリ、待機している待機イベントなどの情報を収集します。

この情報は、`postgresql.conf` ファイルに追加される `edb_wait_states.directory` パラメーターで指定されたユーザー構成可能なパスとディレクトリフォルダーにある一連のファイルに保存されます。指定するパスは、相対パスではなく、絶対パスである必要があります。

以下では、Linux システムでのインストールプロセスについて説明します。

ステップ 1: EDB の待機状態は、`edb-asxx-server-edb_wait_states` RPM パッケージでインストールされます。ここで、`xx` は EDB Postgres Advanced Server のバージョン番号です。

ステップ 2: ワーカーを起動するには、`shared_preload_libraries` パラメーターを使用して `postgresql.conf` ファイルに登録する必要があります。例：

```
shared_preload_libraries = $libdir/edb_wait_states
```

ステップ 3: データベースサーバーを再起動します。再起動が成功すると、バックグラウンドワーカーはデータの収集を開始します。

ステップ 4: データを確認するには、次の拡張機能を作成します。

```
CREATE EXTENSION edb_wait_states;
```

ステップ 5: EDB 待機状態ワーカーを終了するには、`shared_preload_libraries` パラメーターから `$libdir/edb_wait_states` を削除し、データベースサーバーを再起動します。

以下では、Windows システムでのインストールプロセスについて説明します。

ステップ 1: EDB 待機状態モジュールは、EDB Modules インストーラーで StackBuilder Plus ユーティリティを呼び出すことによりインストールされます。画面の指示に従って、EDB Modules のインストールを完了します。

手順 2: ワーカーを登録するには、`postgresql.conf` ファイルを変更して、`shared_preload_libraries` 構成パラメーターに待機状態ライブラリを含めます。パラメーター値には以下を含める必要があります。

```
shared_preload_libraries = $libdir/edb_wait_states.dll
```

EDB 待機状態のインストールは、`edb_wait_states.dll` ライブラリファイルを次のパスに配置します。

```
C:\Program Files\edb\as14\lib\
```

ステップ 3: 変更を有効にするためにデータベースサーバーを再起動します。再起動が成功すると、バックグラウンドワーカーが起動し、データの収集を開始します。

ステップ 4: データを表示するには、次の拡張機能を作成します。

```
CREATE EXTENSION edb_wait_states;
```

インストーラーは、`edb_wait_states.control` ファイルを次のパスに配置します。

```
C:\Program Files\edb\as14\share\extension
```

7.2.2 待機状態のワーカーを終了する

EDB 待機状態ワーカーを終了するには、`DROP EXTENSION` コマンドを使用して `edb_wait_states` 拡張機能を削除します。次に、`postgresql.conf` ファイルを変更し、`shared_preload_libraries` パラメーターから `$libdir/edb_wait_states.dll` を削除します。`postgresql.conf` ファイルを変更して適用した後、データベースサーバーを再起動します。

7.2.3 ウェイトステートインターフェイス

このインターフェイスには、次のセクションにリストされている機能が含まれています。これらの各関数には、共通の入力パラメーターと出力パラメーターがあります。これらのパラメーターは次のとおりです。

- **start_ts** と **end_ts** (IN)。これらは一緒に、時間間隔とデータを指定します。start_ts のみを指定した場合は、start_ts 以降のデータが出力されます。end_ts のみを指定した場合、end_ts までのデータが出力されます。これらのいずれも指定されていない場合、すべてのデータが出力されます。すべての関数は異なるデータを出力します。以下、各関数の出力について説明します。
- **query_id** (OUT)。正規化されたクエリを識別します。これは、クエリから計算された内部ハッシュコードです。
- **session_id** (OUT)。セッションを識別します。
- **ref_start_ts** および **ref_end_ts**(OUT)。特定のデータポイントを含むファイルのタイムスタンプを提供します。データポイントは、待機イベントのサンプルレコード、クエリレコード、またはセッションレコードです。

各関数の構文については、次のセクションで説明します。

注釈: 以下のセクションに示す例は、異なるユーザーを使用して異なるデータベースに接続された4つの異なるセッションで同時に実行される次の3つのクエリに基づいています。

```
SELECT schemaname FROM pg_tables, pg_sleep(15) WHERE schemaname <>
pg_catalog; /* ran on 2 sessions */
SELECT tablename FROM pg_tables, pg_sleep(10) WHERE schemaname <>
pg_catalog;
SELECT tablename, schemaname FROM pg_tables, pg_sleep(10) WHERE schemaname
<> pg_catalog;
```

7.2.4 edb_wait_states_data

この機能は、EWSBW が収集したデータを読み取るために使用されます。

```
edb_wait_states_data(
  IN start_ts timestamptz default -infinity::timestamptz,
  IN end_ts timestamptz default infinity::timestamptz,
  OUT session_id int4,
  OUT <dbname> text,
  OUT <username> text,
  OUT <query> text,
  OUT <query_start_time> timestamptz,
  OUT <sample_time> timestamptz,
  OUT <wait_event_type> text,
```

(次のページに続く)

(前のページからの続き)

```
OUT <wait_event> text
)
```

この機能を使用して、次のことを確認できます。

すべてのセッションで指定された期間（`start_ts` および `end_ts` で定義）実行されているクエリと、それらが待機していた待機イベント。例：

```
SELECT query, session_id, wait_event_type, wait_event
FROM edb_wait_states_data(start_ts, end_ts);
```

特定の期間内のセッションの進行状況（つまり、セッションで実行されたクエリ（`session_id = 100000`）と、クエリが待機した待機イベント。例：

```
SELECT query, wait_event_type, wait_event
FROM edb_wait_states_data(start_ts, end_ts)
WHERE session_id = 100000;
```

サンプルが利用可能な期間。例：

```
SELECT min(sample_time), max(sample_time)
FROM edb_wait_states_data();
```

7.2.5 パラメーター

前述の一般的なパラメーターに加えて、出力の各行には次のものが含まれます。

`dbname`

セッションのデータベース

`username`

セッションにログインしているユーザー

`query`

セッションで実行されているクエリ

`query_start_time`

クエリが開始された時間

`sample_time`

待ちイベントデータを収集した時刻

`wait_event_type`

セッション（バックエンド）が待機している待機イベントのタイプ

`wait_event`

セッション（バックエンド）が待機している待機イベント

7.2.6 例

次に、`edb_wait_states_data()` 関数の出力例を示します。

```
edb=# SELECT * FROM edb_wait_states_data();
__OUTPUT__
- [ RECORD 1 ]-----+-----
->-----
session_id      | 4398
dbname          | edb
username        | enterprisedb
query           | SELECT schemaname FROM pg_tables, pg_sleep($1) WHERE schemaname <>
->$2
query_start_time | 17-AUG-18 11:56:05.271962 -04:00
sample_time      | 17-AUG-18 11:56:19.700236 -04:00
wait_event_type  | Timeout
wait_event       | PgSleep
- [ RECORD 2 ]-----+-----
->-----
session_id      | 4398
dbname          | edb
username        | enterprisedb
query           | SELECT schemaname FROM pg_tables, pg_sleep($1) WHERE schemaname <>
->$2
query_start_time | 17-AUG-18 11:56:05.271962 -04:00
sample_time      | 17-AUG-18 11:56:18.699938 -04:00
wait_event_type  | Timeout
wait_event       | PgSleep
- [ RECORD 3 ]-----+-----
->-----
session_id      | 4398
dbname          | edb
username        | enterprisedb
query           | SELECT schemaname FROM pg_tables, pg_sleep($1) WHERE schemaname <>
->$2
query_start_time | 17-AUG-18 11:56:05.271962 -04:00
sample_time      | 17-AUG-18 11:56:17.700253 -04:00
wait_event_type  | Timeout
wait_event       | PgSleep
                  .
                  .
                  .
```

7.2.7 edb_wait_states_queries

この機能は、EWSBW によってサンプリングされたクエリに関する情報を提供します。

```
edb_wait_states_queries(
  IN start_ts timestamptz default -infinity::timestamptz,
  IN end_ts timestamptz default infinity::timestamptz,
  OUT query_id int8,
  OUT <query> text,
  OUT ref_start_ts timestamptz
  OUT ref_end_ts timestamptz
)
```

新しいクエリファイルは定期的に作成されるため、特定の間隔に対応して複数のクエリファイルが生成される場合があります。

この関数は、指定された時間間隔と重なるクエリファイル内のすべてのクエリを返します。以下に示すクエリは、start_ts と end_ts の間でサンプリングされたクエリを含むクエリファイル内のすべてのクエリを提供します。

言い換えれば、この関数は、指定された間隔で実行されなかったクエリを出力する場合があります。ユーザーが edb_wait_states_data() を使用する必要があることを正確に知るため。

```
SELECT query FROM edb_wait_states_queries(start_ts, end_ts);
```

7.2.8 パラメーター

前述の一般的なパラメーターに加えて、出力の各行には次のものが含まれます。

query

正規化されたクエリテキスト

7.2.9 例

次に、edb_wait_states_queries() 関数の出力例を示します。

```
edb=# SELECT * FROM edb_wait_states_queries();
__OUTPUT__
- [ RECORD 1 ]+-----
->-----
query_id      | 4292540138852956818
query         | SELECT schemaname FROM pg_tables, pg_sleep($1) WHERE schemaname <> $2
ref_start_ts  | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts    | 18-AUG-18 11:52:38.698793 -04:00
- [ RECORD 2 ]+-----
```

(次のページに続く)

(前のページからの続き)

```

query_id      | 3754591102365859187
query         | SELECT tablename FROM pg_tables, pg_sleep($1) WHERE schemaname <> $2
ref_start_ts  | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts    | 18-AUG-18 11:52:38.698793 -04:00
- [ RECORD 3 ]+-----
→-----
query_id      | 349089096300352897
query         | SELECT tablename, schemaname FROM pg_tables, pg_sleep($1) WHERE
→schemaname <> $2
ref_start_ts  | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts    | 18-AUG-18 11:52:38.698793 -04:00

```

7.2.10 edb_wait_states_sessions

この機能は、EWSBW によってサンプリングされたセッションに関する情報を提供します。

```

edb_wait_states_sessions(
  IN start_ts timestampz default -infinity::timestampz,
  IN end_ts timestampz default infinity::timestampz,
  OUT session_id int4,
  OUT <dbname> text,
  OUT <username> text,
  OUT ref_start_ts timestampz
  OUT ref_end_ts timestampz
)

```

この機能を使用して、接続されたデータベースおよび/またはそれらのセッションを開始したユーザーを特定できます。例：

```

SELECT dbname, username, session_id
FROM edb_wait_states_sessions();

```

`edb_wait_states_queries()` と同様に、このファンクションは、指定された間隔内でサンプリングされたセッションだけでなく、指定された間隔内でサンプリングされたセッションを含むセッションファイルに記録されたすべてのセッションを出力します。それを識別するには、`edb_wait_states_data()` を使用する必要があります。

7.2.11 パラメーター

前述の一般的なパラメーターに加えて、出力の各行には次のものが含まれます。

dbname

セッションが接続されているデータベース

username

セッションのログインユーザー

7.2.12 例

次に、`edb_wait_states_sessions()` 関数の出力例を示します。

```
edb=# SELECT * FROM edb_wait_states_sessions();
__OUTPUT__
- [ RECORD 1 ]+-----
session_id   | 4340
dbname       | edb
username     | enterprisedb
ref_start_ts | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts   | 18-AUG-18 11:52:38.698793 -04:00
- [ RECORD 2 ]+-----
session_id   | 4398
dbname       | edb
username     | enterprisedb
ref_start_ts | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts   | 18-AUG-18 11:52:38.698793 -04:00
- [ RECORD 3 ]+-----
session_id   | 4410
dbname       | db1
username     | user1
ref_start_ts | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts   | 18-AUG-18 11:52:38.698793 -04:00
- [ RECORD 4 ]+-----
session_id   | 4422
dbname       | db2
username     | user2
ref_start_ts | 17-AUG-18 11:52:38.698793 -04:00
ref_end_ts   | 18-AUG-18 11:52:38.698793 -04:00
```

7.2.13 edb_wait_states_samples

この機能は、EWSBW によってサンプリングされた待機イベントに関する情報を提供します。

```
edb_wait_states_samples(
  IN start_ts timestamptz default -infinity::timestamptz,
  IN end_ts timestamptz default infinity::timestamptz,
  OUT query_id int8,
  OUT session_id int4,
  OUT <query_start_time> timestamptz,
  OUT <sample_time> timestamptz,
  OUT <wait_event_type> text,
  OUT <wait_event> text
)
```

通常、ユーザーはこの関数を直接呼び出す必要はありません。

7.2.14 パラメーター

前述の一般的なパラメーターに加えて、出力の各行には次のものが含まれます。

query_start_time

このセッションでクエリが開始された時間

sample_time

待ちイベントデータを収集した時刻

wait_event_type

セッションが待機している待機イベントのタイプ

wait_event

セッション（バックエンド）が待機している待機イベント

7.2.15 例

次に、edb_wait_states_samples() 関数の出力例を示します。

```
edb=# SELECT * FROM edb_wait_states_samples();
__OUTPUT__
- [ RECORD 1 ]-----+-----
query_id       | 4292540138852956818
session_id     | 4340
query_start_time | 17-AUG-18 11:56:00.39421 -04:00
sample_time     | 17-AUG-18 11:56:00.699934 -04:00
```

(次のページに続く)

(前のページからの続き)

```

wait_event_type | Timeout
wait_event      | PgSleep
- [ RECORD 2 ]-----+-----
query_id       | 4292540138852956818
session_id     | 4340
query_start_time | 17-AUG-18 11:56:00.39421 -04:00
sample_time    | 17-AUG-18 11:56:01.699003 -04:00
wait_event_type | Timeout
wait_event      | PgSleep
- [ RECORD 3 ]-----+-----
query_id       | 4292540138852956818
session_id     | 4340
query_start_time | 17-AUG-18 11:56:00.39421 -04:00
sample_time    | 17-AUG-18 11:56:02.70001 -04:00
wait_event_type | Timeout
wait_event      | PgSleep
- [ RECORD 4 ]-----+-----
query_id       | 4292540138852956818
session_id     | 4340
query_start_time | 17-AUG-18 11:56:00.39421 -04:00
sample_time    | 17-AUG-18 11:56:03.700081 -04:00
wait_event_type | Timeout
wait_event      | PgSleep
                .
                .
                .

```

7.2.16 edb_wait_states_purge

この関数は、start_ts の後に作成され、end_ts の前にエージング（ローテーション）されたすべてのサンプリングデータファイル（クエリ、セッション、および待機イベントサンプル）を削除します。

```

edb_wait_states_purge(
  IN start_ts timestampz default -infinity::timestampz,
  IN end_ts timestampz default infinity::timestampz
)

```

通常、ユーザーはこの機能を実行する必要はありません。バックエンドは保持期間に従ってそれらをパージする必要がありますが、何らかの理由でそれが起こらない場合、この機能が使用される場合があります。

サンプルが保持されている期間を知るには、その関数の前の例で説明したように edb_wait_states_data() を使用します。

7.2.17 例

edb_wait_states_purge() を実行する前の\$PGDATA/edb_wait_states ディレクトリ：

```
[root@localhost data]# pwd
/var/lib/edb/as14/data
[root@localhost data]# ls -l edb_wait_states
total 12
-rw----- 1 enterprisedb ... 253 Aug 17 11:56 edb_ws_queries_587836358698793_
→587922758698793
-rw----- 1 enterprisedb ... 1600 Aug 17 11:56 edb_ws_samples_587836358698793_
→587839958698793
-rw----- 1 enterprisedb ... 94 Aug 17 11:56 edb_ws_sessions_587836358698793_
→587922758698793
```

edb_wait_states_purge() 実行後の\$PGDATA/edb_wait_states ディレクトリ：

```
edb=# SELECT * FROM edb_wait_states_purge();
__OUTPUT__
edb_wait_states_purge
- -----
(1 row)
```

```
[root@localhost data]# pwd
/var/lib/edb/as14/data
[root@localhost data]# ls -l edb_wait_states
total 0
```

第8章 EDB clone schema

- EDB クローンスキーマ*は、ローカルまたはリモートデータベース（ソースデータベース）から受信側データベース（ターゲットデータベース）にスキーマとそのデータベースオブジェクトをコピーできるようにする EDB Postgres Advanced Server の拡張モジュールです。

ソースデータベースとターゲットデータベースは、同じ物理データベースでも、同じデータベースクラスター内の異なるデータベースでも、別のデータベースサーバーホスト上の別のデータベースクラスターで実行されている別のデータベースでもかまいません。

EDB クローンスキーマで次の機能を使用します。

- **localcopyschema**。この関数は、スキーマとそのデータベースオブジェクトのコピーをソースデータベースから同じデータベース（ターゲット）に作成しますが、元のスキーマ名とは異なります。このファンクションは、元のソーススキーマと結果のコピーを同じデータベース内に配置する場合に使用します。localcopyschema 関数については、

[ローカルコピースキーマ](#) を参照してください。

- **localcopyschema_nb**。この関数は localcopyschema と同じ目的を実行しますが、バックグラウンドジョブとして、したがって、関数が開始された端末を解放します。これは、**ノンブロッキング機能**と呼ばれます。localcopyschema_nb 関数については、

[localcopyschema_nb](#) を参照してください。

- **remotecopyschema**。この機能は、スキーマとそのデータベースオブジェクトのコピーをソースデータベースから別のターゲットデータベースに作成します。このファンクションは、元のソーススキーマと結果のコピーを2つの別個のデータベースに配置する場合に使用します。別のデータベースは、同じまたは異なる EDB Postgres Advanced Server データベースクラスターに存在できます。remotecopyschema 関数については、

[リモートコピースキーマ](#) を参照してください。

- **remotecopyschema_nb**。この関数は remotecopyschema と同じ目的を実行しますが、バックグラウンドジョブとして、したがって、関数が開始された端末を解放します。これは、**ノンブロッキング機能**と呼ばれます。remotecopyschema_nb 関数については、

[remotecopyschema_nb](#) を参照してください。

- **process_status_from_log**。この機能は、クローニング機能のステータスを表示します。クローニング機能呼び出し時に指定する必要があるログファイルから情報を取得します。process_status_from_log 関数については、

[process_status_from_log](#) を参照してください。

- **remove_log_file_and_job**。この機能は、クローニング機能によって作成されたログファイルを削除します。この関数は、関数のノンブロッキング形式で作成されたジョブを削除するためにも使用できま

す。`remove_log_file_and_job` 関数については、

[*remove_log_file_and_job*](#) を参照してください。

あるスキーマから別のスキーマに複製できるデータベースオブジェクトは次のとおりです。

- データ型
- パーティションテーブルを含むテーブル、外部テーブルを除く
- インデックス
- 制約
- シークエンス
- 定義を表示
- マテリアライズドビュー
- プライベート同義語
- テーブルトリガー、ただしイベントトリガーは除く
- ルール
- 機能
- 手続き
- パッケージ
- サポートされているすべてのオブジェクトタイプのコメント
- サポートされているすべてのオブジェクトタイプのアクセス制御リスト (ACL)

次のデータベースオブジェクトは複製できません。

- ラージオブジェクト (Postgres LOBs および BFILEs)
- テーブルの論理レプリケーション属性
- データベースリンク
- 外部データラッパー
- 外部テーブル
- イベントトリガー
- 拡張機能 (拡張機能に依存するオブジェクトのクローンについては、次の制限リストの 3 番目の箇条書きを参照してください。)
- 行レベルのセキュリティ
- ポリシー
- オペレータークラス

さらに、次の制限が適用されます。

- EDB クローンスキーマは、インストール中に EDB Postgres Advanced Server EDB Postgres Advanced Server の Dialect ダイアログで **Compatible with Oracle** 方言が指定された場合、またはテキストモードのインストールまたはクラスター初期化中に `--redwood-like` キーワードが含まれる場合にのみサポートされます。
- ファンクション、プロシージャ、トリガー、パッケージなど内のソースコードは、ターゲットスキーマにコピーした後に変更されません。このようなプログラムにスキーマ名を持つオブジェクトへのコード参照が含まれている場合、そのようなスキーマ名がターゲットスキーマ内で一貫していないと、プログラムはターゲットスキーマでの呼び出しに失敗する場合があります。
- クロススキーマオブジェクトの依存関係が解決されません。ターゲットスキーマのオブジェクトが別のスキーマのオブジェクトに依存している場合、この依存関係はクローニング機能では解決されません。
- リモートクローニングの場合、ソーススキーマ内のオブジェクトが拡張機能に依存している場合、リモートクローニングファンクションを呼び出す前にこの拡張機能をリモートデータベースのパブリックスキーマに作成する必要があります。
- 最大で 16 のコピージョブをクローンスキーマに並列で実行できますが、各ジョブにはテーブルデータを並列でコピーする最大 16 のワーカープロセスがあります。
- バックグラウンドワーカーによって実行されているクエリはキャンセルできません。

次のセクションでは、データベースで EDB クローンスキーマを設定する方法について説明します。

8.1 セットアッププロセス

EDB Clone Schema 機能でソースまたはターゲットデータベースとして使用するには、PL / Perl 言語とともにいくつかの拡張機能をデータベースにインストールする必要があります。

さらに、データベースサーバーの `postgresql.conf` ファイルの一部の構成パラメーターを変更すると効果がある場合があります。

以下は、これらの要件のセットアップ手順です。

8.1.1 拡張機能と PL / Perl のインストール

以下では、必要な拡張機能と PL / Perl 言語をインストールする手順について説明します。

これらの手順は、EDB Clone Schema 機能によってソースまたはターゲットデータベースとして使用されるデータベースで実行する必要があります。

ステップ 1： 次の拡張機能をデータベースにインストールする必要があります。

- `postgres_fdw`
- `dblink`
- `adminpack`
- `pgagent`

pgagent 拡張機能を作成する前に、pgAgent がインストールされていることを確認します。Linux では、edb-asxx-pgagent RPM パッケージを使用できます。ここで、xx は EDB Postgres Advanced Server のバージョン番号です。Windows、StackBuilder Plus を使用して pgAgent をダウンロードしてインストールします。

前述の拡張機能は、まだ存在しない場合は、次のコマンドでインストールできます。

```
CREATE EXTENSION postgres_fdw SCHEMA public;
CREATE EXTENSION dblink SCHEMA public;
CREATE EXTENSION adminpack;
CREATE EXTENSION pgagent;
```

CREATE EXTENSION コマンドの使用の詳細については、次の PostgreSQL コードドキュメントを参照してください。

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

ステップ 2: postgresql.conf ファイルを変更します。

次の例に示すように、\$libdir/parallel_clone を shared_preload_libraries 構成パラメーターに追加して、postgresql.conf ファイルを変更します。

```
shared_preload_libraries = $libdir/dbms_pipe,$libdir/dbms_aq,$libdir/parallel_clone
```

ステップ 3: Perl 手続き言語 (PL / Perl) をデータベースにインストールし、CREATE TRUSTED LANGUAGE plperl コマンドを実行する必要があります。Linux の場合、edb-asxx-server-plperl RPM パッケージを使用して PL / Perl をインストールします。ここで、xx は EDB Postgres Advanced Server のバージョン番号です。Windows の場合、EDB Postgres 言語パックを使用します。EDB 言語パックについては、次の* EDB Postgres 言語パックガイド*を参照してください。

<https://www.enterprisedb.com/docs>

ステップ 4: スーパーユーザーとしてデータベースに接続し、次のコマンドを実行します。

```
CREATE TRUSTED LANGUAGE plperl;
```

CREATE LANGUAGE コマンドの使用の詳細については、次の PostgreSQL コードドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/sql-createlanguage.html>

8.1.2 構成パラメーターの設定

以下のセクションでは、postgresql.conf ファイルで変更が必要になる場合がある構成パラメーターについて説明します。

8.1.3 パフォーマンス構成パラメーター

1つのトランザクションの一部として大規模なスキーマをコピーするようにシステムを調整する必要があります。

構成パラメーターのチューニングは、クローニング機能で参照されるソースデータベースサーバー用です。

調整が必要な `postgresql.conf` ファイルの構成パラメーターは次のとおりです。

- **work_mem。** 一時ディスクファイルに書き込む前に、内部並べ替え操作とハッシュテーブルが使用するメモリの量を指定します。
- **maintenance_work_mem。** VACUUM、CREATE INDEX、ALTER TABLE ADD FOREIGN KEYなどのメンテナンス操作で使用するメモリの最大量を指定します。
- **max_worker_processes。** システムがサポートできるバックグラウンドプロセスの最大数を設定します。
- **checkpoint_timeout。** 自動 WAL チェックポイント間の最大時間（秒単位）。
- **checkpoint_completion_target。** チェックポイント間の合計時間の割合として、チェックポイント完了のターゲットを指定します。
- **checkpoint_flush_after。** チェックポイントの実行中に `checkpoint_flush_after` バイトを超える書き込みが行われた場合は常に、OS に基になるストレージにこれらの書き込みを発行させます。
- **max_wal_size。** 自動 WAL チェックポイント間で WAL を成長させる最大サイズ。
- **max_locks_per_transaction。** このパラメーターは、各トランザクションに割り当てられるオブジェクトロックの平均数を制御します。すべてのトランザクションのロックがロックテーブルに収まる限り、個々のトランザクションはより多くのオブジェクトをロックできます。

構成パラメーターについては、次の PostgreSQL コアドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/runtime-config.html>

8.1.4 ステータスログ

クローニング機能によるステータスロギングは、クローニング機能の呼び出し時に接続しているデータベースサーバーの `postgresql.conf` ファイルの `log_directory` パラメーターで指定されたディレクトリにログファイルを作成します。

デフォルトの場所は、次に示すように `PGDATA/log` です。

```
# log_directory = log           # directory where log files are written,
                                # can be absolute or relative to PGDATA
```

このディレクトリは、クローニング機能を実行する前に存在する必要があります。

ログファイルの名前は、クローニング機能呼び出すときにパラメーターリストで指定したものによって決まります。

ログファイルからステータスを表示するには、`process_status_from_log` 関数を使用します。

ログファイルを削除するには、`remove_log_file_and_job` ファンクションを使用するか、単に `log` ディレクトリに移動して手動で削除します。

8.1.5 EDB クローンスキーマのインストール

EDB Clone Schema をインストールする手順は次のとおりです。

これらの手順は、EDB Clone Schema 機能によってソースまたはターゲットデータベースとして使用されるデータベースで実行する必要があります。

ステップ 1: 以前に古いバージョンの `edb_cloneschema` 拡張機能をインストールしたことがある場合は、次のコマンドを実行する必要があります。

```
DROP EXTENSION parallel_clone CASCADE;
```

このコマンドは、`edb_cloneschema` 拡張機能も削除します。

ステップ 2: 次のコマンドを使用して拡張機能をインストールします。

```
CREATE EXTENSION parallel_clone SCHEMA public;
```

```
CREATE EXTENSION edb_cloneschema;
```

`edb_cloneschema` 拡張機能を作成する前に、必ず `parallel_clone` 拡張機能を作成してください。

8.1.6 外部サーバーとユーザーマッピングの作成

ローカルクローニング関数 `localcopyschema` または `localcopyschema_nb` のいずれかを使用する場合、必要なパラメーターの 1 つに、データベースサーバーと、複製されたスキーマのソースおよびレシーバーであるデータベースを識別するための単一の外部サーバーが含まれます。

リモートクローニング機能 `remotecopyschema` または `remotecopyschema_nb` のいずれかを使用する場合、2 つの必須パラメーターに 2 つの外部サーバーが含まれます。最初のパラメーターとして指定された外部サーバーは、複製されたスキーマのプロバイダーであるデータベースとともにソースデータベースサーバーを識別します。2 番目のパラメーターとして指定された外部サーバーは、ターゲットデータベースサーバーと、複製されたスキーマの受信者であるデータベースを識別します。

外部サーバーごとに、ユーザーマッピングを作成する必要があります。選択したデータベーススーパーユーザーがクローニングファンクションを呼び出す場合、ファンクションを呼び出すデータベーススーパーユーザーは、クローニングファンクションのパラメーターとして指定された外部サーバーへのアクセス権を持つデータベースユーザー名とパスワードにマップされている必要があります。

外部データ、外部サーバー、およびユーザーマッピングに関する一般的な情報については、次の PostgreSQL コードキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/ddl-foreign-data.html>

次の 2 つのセクションでは、これらの外部サーバーとユーザーマッピングがどのように定義されるかについて説明します。

8.1.7 ローカルクローニング実行の外部サーバーとユーザーマッピング

localcopyschema および localcopyschema_nb ファンクションの場合、ソーススキーマとターゲットスキーマは両方とも同じデータベースサーバーの同じデータベース内にあります。したがって、これらの機能に対して定義および指定する必要がある外部サーバーは 1 つだけです。この外部サーバーは、**ローカルサーバー**とも呼ばれます。

このサーバーは、localcopyschema または localcopyschema_nb ファンクションを呼び出すときに接続する必要があるため、ローカルサーバーと呼ばれます。

ユーザーマッピングは、外部サーバーの接続と認証情報を定義します。

この外部サーバーとユーザーマッピングは、クローニングが発生するローカルサーバーのデータベース内に作成する必要があります。

ユーザーマッピングが定義されるデータベースユーザーは、スーパーユーザーであり、EDB Clone Schema ファンクションを呼び出すときにローカルサーバーに接続している必要があります。

次の例では、複製するスキーマを含むデータベースの外部サーバーを作成し、複製されたスキーマも受信します。

```
CREATE SERVER local_server FOREIGN DATA WRAPPER postgres_fdw
  OPTIONS(
    host localhost,
    port 5444,
    dbname edb
  );
```

CREATE SERVER コマンドの使用の詳細については、次の PostgreSQL コードドキュメントを参照してください。

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

このサーバーのユーザーマッピングは次のとおりです。

```
CREATE USER MAPPING FOR enterprisedb SERVER local_server
  OPTIONS (
    user enterprisedb,
    password password
  );
```

CREATE USER MAPPING コマンドの使用の詳細については、次の PostgreSQL コードドキュメントを参照してください。

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

次の psql コマンドは、外部サーバーとユーザーのマッピングを表示します。

```
edb=# \des+
__OUTPUT__
List of foreign servers
- [ RECORD 1 ]-----+-----
```

(次のページに続く)

(前のページからの続き)

Name	local_server
Owner	enterprisedb
Foreign-data wrapper	postgres_fdw
Access privileges	
Type	
Version	
FDW options	(host localhost, port 5444, dbname edb)
Description	

edb=# \deu+

__OUTPUT__

List of user mappings

Server		User name		FDW options
-----+-----+-----				
local_server		enterprisedb		("user" enterprisedb, password password)
(1 row)				

データベーススーパーユーザー **enterprisedb** がクローニング機能呼び出すと、データベースユーザー **enterprisedb** とそのパスワードが使用され、ポート 5444 でデータベース **edb** に接続します。

この場合、ローカルの **edb** データベースへの接続に使用されるマップされたデータベースユーザー **enterprisedb** とデータベースユーザー **enterprisedb** は、たまたま同じ同一のデータベースユーザーですが、それは絶対的な要件ではありません。

これらの外部サーバーとユーザーマッピングの例の具体的な使用法については、**localcopyschema** の例を参照してください。

8.1.8 リモートクローニング機能の外部サーバーとユーザーマッピング

remotecopyschema および **remotecopyschema_nb** ファンクションの場合、ソーススキーマとターゲットスキーマは、同じまたは異なるデータベースサーバーの異なるデータベースにあります。したがって、これらの機能には 2 つの外部サーバーを定義および指定する必要があります。

元のデータベースサーバーと、複製されるソーススキーマを含むそのデータベースを定義する外部サーバーは、**ソースサーバー**または**リモートサーバー**と呼ばれます。

複製するスキーマを受信するデータベースサーバーとそのデータベースを定義する外部サーバーは、**ターゲットサーバー**または**ローカルサーバー**と呼ばれます。

ターゲットサーバーは、**remotecopyschema** または **remotecopyschema_nb** ファンクションを呼び出すときに接続する必要があるため、ローカルサーバーとも呼ばれます。

ユーザーマッピングは、外部サーバーの接続と認証情報を定義します。

これらの外部サーバーとユーザーマッピングはすべて、ターゲット/ローカルサーバーのターゲットデータベース内に作成する必要があります。

ユーザーマッピングが定義されるデータベースユーザーは、スーパーユーザーであり、EDB Clone Schema フังก์ションを呼び出すときにローカルサーバーに接続している必要があります。

次の例では、複製されたスキーマを受信するローカルのターゲットデータベースの外部サーバーを作成します。

```
CREATE SERVER tgt_server FOREIGN DATA WRAPPER postgres_fdw
  OPTIONS(
    host localhost,
    port 5444,
    dbname tgtdb
  );
```

このサーバーのユーザーマッピングは次のとおりです。

```
CREATE USER MAPPING FOR enterprisedb SERVER tgt_server
  OPTIONS (
    user tgtuser,
    password tgtpassword
  );
```

次の例では、複製されたスキーマのソースとなるリモートのソースデータベースの外部サーバーを作成します。

```
CREATE SERVER src_server FOREIGN DATA WRAPPER postgres_fdw
  OPTIONS(
    host 192.168.2.28,
    port 5444,
    dbname srcdb
  );
```

このサーバーのユーザーマッピングは次のとおりです。

```
CREATE USER MAPPING FOR enterprisedb SERVER src_server
  OPTIONS (
    user srcuser,
    password srcpassword
  );
```

次の `psql` コマンドは、外部サーバーとユーザーマッピングを表示します。

```
tgtdb=# \des+
__OUTPUT__
List of foreign servers
- [ RECORD 1 ]-----+-----
Name                | src_server
Owner                | tgtuser
Foreign-data wrapper | postgres_fdw
```

(次のページに続く)

(前のページからの続き)

```

Access privileges |
Type             |
Version          |
FDW options       | (host 192.168.2.28, port 5444, dbname srcdb)
Description       |
- [ RECORD 2 ]-----+-----
Name             | tgt_server
Owner            | tgtuser
Foreign-data wrapper | postgres_fdw
Access privileges |
Type             |
Version          |
FDW options       | (host localhost, port 5444, dbname tgtddb)
Description       |

```

tgtddb=# \deu+

__OUTPUT__

List of user mappings

```

Server | User name | FDW options
-----+-----
src_server | enterprisedb | ("user" srcuser, password srcpassword)
tgt_server | enterprisedb | ("user" tgtuser, password tgtpassword)
(2 rows)

```

データベーススーパーユーザー `enterprisedb` がクローン機能呼び出すと、パスワード `tgtpassword` を持つデータベースユーザー `tgtuser` を使用して、ポート 5444 でデータベース `tgtddb` に接続します。

さらに、パスワード `srcpassword` を持つデータベースユーザー `srcuser` は、ポート 5444 を使用してデータベース `srcdb` に接続します。

注釈: ソースデータベース ``srcdb`` を実行しているデータベースサーバーの ``pg_hba.conf`` ファイルに、ターゲットサーバーの場所（次の例のアドレス ``192.168.2.27``）からの接続を許可する適切なエントリがあることを確認してください。ソースサーバーとデータベースを定義するサーバー ``src_server``。

```

# TYPE DATABASE USER ADDRESS METHOD

# "local" is for Unix domain socket connections only

local all all md5
# IPv4 local connections:

host srcdb srcuser 192.168.2.27/32 md5

```

これらの外部サーバーとユーザーマッピングの例の具体的な使用法については、`remotecopyschema` の例

を参照してください。

8.2 EDB クローンスキーマ機能

`parallel_clone` および `edb_cloneschema` 拡張機能がインストールされると、EDB クローンスキーマファンクションが `edb_util` スキーマに作成されます。

EDB Clone Schema 機能を使用する前に、次の条件を確認してください。

- ターゲットまたはローカルデータベースの外部サーバーの `CREATE USER MAPPING` コマンドで定義されたデータベーススーパーユーザーとして、ターゲットまたはローカルデータベースに接続しています。
- `edb_util` スキーマが検索パスにあるか、クローニング機能を `edb_util` プレフィックスを付けて呼び出されます。
- 対象データベースに対象スキーマが存在しません。
 - リモートコピー機能を使用する場合、`on_tblspace` パラメーターが `true` に設定される場合、ターゲットデータベースクラスターには、ソーススキーマ内のオブジェクトによって参照されるすべてのテーブルスペースが含まれます。それ以外の場合、それらのデータベースオブジェクトの DDL ステートメントの作成は失敗しますターゲットスキーマ。これにより、クローン作成プロセスが失敗します。
 - リモートコピー機能を使用する場合、`copy_acls` パラメーターが `true` に設定されている場合、ソーススキーマのオブジェクトに対する `GRANT` 特権を持つすべてのロールがターゲットデータベースクラスターに存在します。それ以外の場合、ターゲットでこれらのロールへの特権の付与に失敗しますスキーマ。これにより、クローン作成プロセスが失敗します。
 - 非ブロック形式の関数を使用する場合、ターゲットデータベースに対して `pgAgent` が実行されています。

`pgAgent` の詳細については、次の場所にある `pgAdmin` ドキュメントを参照してください。

<https://www.pgadmin.org/docs/pgadmin4/dev/pgagent.html>

`pgAgent` は EDB Postgres Advanced Server のコンポーネントとして提供されていることに注意してください。

8.2.1 ローカルコピースキーマ

`localcopyschema` ファンクションは、`source_fdw` 外部サーバー内で指定されたローカルデータベース内のスキーマとそのデータベースオブジェクトを、ソーススキーマから同じデータベース内の指定されたターゲットスキーマにコピーします。

```
localcopyschema(
  <source_fdw> TEXT,
  <source_schema> TEXT,
  <target_schema> TEXT,
  <log_filename> TEXT
```

(次のページに続く)

```
[, <on_tblspace> BOOLEAN
[, <verbose_on> BOOLEAN
[, <copy_acls> BOOLEAN
[, <worker_count> INTEGER ]]]
)
```

BOOLEAN 値が関数によって返されます。関数が成功すると、**true** が返されます。関数が失敗すると、**false** が返されます。

source_fdw, **source_schema**, **target_schema**、および **log_filename** は必須パラメーターですが、他のすべてのパラメーターはオプションです。

8.2.2 パラメーター

source_fdw

データベースオブジェクトのクローン元の **postgres_fdw** 外部データラッパーによって管理される外部サーバーの名前。

source_schema

データベースオブジェクトのクローン元のスキーマの名前。

target_schema

ソーススキーマからデータベースオブジェクトのクローンを作成するスキーマの名前。

log_filename

関数からの情報が記録されるログファイルの名前。ログファイルは、**postgresql.conf** ファイルの **log_directory** 構成パラメーターで指定されたディレクトリの下に作成されます。

on_tblspace

BOOLEAN 値を使用して、データベースオブジェクトを表領域内に作成するかどうかを指定します。**false** が指定されている場合、**TABLESPACE** 句はターゲットスキーマに追加されたときに該当する **CREATE DDL** ステートメントに含まれません。**true** が指定されている場合、**TABLESPACE** 句はターゲットスキーマに追加されるときに **CREATE DDL** ステートメントに含まれます。**on_tblspace** パラメーターを省略した場合、デフォルト値は **false** です。

verbose_on

ターゲットスキーマでオブジェクトを作成するときに **DDL** を **log_filename** に出力するかどうかを指定する BOOLEAN 値。**false** が指定されている場合、**DDL** は出力されません。**true** が指定されている場合、**DDL** が出力されます。省略した場合のデフォルト値は **false** です。

copy_acls

BOOLEAN 値を使用して、ターゲットスキーマでオブジェクトを作成するときにアクセス制御リスト (ACL) を含めるかどうかを指定します。アクセス制御リストは、**GRANT** 特権ステートメントのセットです。**false** が指定されている場合、アクセス制御リストはターゲットスキーマに含まれません。**true** が指定されてい

る場合、ターゲットスキーマのアクセス制御リストが含まれます。copy_acls パラメーターを省略した場合、デフォルト値は false です。

worker_count

クローンを並列実行するバックグラウンドワーカーの数。省略した場合のデフォルト値は 1 です。

8.2.3 例

次の例は、一連のデータベースオブジェクトを含むスキーマ edb のターゲットスキーマ edbcopy へのクローニングを示しています。両方とも local_server で定義されたデータベース edb 内にあります。

この例は次の環境の場合です。

- データベースサーバ稼働ホスト：localhost
 - データベースサーバのポート：5444
 - クローンのデータベースソース/ターゲット：edb
 - 外部サーバ (local_server) と先行箇条書きの情報によるユーザーマッピング
 - ソーススキーマ：edb
- 対象スキーマ：edbcopy
 - localcopyschema: enterprisedb を呼び出すデータベーススーパーユーザー

関数を呼び出す前に、データベースユーザー enterprisedb によってデータベース edb に接続されます。

```
edb=# SET search_path TO "$user",public,edb_util;
SET
edb=# SHOW search_path;
__OUTPUT__
      search_path
-----
"$user", public, edb_util
(1 row)
```

```
edb=# SELECT localcopyschema (local_server,edb,edbcopy,clone_edb_edbcopy);
__OUTPUT__
localcopyschema
-----
t
(1 row)
```

次に、process_status_from_log 関数を使用したロギングステータスを表示します。

```
edb=# SELECT process_status_from_log(clone_edb_edbcopy);
__OUTPUT__
```

(次のページに続く)

(前のページからの続き)

```

process_status_from_log
-----
→ -----
(FINISH,"2017-06-29 11:07:36.830783-04",3855,INFO,"STAGE: FINAL","successfully
→cloned schema")
(1 row)

```

クローンが完了した後、edbcopy スキーマにコピーされたデータベースオブジェクトの一部を次に示します。

```

edb=# SET search_path TO edbcopy;
SET
edb=# \dt+
__OUTPUT__

              List of relations
 Schema | Name   | Type | Owner      | Size   | Description
-----+-----+-----+-----+-----+-----
 edbcopy | dept   | table | enterprisedb | 8192 bytes |
 edbcopy | emp    | table | enterprisedb | 8192 bytes |
 edbcopy | jobhist | table | enterprisedb | 8192 bytes |
(3 rows)

```

```

edb=# \dv
__OUTPUT__

              List of relations
 Schema | Name      | Type | Owner      |
-----+-----+-----+-----+
 edbcopy | salesemp  | view | enterprisedb |
(1 row)

```

```

edb=# \di
__OUTPUT__

              List of relations
 Schema | Name          | Type | Owner      | Table
-----+-----+-----+-----+-----
 edbcopy | dept_dname_uq | index | enterprisedb | dept
 edbcopy | dept_pk       | index | enterprisedb | dept
 edbcopy | emp_pk        | index | enterprisedb | emp
 edbcopy | jobhist_pk    | index | enterprisedb | jobhist
(4 rows)

```

```

edb=# \ds
__OUTPUT__

              List of relations

```

(次のページに続く)

(前のページからの続き)

Schema	Name	Type	Owner
edbcopy	next_empno	sequence	enterprisedb

(1 row)

```
edb=# SELECT DISTINCT schema_name, name, type FROM user_source WHERE
schema_name = EDBCOPY ORDER BY type, name;
```

__OUTPUT__

schema_name	name	type
EDBCOPY	EMP_COMP	FUNCTION
EDBCOPY	HIRE_CLERK	FUNCTION
EDBCOPY	HIRE_SALESMAN	FUNCTION
EDBCOPY	NEW_EMPNO	FUNCTION
EDBCOPY	EMP_ADMIN	PACKAGE
EDBCOPY	EMP_ADMIN	PACKAGE BODY
EDBCOPY	EMP_QUERY	PROCEDURE
EDBCOPY	EMP_QUERY_CALLER	PROCEDURE
EDBCOPY	LIST_EMP	PROCEDURE
EDBCOPY	SELECT_EMP	PROCEDURE
EDBCOPY	EMP_SAL_TRIG	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_a_19991"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_a_19992"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_a_19999"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_a_20000"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_a_20004"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_a_20005"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_c_19993"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_c_19994"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_c_20001"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_c_20002"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_c_20006"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_c_20007"	TRIGGER
EDBCOPY	USER_AUDIT_TRIG	TRIGGER

(24 rows)

8.2.4 localcopyschema_nb

localcopyschema_nb ファンクションは、source_fdw 外部サーバー内で指定されたローカルデータベース内のスキーマとそのデータベースオブジェクトを、ソーススキーマから同じデータベース内の指定されたターゲットスキーマにコピーしますが、pgAgent に送信されたジョブとして非ブロッキング方法でコピーします。

```
localcopyschema_nb(  
  <source_fdw> TEXT,  
  <source> TEXT,  
  <target> TEXT,  
  <log_filename> TEXT  
  [, <on_tblspace> BOOLEAN  
  [, <verbose_on> BOOLEAN  
  [, <copy_acls> BOOLEAN  
  [, <worker_count> INTEGER ]]]  
)
```

INTEGER 値のジョブ ID は、pgAgent に送信されたジョブの関数によって返されます。関数が失敗した場合、null が返されます。

source_fdw, source, target、および log_filename は必須パラメーターですが、他のすべてのパラメーターはオプションです。

pgAgent ジョブが完了したら、remove_log_file_and_job ファンクションでジョブを削除します。

8.2.5 パラメーター

source_fdw

データベースオブジェクトのクローン元の postgres_fdw 外部データラッパーによって管理される外部サーバーの名前。

source

データベースオブジェクトのクローン元のスキーマの名前。

target

ソーススキーマからデータベースオブジェクトのクローンを作成するスキーマの名前。

log_filename

関数からの情報が記録されるログファイルの名前。ログファイルは、postgresql.conf ファイルの log_directory 構成パラメーターで指定されたディレクトリの下に作成されます。

on_tblspace

BOOLEAN 値を使用して、データベースオブジェクトを表領域内に作成するかどうかを指定します。false が指定されている場合、TABLESPACE 句はターゲットスキーマに追加されたときに該当する CREATE DDL ステートメントに含まれません。true が指定されている場合、TABLESPACE 句はターゲットスキーマに追

加されるときに CREATE DDL ステートメントに含まれます。on_tblspace パラメーターを省略した場合、デフォルト値は false です。

verbose_on

ターゲットスキーマでオブジェクトを作成するときに DDL を log_filename に出力するかどうかを指定する BOOLEAN 値。false が指定されている場合、DDL は出力されません。true が指定されている場合、DDL が出力されます。省略した場合のデフォルト値は false です。

copy_acls

BOOLEAN 値を使用して、ターゲットスキーマでオブジェクトを作成するときにアクセス制御リスト (ACL) を含めるかどうかを指定します。アクセス制御リストは、GRANT 特権ステートメントのセットです。false が指定されている場合、アクセス制御リストはターゲットスキーマに含まれません。true が指定されている場合、ターゲットスキーマのアクセス制御リストが含まれます。copy_acls パラメーターを省略した場合、デフォルト値は false です。

worker_count

クローンを並列実行するバックグラウンドワーカーの数。省略した場合のデフォルト値は 1 です。

8.2.6 例

localcopyschema の例と同じクローニング操作が実行されますが、ノンブロッキングファンクション localcopyschema_nb を使用します。

次のコマンドを使用して、pgAgent が適切なローカルデータベースで実行されているかどうかを確認できます。

```
[root@localhost ~]# ps -ef | grep pgagent
root      4518      1  0 11:35 pts/1    00:00:00 pgagent -s /tmp/pgagent_edb_log
hostaddr=127.0.0.1 port=5444 dbname=edb user=enterprisedb password=password
root      4525    4399  0 11:35 pts/1    00:00:00 grep --color=auto pgagent
```

pgAgent が起動していない場合、以下のように起動できます。pgagent プログラムファイルは、EDB Postgres Advanced Server インストールディレクトリの bin サブディレクトリにあります。

```
[root@localhost bin]# ./pgagent -l 2 -s /tmp/pgagent_edb_log hostaddr=127.0.0.1
↪port=5444
dbname=edb user=enterprisedb password=password
```

注釈: pgagent -l 2 オプションは pgAgent を `DEBUG` モードで起動し、継続的なデバッグ情報を -s オプションで指定されたログファイルに記録します。-l オプションにより小さい値を使用するか、完全に省略してより少ない情報を記録します。

localcopyschema_nb ファンクションは、例に 4 として示されているジョブ ID を返します。

```
edb=# SELECT edb_util.localcopyschema_nb (local_server,edb,edbcopy,clone_edb_edbcopy);
__OUTPUT__
localcopyschema_nb
-----
4
(1 row)
```

以下にジョブのステータスが表示されます。

```
edb=# SELECT edb_util.process_status_from_log(clone_edb_edbcopy);
__OUTPUT__
process_status_from_log
-----
→ (FINISH,"29-JUN-17 11:39:11.620093 -04:00",4618,INFO,"STAGE: FINAL","successfully_
→ cloned schema")
(1 row)
```

以下は、pgAgent ジョブを削除します。

```
edb=# SELECT edb_util.remove_log_file_and_job (4);
__OUTPUT__
remove_log_file_and_job
-----
t
(1 row)
```

8.2.7 リモートコピースキーマ

`remotecopyschema` ファンクションは、スキーマとそのデータベースオブジェクトを、`source_fdw` 外部サーバー内で指定されたリモートソースデータベースのソーススキーマから、`target_fdw` 外部サーバー内で指定されたローカルターゲットデータベースのターゲットスキーマにコピーします。

```
remotecopyschema(
  <source_fdw> TEXT,
  <target_fdw> TEXT,
  <source_schema> TEXT,
  <target_schema> TEXT,
  <log_filename> TEXT
  [, <on_tblspace> BOOLEAN
  [, <verbose_on> BOOLEAN
  [, <copy_acls> BOOLEAN
  [, <worker_count> INTEGER ]]]
)
```

BOOLEAN 値が関数によって返されます。関数が成功すると、true が返されます。関数が失敗すると、false が返されます。

source_fdw, target_fdw, source_schema, target_schema、および log_filename は必須パラメーターですが、他のすべてのパラメーターはオプションです。

8.2.8 パラメーター

source_fdw

データベースオブジェクトのクローン元の postgres_fdw 外部データラッパーによって管理される外部サーバーの名前。

target_fdw

データベースオブジェクトのクローン先の postgres_fdw 外部データラッパーによって管理される外部サーバーの名前。

source_schema

データベースオブジェクトのクローン元のスキーマの名前。

target_schema

ソーススキーマからデータベースオブジェクトのクローンを作成するスキーマの名前。

log_filename

関数からの情報が記録されるログファイルの名前。ログファイルは、postgresql.conf ファイルの log_directory 構成パラメーターで指定されたディレクトリの下に作成されます。

on_tblspace

BOOLEAN 値を使用して、データベースオブジェクトを表領域内に作成するかどうかを指定します。false が指定されている場合、TABLESPACE 句はターゲットスキーマに追加されたときに該当する CREATE DDL ステートメントに含まれません。true が指定されている場合、TABLESPACE 句はターゲットスキーマに追加されるときに CREATE DDL ステートメントに含まれます。on_tblspace パラメーターを省略した場合、デフォルト値は false です。

注釈: true が指定され、データベースオブジェクトに `TABLESPACE` 句があるが、その表領域がターゲットデータベースクラスターに存在しない場合、クローン作成機能は失敗します。

verbose_on

ターゲットスキーマでオブジェクトを作成するときに DDL を log_filename に出力するかどうかを指定する BOOLEAN 値。false が指定されている場合、DDL は出力されません。true が指定されている場合、DDL が出力されます。省略した場合のデフォルト値は false です。

copy_acls

BOOLEAN 値を使用して、ターゲットスキーマでオブジェクトを作成するときにアクセス制御リスト (ACL) を含めるかどうかを指定します。アクセス制御リストは、GRANT 特権ステートメントのセットです。false

が指定されている場合、アクセス制御リストはターゲットスキーマに含まれません。`true` が指定されている場合、ターゲットスキーマのアクセス制御リストが含まれます。`copy_acls` パラメーターを省略した場合、デフォルト値は `false` です。

注釈: `true` が指定され、`GRANT` 権限を持つロールがターゲットデータベースクラスターに存在しない場合、クローン作成機能は失敗します。

worker_count

クローンを並列実行するバックグラウンドワーカーの数。省略した場合のデフォルト値は 1 です。

8.2.9 例

次の例は、`src_server` で定義されたデータベース `srcdb` 内のスキーマ `srcschema` を、`tgt_server` で定義されたデータベース `tgtadb` 内のターゲットスキーマ `tgtschema` にクローニングする方法を示しています。

ソースサーバー環境：

- ソースデータベースサーバーが実行されているホスト：192.168.2.28
- ソースデータベースサーバーのポート：5444
- クローンのデータベースソース：`srcdb`
- 外部サーバー (`src_server`) と先行箇条書きの情報によるユーザーマッピング
- ソーススキーマ：`srcschema`

ターゲットサーバー環境：

- 対象データベースサーバが稼働しているホスト：localhost
- ターゲットデータベースサーバのポート：5444
- クローンのデータベースターゲット：`tgtadb`
- 外部サーバー (`tgt_server`) と先行箇条書きの情報によるユーザーマッピング
- 対象スキーマ：`tgtschema`
- `remotecopyschema`: `enterprisedb` を呼び出すデータベーススーパーユーザー

関数を呼び出す前に、データベースユーザー `enterprisedb` によってデータベース `tgtadb` に接続されます。この機能には 4 の `worker_count` が指定されています。

```
tgtadb=# SELECT edb_util.remotecopyschema
(src_server,tgt_server,srcschema,tgtschema,clone_rmt_src_tgt,worker_count => 4);
__OUTPUT__
remotecopyschema
- - - - -
```

(次のページに続く)

(前のページからの続き)

```
t
(1 row)
```

次に、クローン作成プロセスのさまざまな時点でのログファイルのステータスを表示します。

```
tgtdb=# SELECT edb_util.process_status_from_log(clone_rmt_src_tgt);
__OUTPUT__

                                process_status_from_log
-----
-> -----
-
(RUNNING,"28-JUN-17 13:18:05.299953 -04:00",4021,INFO,"STAGE: DATA-COPY","[0][0]
->successfully
copied data in [tgtschema.pgbench_tellers]
")
(1 row)
```

```
tgtdb=# SELECT edb_util.process_status_from_log(clone_rmt_src_tgt);
__OUTPUT__

                                process_status_from_log
-----
-> -----
-
(RUNNING,"28-JUN-17 13:18:06.634364 -04:00",4022,INFO,"STAGE: DATA-COPY","[0][1]
->successfully
copied data in [tgtschema.pgbench_history]
")
(1 row)
```

```
tgtdb=# SELECT edb_util.process_status_from_log(clone_rmt_src_tgt);
__OUTPUT__

                                process_status_from_log
-----
-> -----
-
(RUNNING,"28-JUN-17 13:18:10.550393 -04:00",4039,INFO,"STAGE: POST-DATA","CREATE
->PRIMARY KEY
CONSTRAINT pgbench_tellers_pkey successful"
)
(1 row)
```

```

tgt>=# SELECT edb_util.process_status_from_log(clone_rmt_src_tgt);
__OUTPUT__

                                process_status_from_log
-----
(1 row)

```

次に、クローンされたテーブルを示します。

```

tgt>=# \dt+
__OUTPUT__

                                List of relations
 Schema | Name | Type | Owner | Size | Description
-----+-----+-----+-----+-----+-----
tgtschema | pgbench_accounts | table | enterprisedb | 256 MB | 
tgtschema | pgbench_branches | table | enterprisedb | 8192 bytes | 
tgtschema | pgbench_history | table | enterprisedb | 25 MB | 
tgtschema | pgbench_tellers | table | enterprisedb | 16 kB | 
(4 rows)

```

remotecopyschema ファンクションが呼び出されたときに、4つのバックグラウンドワーカーが指定されました。

ログファイル clone_rmt_src_tgt の次の部分は、4つのバックグラウンドワーカーを使用した並列データのコピー操作のステータスを示しています。

```

Wed Jun 28 13:18:05.232949 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] [0] table count
[4]
Wed Jun 28 13:18:05.233321 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] [0][0] worker
started to
copy data
Wed Jun 28 13:18:05.233640 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] [0][1] worker
started to
copy data
Wed Jun 28 13:18:05.233919 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] [0][2] worker
started to
copy data
Wed Jun 28 13:18:05.234231 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] [0][3] worker
started to
copy data
Wed Jun 28 13:18:05.298174 2017 EDT: [4024] INFO: [STAGE: DATA-COPY] [0][3]
successfully

```

(次のページに続く)

(前のページからの続き)

```

copied data in [tgtschema.pgbench_branches]
Wed Jun 28 13:18:05.299913 2017 EDT: [4021] INFO: [STAGE: DATA-COPY] [0][0]
→successfully
copied data in [tgtschema.pgbench_tellers]
Wed Jun 28 13:18:06.634310 2017 EDT: [4022] INFO: [STAGE: DATA-COPY] [0][1]
→successfully
copied data in [tgtschema.pgbench_history]
Wed Jun 28 13:18:10.477333 2017 EDT: [4023] INFO: [STAGE: DATA-COPY] [0][2]
→successfully
copied data in [tgtschema.pgbench_accounts]
Wed Jun 28 13:18:10.477609 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] [0] all workers
→finished
[4]
Wed Jun 28 13:18:10.477654 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] [0] copy done
→[4] tables
Wed Jun 28 13:18:10.493938 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] successfully
→copied data
into tgtschema

```

DATA-COPY ログメッセージには、2つの大括弧番号が含まれていることに注意してください（たとえば、[0][3]）。

最初の数字はジョブインデックスで、2番目の数字はワーカーインデックスです。ワーカーインデックス値の範囲は、4つのバックグラウンドワーカーの0から3です。

2つのスキーマのクローンジョブが並行して実行されている場合、最初のログファイルにはジョブインデックスとして0があり、2番目のログファイルにはジョブインデックスとして1があります。

8.2.10 remotecopyschema_nb

remotecopyschema_nb ファンクションは、source_fdw 外部サーバー内で指定されたリモートソースデータベースのソーススキーマから、target_fdw 外部サーバー内で指定されたローカルターゲットデータベースのターゲットスキーマに、スキーマとそのデータベースオブジェクトをコピーします。pgAgent に送信されたジョブ。

```

remotecopyschema_nb(
  <source_fdw> TEXT,
  <target_fdw> TEXT,
  <source> TEXT,
  <target> TEXT,
  <log_filename> TEXT
  [, <on_tblspace> BOOLEAN
  [, <verbose_on> BOOLEAN
  [, <copy_acls> BOOLEAN

```

(次のページに続く)

```
[, <worker_count> INTEGER ]]]
)
```

INTEGER 値のジョブ ID は、pgAgent に送信されたジョブの関数によって返されます。関数が失敗した場合、null が返されます。

source_fdw, target_fdw, source, target、および log_filename は必須パラメーターですが、他のすべてのパラメーターはオプションです。

pgAgent ジョブが完了したら、remove_log_file_and_job ファンクションでジョブを削除します。

8.2.11 パラメーター

source_fdw

データベースオブジェクトのクローン元の postgres_fdw 外部データラッパーによって管理される外部サーバーの名前。

target_fdw

データベースオブジェクトのクローン先の postgres_fdw 外部データラッパーによって管理される外部サーバーの名前。

source

データベースオブジェクトのクローン元のスキーマの名前。

target

ソーススキーマからデータベースオブジェクトのクローンを作成するスキーマの名前。

log_filename

関数からの情報が記録されるログファイルの名前。ログファイルは、postgresql.conf ファイルの log_directory 構成パラメーターで指定されたディレクトリの下に作成されます。

on_tblspace

BOOLEAN 値を使用して、データベースオブジェクトを表領域内に作成するかどうかを指定します。false が指定されている場合、TABLESPACE 句はターゲットスキーマに追加されたときに該当する CREATE DDL ステートメントに含まれません。true が指定されている場合、TABLESPACE 句はターゲットスキーマに追加されるときに CREATE DDL ステートメントに含まれます。on_tblspace パラメーターを省略した場合、デフォルト値は false です。

注釈: true が指定され、データベースオブジェクトに `TABLESPACE` 句があるが、その表領域がターゲットデータベースクラスターに存在しない場合、クローン作成機能は失敗します。

verbose_on

ターゲットスキーマでオブジェクトを作成するときに DDL を `log_filename` に出力するかどうかを指定する `BOOLEAN` 値。 `false` が指定されている場合、DDL は出力されません。 `true` が指定されている場合、DDL が出力されます。省略した場合のデフォルト値は `false` です。

`copy_acls`

`BOOLEAN` 値を使用して、ターゲットスキーマでオブジェクトを作成するときにアクセス制御リスト (ACL) を含めるかどうかを指定します。アクセス制御リストは、`GRANT` 特権ステートメントのセットです。 `false` が指定されている場合、アクセス制御リストはターゲットスキーマに含まれません。 `true` が指定されている場合、ターゲットスキーマのアクセス制御リストが含まれます。 `copy_acls` パラメーターを省略した場合、デフォルト値は `false` です。 **注：** `true` が指定され、`GRANT` 権限を持つロールがターゲットデータベースクラスターに存在しない場合、クローン作成機能は失敗します。

`worker_count`

クローンを並列実行するバックグラウンドワーカーの数。省略した場合のデフォルト値は 1 です。

8.2.12 例

`remotecopyschema` の例と同じクローニング操作が実行されますが、ノンブロッキングファンクション `remotecopyschema_nb` を使用します。

次のコマンドは、ターゲットデータベース `tgtdb` で `pgAgent` を起動します。 `pgagent` プログラムファイルは、EDB Postgres Advanced Server インストールディレクトリの `bin` サブディレクトリにあります。

```
[root@localhost bin]# ./pgagent -l 1 -s /tmp/pgagent_tgtdb_log hostaddr=127.0.0.1
→port=5444
user=enterisedb dbname=tgtdb password=password
```

`remotecopyschema_nb` ファンクションは、例に 2 として示されているジョブ ID を返します。

```
tgtdb=# SELECT edb_util.remotecopyschema_nb
(src_server,tgt_server,srcschema,tgtschema,clone_rmt_src_tgt,worker_count => 4);
__OUTPUT__
remotecopyschema_nb
-----
                2
(1 row)
```

ジョブの完了ステータスは次のように表示されます。

```
tgtdb=# SELECT edb_util.process_status_from_log(clone_rmt_src_tgt);
__OUTPUT__
                                     process_status_from_log
-----
→-----
(FINISH,"29-JUN-17 current:16:00.100284 -04:00",3849,INFO,"STAGE: FINAL",
→"successfully clone schema into tgtschema")
```

(次のページに続く)

(前のページからの続き)

(1 row)

以下は、ログファイルと pgAgent ジョブを削除します。

```
tgtdb=# SELECT edb_util.remove_log_file_and_job (clone_rmt_src_tgt,2);
__OUTPUT__
remove_log_file_and_job
-----
t
(1 row)
```

8.2.13 process_status_from_log

process_status_from_log ファンクションは、ログファイルからクローニングファンクションのステータスを提供します。

```
process_status_from_log (
    <log_file> TEXT
)
```

この関数は、ログファイルから次のフィールドを返します。

Field name	Description
<i>status</i>	<i>STARTING</i> 、 <i>RUNNING</i> 、 <i>FINISH</i> 、または <i>FAILED</i> のいずれかを表示します。
<i>execution_time</i>	コマンドが実行された日時。タイムスタンプ形式で表示されます。
<i>pid</i>	クロンスキーマが呼び出されるセッションプロセス ID。
<i>level</i>	<i>INFO</i> 、 <i>ERROR</i> 、または <i>SUCCESSFUL</i> のいずれかを表示します。
<i>stage</i>	<i>STARTUP</i> 、 <i>INITIAL</i> 、 <i>DDL-COLLECTION</i> 、 <i>PRE-DATA</i> 、 <i>DATA-COPY</i> 、 <i>POST-DATA</i> 、または <i>FINAL</i> のいずれかを表示します。
<i>message</i>	各コマンドまたは障害に関する情報。

8.2.14 パラメーター

log_file

クローニング機能が呼び出されたときに指定されたスキーマのクローニングを記録するログファイルの名前。

8.2.15 例

以下は、`process_status_from_log` 関数の使用法を示しています。

```
edb=# SELECT edb_util.process_status_from_log(clone_edb_edbcopy);
__OUTPUT__

              process_status_from_log
-----
(1 row)
```

→ (FINISH, "26-JUN-17 11:57:03.214458 -04:00", 3691, INFO, "STAGE: FINAL", "successfully cloned schema")

8.2.16 remove_log_file_and_job

`remove_log_file_and_job` ファンクションは、スキーマクローニング機能によって作成されたログファイルと非ブロック機能によって作成されたジョブを削除することにより、クリーンアップタスクを実行します。

```
remove_log_file_and_job (
  { <log_file> TEXT |
    <job_id> INTEGER |
    <log_file> TEXT, <job_id> INTEGER
  }
)
```

`remove_log_file_and_job` ファンクションを呼び出すときに、2つのパラメーターのいずれかまたは両方の値を指定できます。

- `log_file` のみが指定されている場合、この関数はログファイルのみを削除します。
- `job_id` のみが指定されている場合、この関数はジョブのみを削除します。
- 両方が指定された場合、この関数はログファイルとジョブのみを削除します。

8.2.17 パラメーター

`log_file`

削除するログファイルの名前。

`job_id`

削除するジョブのジョブ ID。

8.2.18 例

次の例では、ログファイル名を指定してログファイルのみを削除します。

```
edb=# SELECT edb_util.remove_log_file_and_job (clone_edb_edbcopy);
__OUTPUT__
remove_log_file_and_job
-----
t
(1 row)
```

次の例では、指定されたジョブ ID でジョブのみを削除します。

```
edb=# SELECT edb_util.remove_log_file_and_job (3);
__OUTPUT__
remove_log_file_and_job
-----
t
(1 row)
```

次の例では、両方の値を指定して、ログファイルとジョブを削除します。

```
tgtdb=# SELECT edb_util.remove_log_file_and_job (clone_rmt_src_tgt,2);
__OUTPUT__
remove_log_file_and_job
-----
t
(1 row)
```

第9章 Enhanced SQL and other miscellaneous features

EDB Postgres Advanced Server には、拡張された SQL 機能と、さらなる柔軟性と利便性を提供するその他のさまざまな機能が含まれています。この章では、これらの追加のいくつかについて説明します。

9.1 コメント

PostgreSQL COMMENT コマンドでサポートされているオブジェクトへのコメントに加えて、EDB Postgres Advanced Server は追加のオブジェクトタイプに関するコメントをサポートしています。サポートされている完全な構文は次のとおりです。

```
COMMENT ON
{
  AGGREGATE <aggregate_name> ( <aggregate_signature> ) |
  CAST (<source_type> AS <target_type>) |
  COLLATION <object_name> |
  COLUMN <relation_name>.<column_name> |
  CONSTRAINT <constraint_name> ON <table_name> |
  CONSTRAINT <constraint_name> ON DOMAIN <domain_name> |
  CONVERSION <object_name> |
  DATABASE <object_name> |
  DOMAIN <object_name> |
  EXTENSION <object_name> |
  EVENT TRIGGER <object_name> |
  FOREIGN DATA WRAPPER <object_name> |
  FOREIGN TABLE <object_name> |
  FUNCTION <func_name> ([[<argmode>] [<argname>] <argtype> [,...]])|
  INDEX <object_name> |
  LARGE OBJECT <large_object_oid> |
  MATERIALIZED VIEW <object_name> |
  OPERATOR <operator_name> (left_type, right_type) |
  OPERATOR CLASS <object_name> USING <index_method> |
  OPERATOR FAMILY <object_name> USING <index_method> |
  PACKAGE <object_name>
  POLICY <policy_name> ON <table_name> |
  [ PROCEDURAL ] LANGUAGE <object_name> |
  PROCEDURE <proc_name> [[[<argmode>] [<argname>] <argtype> [, ...]]]
```

(次のページに続く)

```

PUBLIC SYNONYM <object_name>
ROLE <object_name> |
RULE <rule_name> ON <table_name> |
SCHEMA <object_name> |
SEQUENCE <object_name> |
SERVER <object_name> |
TABLE <object_name> |
TABLESPACE <object_name> |
TEXT SEARCH CONFIGURATION <object_name> |
TEXT SEARCH DICTIONARY <object_name> |
TEXT SEARCH PARSER <object_name> |
TEXT SEARCH TEMPLATE <object_name> |
TRANSFORM FOR <type_name> LANGUAGE <lang_name> |
TRIGGER <trigger_name> ON <table_name> |
TYPE <object_name> |
VIEW <object_name>
} IS <text>

```

aggregate_signature は次のとおりです。

```

* |
[ <argmode> ] [ <argname> ] <argtype> [ , ... ] |
[ [ <argmode> ] [ <argname> ] <argtype> [ , ... ] ]
ORDER BY [ <argmode> ] [ <argname> ] <argtype> [ , ... ]

```

9.1.1 パラメーター

object_name

コメントしているオブジェクトの名前。

AGGREGATE aggregate_name (aggregate_signature)

AGGREGATE 句を含めて、集計に関するコメントを作成します。aggregate_name は集計の名前を指定し、aggregate_signature は関連する署名を次のいずれかの形式で指定します。

```

* |
[ <argmode> ] [ <argname> ] <argtype> [ , ... ] |
[ [ <argmode> ] [ <argname> ] <argtype> [ , ... ] ]
ORDER BY [ <argmode> ] [ <argname> ] <argtype> [ , ... ]

```

argmode がファンクション、プロシージャ、または集計引数のモードである場合、argmode は IN、OUT、INOUT、または VARIADIC です。省略した場合、デフォルトは IN です。

argname は集計引数の名前です。

argtype は、集計引数のデータ型です。

CAST (source_type AS target_type)

キャストに関するコメントを作成するには、**CAST** 句を含めます。キャストに関するコメントを作成する場合、**source_type** はキャストのソースデータ型を指定し、**target_type** はキャストのターゲットデータ型を指定します。

COLUMN relation_name.column_name

COLUMN 句を含めて、列に関するコメントを作成します。**column_name** は、コメントを適用する列の名前を指定します。**relation_name** は、列が存在するテーブル、ビュー、複合型、または外部テーブルです。

CONSTRAINT constraint_name ON table_name

CONSTRAINT constraint_name ON DOMAIN domain_name

CONSTRAINT 句を含めて、制約に関するコメントを追加します。制約に関するコメントを作成する場合、**constraint_name** は制約の名前を指定します。**table_name** または **domain_name** は、制約が定義されているテーブルまたはドメインの名前を指定します。

FUNCTION func_name ([[argmode] [argname] argtype [, ...]])

関数に関するコメントを追加するには、**FUNCTION** 句を含めます。**func_name** は、関数の名前を指定します。**argmode** は、関数のモードを指定します。**argmode** は、**IN**、**OUT**、**INOUT**、または **VARIADIC** です。省略した場合、デフォルトは **IN** です。

argname は、ファンクション、プロシージャ、または集計引数の名前を指定します。**argtype** は、ファンクション、プロシージャ、または集計引数のデータ型を指定します。

large_object_oid

large_object_oid は、コメントしているラージオブジェクトのシステム割り当ての **OID** です。

OPERATOR operator_name (left_type, right_type)

OPERATOR 句を含めて、演算子に関するコメントを追加します。**operator_name** は、コメントする演算子の（オプションでスキーマ修飾）名前を指定します。**left_type** および **right_type** は、演算子の引数の（オプションでスキーマ修飾）データ型です。

OPERATOR CLASS object_name USING index_method

OPERATOR CLASS 句を含めて、オペレータークラスに関するコメントを追加します。**object_name** は、コメントする演算子の（オプションでスキーマ修飾）名前を指定します。**index_method** は、オペレータークラスの関連するインデックスメソッドを指定します。

OPERATOR FAMILY object_name USING index_method

OPERATOR FAMILY 句を含めて、演算子ファミリーに関するコメントを追加します。**object_name** は、コメントする演算子ファミリーの（オプションでスキーマ修飾）名前を指定します。**index_method** は、演算子ファミリーの関連するインデックス方法を指定します。

POLICY policy_name ON table_name

ポリシーに関するコメントを追加するには、**POLICY** 句を含めます。**policy_name** はポリシーの名前を指定し、**table_name** はポリシーが関連付けられるテーブルを指定します。

PROCEDURE proc_name ([[argmode] [argname] argtype [, ...]])

プロシージャに関するコメントを追加するには、PROCEDURE 句を含めます。proc_name は、プロシージャの名前を指定します。argmode は、プロシージャのモードを指定します。argmode は、IN、OUT、INOUT、または VARIADIC です。省略した場合、デフォルトは IN です。

argname は、ファンクション、プロシージャ、または集計引数の名前を指定します。argtype は、ファンクション、プロシージャ、または集計引数のデータ型を指定します。

RULE rule_name ON table_name

RULE 句を含めて、ルールに COMMENT を指定します。rule_name はルールの名前を指定し、table_name はルールが定義されているテーブルの名前を指定します。

TRANSFORM FOR type_name LANGUAGE lang_name

TRANSFORM に COMMENT を指定するには、TRANSFORM FOR 句を含めます。

type_name はトランスフォームのデータ型の名前を指定し、lang_name はトランスフォームの言語の名前を指定します。

TRIGGER trigger_name ON table_name

TRIGGER 句を含めて、トリガーに COMMENT を指定します。trigger_name はトリガーの名前を指定し、table_name はトリガーが定義されているテーブルの名前を指定します。

text

文字列リテラルまたはコメントを削除する NULL として書き込まれるコメント。

9.1.2 ノート

テーブル、集計、照合、変換、ドメイン、外部テーブル、ファンクション、インデックス、演算子、演算子クラス、演算子ファミリー、パッケージ、プロシージャ、シーケンス、テキスト検索オブジェクト、タイプ、およびビューの名前はスキーマ修飾できます。

9.1.3 例

次の例では、new_emp という名前のテーブルにコメントを追加します。

```
COMMENT ON TABLE new_emp IS This table contains information about new employees.;
```

COMMENT コマンドの使用の詳細については、次の PostgreSQL コアドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/sql-comment.html>

9.2 関数 `version()` の出力

`version()` ファンクションのテキスト文字列出力には、製品の名前、そのバージョン、およびインストールされているホストシステムが表示されます。

EDB Postgres Advanced Server の場合、`version()` 出力は、最初のテキストワードが EDB Postgres Advanced Server バージョン 10 以前の *EnterpriseDB* ではなく *PostgreSQL* であるという点で、PostgreSQL コミュニティバージョンと同様の形式です。

`version()` 出力の一般的な形式は次のとおりです。

```
PostgreSQL $PG_VERSION_EXT (EnterpriseDB EDB Postgres Advanced Server $PG_VERSION) on
→$host
```

したがって、現在の EDB Postgres Advanced Server のバージョン文字列は次のように表示されます。

```
edb@45032=#select version();
__OUTPUT__
version
-----
→-----
-----

PostgreSQL 14.0 (EnterpriseDB EDB Postgres Advanced Server 14.0.0) on x86_64-pc-linux-
→gnu, compiled by gcc
(GCC) 4.8.5 20150623 (Red Hat 4.8.5-11), 64-bit
(1 row)
```

対照的に、EDB Postgres Advanced Server 10 の場合、バージョン文字列は次のとおりです。

```
edb=# select version();
__OUTPUT__
version
-----
→-----
-----

EnterpriseDB 10.4.9 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.4.7 20120313 (Red
→Hat
4.4.7-18), 64-bit
(1 row)
```

9.3 スタンバイ時のロジカルデコード

スタンバイサーバーで論理デコードを使用すると、`get`、`peek`、`advance` などの API オペレーションに回答できる論理レプリケーションスロットをスタンバイサーバーに作成できます。

LOGICAL DECODING の詳細については、[PostgreSQL core documentation](#) を参照してください。

スタンバイサーバーの論理スロットが機能するには、スタンバイで `hot_standby_feedback` パラメーターを `ON` に設定する必要があります。`hot_standby_feedback` パラメーターは、スタンバイサーバー上の既存の論理レプリケーションスロットが必要とする最近無効になった行を `VACUUM` が削除しないようにします。スタンバイでスロットの競合が発生した場合、スロットはドロップされます。

スタンバイで論理デコードを機能させるには、プライマリサーバーとスタンバイサーバーの両方で `wal_level` を `logical` に設定する必要があります。`wal_level` が `logical` 以外の値に設定されている場合、スロットは作成されません。プライマリで `wal_level` を `logical` 以外の値に設定し、スタンバイに既存の論理スロットがある場合、そのようなスロットはドロップされ、新しいスロットを作成できません。

トランザクションがプライマリサーバーに書き込まれると、アクティビティによってスタンバイサーバーに論理スロットが作成されます。プライマリサーバーがアイドル状態の場合、スタンバイサーバーに論理スロットを作成するにはかなり時間がかかる場合があります。

レプリケーションと論理デコードの例をサポートする関数の詳細については、次の場所にある PostgreSQL ドキュメントを参照してください。

<https://www.postgresql.org/docs/current/functions-admin.html#FUNCTIONS-REPLICATION>

<https://www.postgresql.org/docs/current/logicaldecoding-example.html>

第10章 System catalog tables

次のシステムカタログテーブルには、データベースオブジェクトの定義が含まれています。システムテーブルのレイアウトは変更される場合があります。システムテーブルに格納されている情報に依存するアプリケーションを作成する場合は、既存のカタログビューを使用するか、カタログビューを作成してアプリケーションを変更することをお勧めします。システムテーブル。

10.1 edb_dir

`edb_dir` テーブルには、`CREATE DIRECTORY` コマンドで作成されたディレクトリを指すエイリアスごとに1つの行が含まれています。ディレクトリは、ホストファイルシステムへの制限付きアクセスをユーザーに許可するパス名のエイリアスです。

ディレクトリを使用して、ファイルシステム内の特定のディレクトリツリーにユーザーをフェンスできます。たとえば、`UTL_FILE` パッケージは、ユーザーがホストファイルシステム内のファイルとディレクトリを読み書きできるようにする機能を提供しますが、データベース管理者が `CREATE DIRECTORY` コマンドを介してアクセスを許可したパスへのアクセスのみを許可します。

Column	Type	Modifiers	Description
<i>dirname</i>	<i>"name"</i>	<i>not null</i>	エイリアスの名前。
<i>diowner</i>	<i>oid</i>	<i>not null</i>	エイリアスを所有するユーザーのOID。
<i>dirpath</i>	<i>text</i>		アクセスが許可されるディレクトリ名。
<i>diracl</i>	<i>aclitem[]</i>		エイリアスにアクセスできるユーザーを決定するアクセス制御リスト。

10.2 edb_all_resource_groups

`edb_all_resource_groups` テーブルには、`CREATE RESOURCE GROUP` コマンドで作成されたリソースグループごとに1つの行が含まれ、各リソースグループのアクティブなプロセスの数を表示します。

Column	Type	Modifiers	Description
<i>group_name</i>	<i>"name"</i>		リソース グループの名前。

次のページに続く

表 2 – 前のページからの続き

Column	Type	Modifiers	Description
<i>active_processes</i>	<i>integer</i>		リソースグループ内の現在アクティブなプロセスの数。
<i>cpu_rate_limit</i>	<i>float8</i>		リソースグループの最大 CPU レート制限。`0`は無制限を意味します。
<i>per_process_cpu_rate_limit</i>	<i>float8</i>		リソースグループ内の現在アクティブなプロセスごとの最大 CPU レート制限。
<i>dirty_rate_limit</i>	<i>float8</i>		リソースグループの最大ダーティレート制限。`0`は無制限を意味します。
<i>per_process_dirty_rate_limit</i>	<i>float8</i>		リソースグループ内の現在アクティブなプロセスごとの最大ダーティレート制限。

10.3 edb_policy

edb_policy 表には、ポリシーごとに 1 つの行が含まれています。

Column	Type	Modifiers	Description
<i>policyname</i>	<i>name</i>	<i>not null</i>	ポリシー名。
<i>policygroup</i>	<i>oid</i>	<i>not null</i>	現在未使用です。
<i>policyobject</i>	<i>oid</i>	<i>not null</i>	このポリシーによって保護されたテーブルの OID (<i>object_schema</i> と <i>object_name</i>)。
<i>policykind</i>	<i>char</i>	<i>not null</i>	このポリシーによって保護されるオブジェクトの種類：テーブルの場合 'r' ビューの場合 =シノニム 現在は常に「r」です。
<i>policyproc</i>	<i>oid</i>	<i>not null</i>	ポリシー機能の OID (<i>function_schema</i> と <i>policy_function</i>)
<i>policyinsert</i>	<i>boolean</i>	<i>not null</i>	ポリシーが `INSERT` ステートメントによって適用される場合は true。
<i>polycyselect</i>	<i>boolean</i>	<i>not null</i>	ポリシーが `SELECT` ステートメントによって適用される場合は true。
<i>policydelete</i>	<i>boolean</i>	<i>not null</i>	ポリシーが `DELETE` ステートメントによって適用される場合は true。

次のページに続く

表 3 – 前のページからの続き

Column	Type	Modifiers	Description
<i>policyupdate</i>	<i>boolean</i>	<i>not null</i>	ポリシーが`UPDATE`ステートメントによって適用される場合は <i>true</i> 。
<i>policyindex</i>	<i>boolean</i>	<i>not null</i>	現在未使用です。
<i>policyenabled</i>	<i>boolean</i>	<i>not null</i>	ポリシーが有効になっている場合は <i>true</i> 。
<i>policyupdatecheck</i>	<i>boolean</i>	<i>not null</i>	`UPDATE`ステートメントによって更新された行がポリシーを満たす必要がある場合は <i>true</i> 。
<i>policystatic</i>	<i>boolean</i>	<i>not null</i>	現在未使用です。
<i>policytype</i>	<i>integer</i>	<i>not null</i>	現在未使用です。
<i>policyopts</i>	<i>integer</i>	<i>not null</i>	現在未使用です。
<i>policyseccols</i>	<i>int2vector</i>	<i>not null</i>	`sec_relevant_cols`にリストされている列の列番号。

10.4 edb_profile

`edb_profile` テーブルには、使用可能なプロファイルに関する情報が保存されます。`edb_profiles` は、クラスター内のすべてのデータベースで共有されます。

Column	Type	References	Description
<i>oid</i>	<i>oid</i>		行識別子（非表示の属性、明示的に選択する必要があります）。
<i>prfname</i>	<i>name</i>		プロファイルの名前。
<i>prffailedloginattempts</i>	<i>integer</i>		プロファイルで許可されるログイン試行の失敗回数。-1 は、デフォルトのプロファイルの値を使用することを示します。-2 は、ログインの失敗に制限がないことを示します。
<i>prfpasswordlocktime</i>	<i>integer</i>		プロファイルに関連付けられたパスワードロック時間（秒）。-1 は、デフォルトのプロファイルの値を使用することを示します。-2 は、アカウントを永久にロックすることを示します。

次のページに続く

表 4 – 前のページからの続き

Column	Type	References	Description
<i>prfpasswordlifetime</i>	<i>integer</i>		プロファイルに関連付けられたパスワードの有効期間 (秒)。-1 は、デフォルトのプロファイルの値を使用することを示します。-2 は、パスワードが無期限であることを示します。
<i>prfpasswordgracetime</i>	<i>integer</i>		プロファイルに関連付けられたパスワードの猶予時間 (秒)。-1 は、デフォルトのプロファイルの値を使用することを示します。-2 は、パスワードが無期限であることを示します。
<i>prfpasswordreusetime</i>	<i>integer</i>		ユーザーがパスワードを再利用するまでに待機する必要がある秒数。-1 は、デフォルトのプロファイルの値を使用することを示します。-2 は、古いパスワードを再利用できないことを示します。
<i>prfpasswordreusemax</i>	<i>integer</i>		パスワードを再利用するまでに発生する必要があるパスワードの変更の回数。-1 は、デフォルトのプロファイルの値を使用することを示します。-2 は、古いパスワードを再利用できないことを示します。
<i>prfpasswordverifyfuncdb</i>	<i>oid</i>	<i>pg_database.oid</i>	パスワード検証機能が存在するデータベースの OID。
<i>prfpasswordverifyfunc</i>	<i>oid</i>	<i>pg_proc.oid</i>	プロファイルに関連付けられたパスワード検証機能の OID。

10.5 edb_redaction_column

カタログ `edb_redaction_column` には、テーブルの列に添付されたデータリダクションポリシーの情報が格納されます。

Column	Type	References	Description
<i>oid</i>	<i>oid</i>		行識別子 (非表示の属性、明示的に選択する必要があります)。
<i>rdpolicyid</i>	<i>oid</i>	<i>edb_redaction_policy.oid</i>	データリダクションポリシーは、説明されている列に適用されます。

次のページに続く

表 5 – 前のページからの続き

Column	Type	References	Description
<i>rdrelid</i>	<i>oid</i>	<i>pg_class.oid</i>	説明された列が属するテーブル。
<i>rdattnum</i>	<i>int2</i>	<i>pg_attribute.attnum</i>	記述された列の番号。
<i>rdscope</i>	<i>int2</i>		リダクションスコープ: 1 = query、 2 = top_tlist、4 = top_tlist_or_error
<i>rdexception</i>	<i>int2</i>		リダクション例外: 8 = none、16 = equal、32 = リークプルーフ
<i>rdfuncexpr</i>	<i>pg_node_tree</i>		データリダクション関数式

注釈: 表のリダクションポリシー *edb_redaction_column.rdpolicyid* が有効であり、リダクションポリシー式 *edb_redaction_policy.rdexpr* が true と評価される場合、Description 列はリダクションされます。

10.6 edb_redaction_policy

カタログ *edb_redaction_policy* には、テーブルのリダクションポリシーの情報が格納されます。

Column	Type	References	Description
<i>oid</i>	<i>oid</i>		行識別子（非表示の属性、明示的に選択する必要があります）。
<i>rdname</i>	<i>name</i>		データリダクションポリシーの名前
<i>rdrelid</i>	<i>oid</i>	<i>pg_class.oid</i>	データリダクションポリシーが適用されるテーブル。
<i>rdenable</i>	<i>boolean</i>		データリダクションポリシーは有効になっていますか？
<i>rdexpr</i>	<i>pg_node_tree</i>		データリダクションポリシー式。

注釈: データリダクションポリシーは、テーブルが有効になっていて、式が true と評価された場合に適用されます。

10.7 edb_resource_group

edb_resource_group テーブルには、CREATE RESOURCE GROUP コマンドで作成されたリソースグループごとに 1 つの行が含まれています。

Column	Type	Modifiers	Description
<i>rgrpname</i>	<i>"name"</i>	<i>not null</i>	リソース グループの名前。
<i>rgrpcpuratelimit</i>	<i>float8</i>	<i>not null</i>	リソースグループの最大 CPU レート制限。`0`は無制限を意味します。
<i>rgrpdirtyratelimit</i>	<i>float8</i>	<i>not null</i>	リソースグループの最大ダーティ レート制限。`0`は無制限を意味します。

10.8 edb_variable

`edb_variable` テーブルには、パッケージレベルの変数（パッケージ内で宣言された各変数）ごとに1つの行が含まれています。

Column	Type	Modifiers	Description
<i>varname</i>	<i>"name"</i>	<i>not null</i>	変数の名前。
<i>varpackage</i>	<i>oid</i>	<i>not null</i>	パッケージを保存する `pg_namespace` 行の OID。
<i>vartype</i>	<i>oid</i>	<i>not null</i>	変数の型を定義する `pg_type` 行の OID。
<i>varaccess</i>	<i>"char"</i>	<i>not null</i>	+ 変数がパッケージの外部で参照できる場合。 - 変数がパッケージ内でのみ参照できる場合。 注：パブリック変数はパッケージヘッダー内で宣言されます変数はパッケージ本体内で宣言されます。
<i>varsrc</i>	<i>text</i>		変数のデフォルト値式を含む、変数宣言のソースが含まれています。
<i>varseq</i>	<i>smallint</i>	<i>not null</i>	パッケージで変数が宣言された順序。

10.9 pg_synonym

`pg_synonym` テーブルには、`CREATE SYNONYM` コマンドまたは `CREATE PUBLIC SYNONYM` コマンドで作成された同義語ごとに1つの行が含まれています。

Column	Type	Modifiers	Description
<i>synname</i>	<i>"name"</i>	<i>not null</i>	シノニムの名前。

次のページに続く

表 9 – 前のページからの続き

Column	Type	Modifiers	Description
<i>synnamespace</i>	<i>oid</i>	<i>not null</i>	`synowner`を置き換えます。 シノニムが格納されている `pg_namespace`行の OID が含ま れています
<i>synowner</i>	<i>oid</i>	<i>not null</i>	シノニムを所有するユーザーの OID。
<i>synobjschema</i>	<i>"name"</i>	<i>not null</i>	参照されるオブジェクトが定義さ れているスキーマ。
<i>synobjname</i>	<i>"name"</i>	<i>not null</i>	参照されるオブジェクトの名前。
<i>synlink</i>	<i>text</i>		参照されるオブジェクトが定義 されているデータベースリンクの (オプション) 名前。

10.10 product_component_version

`product_component_version` テーブルには機能の互換性に関する情報が含まれています。アプリケーションはインストール時または実行時にこのテーブルを照会して、アプリケーションが使用する機能をこのデプロイメントで利用できることを確認できます。

Column	Type	Description
<i>product</i>	<i>character varying (74)</i>	製品の名前。
<i>version</i>	<i>character varying (74)</i>	製品のバージョン番号。
<i>status</i>	<i>character varying (74)</i>	リリースのステータス。

第11章 EDB Postgres Advanced Server exceptions

次の表に、事前定義された例外、SQLstate 値、関連する redwood エラーコード、および例外の説明を示します。

Exception name	SQLState	Redwood error code	Description
<i>value_error</i>	22000	-6502	文字列から数値への変換に失敗した場合に例外が発生します。
<i>invalid_number</i>	22000	-6502	文字列から数値への変換が失敗すると、PL ステートメントで例外が発生します。
<i>date-time_value_out_of_range</i>	22008	-1863	有効な範囲外の日付形式でフィールドを書き込んでいるときに例外が発生します。
<i>divide_by_zero</i>	22012	-1476	この例外は、プログラムが数値をゼロで除算しようとしたときに発生します。
<i>zero_divide</i>	22012	-1476	この例外は、プログラムが数値をゼロで除算しようとしたときに発生します。
<i>dup_val_on_index</i>	23505	-1	この例外は、プログラムが一意のインデックスによって制限された列に重複する値を保存しようとするとき発生します。
<i>invalid_cursor</i>	34000	-1001	この例外は、プログラムが無効なカーソルでカーソル操作 (オープンされていないカーソルをクローズするなど) を試行すると発生します。
<i>cursor_already_open</i>	42P03	-6511	この例外は、プログラムが既に開いているカーソルを開こうとすると発生します。
<i>collection_is_null</i>	P1403	-6531	この例外は、プログラムが初期化されていないネストした表または VARRAY の要素に値を割り当てようとするとき発生します。

次のページに続く

表 1 – 前のページからの続き

Exception name	SQLState	Redwood error code	Description
<i>subscript_beyond_count</i>	P1404	-6533	この例外は、プログラムがコレクション内の要素数より大きい索引番号を使用してネストした表または VARRAY を参照しようとするが発生します。
<i>subscript_outside_limit</i>	P1405	-6532	この例外は、プログラムが範囲外の索引番号を使用してネストした表または VARRAY 要素を参照しようとするが発生します。

DBMS_CRYPT	
<i>package</i>	
<i>ciphersuiteinvalid</i>	The exception occurs when the cipher suite is not defined.
<i>ciphersuitenull</i>	The exception occurs when no value is specified for the cipher suite or contains a <i>NULL</i> value.
<i>keybadsize</i>	The exception occurs when the specified key size is bad.
<i>keynull</i>	The exception occurs when the key is not specified.
UTL_FILE package	
<i>invalid_filehandle</i>	The exception occurs when file handle is invalid.
<i>invalid_maxlinesize</i>	The exception occurs when the max line size is invalid or the max line size value is not within the range.
<i>invalid_mode</i>	The exception occurs when the <i>open_mode</i> parameter in <i>FOPEN</i> is invalid.
<i>invalid_operation</i>	The exception occurs when the file could not be opened or used upon request.
<i>invalid_path</i>	The exception occurs when the file location or filename is invalid.
<i>read_error</i>	The exception occurs when the operating system error occurred during the read operation.
<i>write_error</i>	The exception occurs when the operating system error occurred during the write operation.
UTL_HTTP package	
<i>end_of_body</i>	The exception occurs when the end of HTTP response body is reached.
UTL_URL package	
<i>bad_fixed_width_charset</i>	The exception occurs when the fixed-width multibyte character is not allowed as a URL character set.
<i>bad_url</i>	The exception occurs when the URL includes badly formed escape code sequences.

11.1 EDB Postgres Advanced Server のキーワード

キーワードとは、EDB Postgres Advanced Server パーサーによって特別な意味または関連があると認識される単語です。pg_get_keywords() ファンクションを使用して、EDB Postgres Advanced Server キーワードの最新リストを取得できます。

```
acctg=#
acctg=# SELECT * FROM pg_get_keywords();
__OUTPUT__
      word      | catcode |      catdesc
-----+-----+-----
  abort         | U      | unreserved
  absolute      | U      | unreserved
  access        | U      | unreserved
  ...
```

pg_get_keywords は、EDB Postgres Advanced Server によって認識されるキーワードを含むテーブルを返します。

- word 列にはキーワードが表示されます。
- catcode 列は、カテゴリコードを表示します。
- catdesc 列には、キーワードが属するカテゴリの簡単な説明が表示されます。

名前が二重引用符で囲まれている場合、識別子には任意の文字を使用できることに注意してください。pg_get_keywords() ファンクションを選択的に照会して、特定のカテゴリに属する EDB Postgres Advanced Server キーワードの最新リストを取得できます。

```
SELECT * FROM pg_get_keywords() WHERE catcode = code;
```

code は次のとおりです。

R - この単語は予約されています。予約済みキーワードは識別子として使用できません。これらはサーバーが使用するために予約されています。

U - 単語は予約されていません。非予約語は一部のコンテキストで内部的に使用されますが、データベースオブジェクトの名前として使用される場合があります。

T - この単語は内部で使用されますが、関数または型の名前として使用できます。

C - この単語は内部で使用されます。関数または型の名前としては使用できません。

EDB Postgres Advanced Server の識別子とキーワードの詳細については、次の PostgreSQL コアドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/sql-syntax-lexical.html>