



EDB

EDB Postgres Advanced Server Guide

リリース *14.1.0*

EnterpriseDB

2021 年 12 月 16 日

目次

第 1 章 What's New	3
1.1 制限	4
1.1.1 Conventions Used in this Guide	5
1.1.2 About the Examples Used in this Guide	6
第 2 章 Enhanced Compatibility Features	19
2.1 互換性機能を有効にする	19
2.2 ストアドプロシージャ言語	19
2.3 オプティマイザのヒント	20
2.4 データディクショナリ Views	20
2.5 dblink_ora	20
2.6 プロファイル管理	21
2.7 組み込みパッケージ	21
2.8 クライアントライブラリを開く	23
2.9 ユーティリティ	23
2.10 ECPGPlus	26
2.11 テーブルのパーティション分割	26
第 3 章 Database Administration	27
3.1 Configuration Parameters	27
3.1.1 Setting Configuration Parameters	27
3.1.2 Summary of Configuration Parameters	30
3.1.3 Configuration Parameters by Functionality	64
3.2 Index Advisor	109
3.2.1 Index Advisor Components	110
3.2.2 Index Advisor Configuration	112
3.2.3 Using Index Advisor	113
3.2.4 Reviewing the Index Advisor Recommendations	117
3.2.5 Index Advisor Limitations	121
3.3 SQL Profiler	122
3.4 pgsnmpd	124
3.4.1 pgsnmpd の構成	124
3.4.2 リスナーアドレスの設定	125
3.4.3 pgsnmpd の呼び出し	125
3.4.4 pgsnmpd ヘルプの表示	126
3.4.5 pgsnmpd からの情報の要求	126
3.5 EDB Audit Logging	127
3.5.1 Audit Logging Configuration Parameters	127
3.5.2 Selecting SQL Statements to Audit	130
3.5.3 Enabling Audit Logging	145

3.5.4	Audit Log File	151
3.5.5	Using Error Codes to Filter Audit Logs	157
3.5.6	Using Command Tags to Filter Audit Logs	158
3.5.7	Redacting Passwords in Audit Logs	160
3.5.8	Audit Log Archiving	161
3.5.9	Object Auditing	164
3.6	Unicode Collation Algorithm	170
3.6.1	Unicode 照合アルゴリズムの基本概念	171
3.6.2	Unicode の国際コンポーネント	171
3.6.3	照合の使用	174
第 4 章	EDB Resource Manager	179
4.1	リソースグループの作成と管理	180
4.1.1	リソースグループの作成	180
4.1.2	リソースグループの変更	181
4.1.3	ドロップリソースグループ	182
4.1.4	リソースグループへのプロセスの割り当て	183
4.1.5	リソースグループからプロセスを削除する	184
4.1.6	リソースグループ内のプロセスの監視	184
4.2	CPU 使用率の調整	186
4.2.1	リソースグループの CPU レート制限の設定	186
4.2.2	例-単一グループ内の単一プロセス	187
4.2.3	例-単一グループ内の複数のプロセス	188
4.2.4	例-複数のグループの複数のプロセス	190
4.3	ダーティバッファスロットリング	193
4.3.1	リソースグループのダーティレート制限の設定	193
4.3.2	例-単一グループ内の単一プロセス	194
4.3.3	例-単一グループ内の複数のプロセス	197
4.3.4	例-複数のグループの複数のプロセス	199
4.4	システムカタログ	202
4.4.1	edb_all_resource_groups	202
4.4.2	edb_resource_group	202
第 5 章	libpq C Library	203
5.1	EDB SPL での libpq の使用	203
5.2	REFCURSOR サポート	203
5.3	配列のバインド	209
5.3.1	PQBulkStart	209
5.3.2	PQexecBulk	210
5.3.3	PQBulkFinish	210
5.3.4	PQexecBulkPrepared	210
5.3.5	サンプルコード (PQBulkStart、PQexecBulk、PQBulkFinish を使用)	211
5.3.6	サンプルコード (PQexecBulkPrepared を使用)	212
第 6 章	Debugger	213
6.1	デバッガーの構成	214
6.2	デバッガーの起動	214
6.3	デバッガーウィンドウ	216

6.4	メインデバッガーウィンドウ	217
6.4.1	プログラム本体パネル	218
6.4.2	タブパネル	218
6.4.3	スタックタブ	219
6.5	プログラムのデバッグ	222
6.5.1	コードのステップ実行	223
6.5.2	ブレークポイントを使用する	224
6.5.3	インコンテキストデバッグ用のグローバルブレークポイントの設定	225
6.5.4	デバッガーの終了	228
第 7 章	Performance Analysis and Tuning	231
7.1	ダイナチューン	231
7.1.1	edb_dynatune	231
7.1.2	edb_dynatune_profile	232
7.2	EDB 待機状態	232
7.2.1	edb_wait_states_data	234
7.2.2	edb_wait_states_queries	236
7.2.3	edb_wait_states_sessions	238
7.2.4	edb_wait_states_samples	239
7.2.5	edb_wait_states_purge	241
第 8 章	EDB Clone Schema	243
8.1	設定プロセス	245
8.1.1	拡張機能と PL / Perl のインストール	245
8.1.2	設定パラメータの設定	246
8.1.3	EDB クロンスキーマのインストール	248
8.1.4	外部サーバーとユーザーマッピングの作成	248
8.2	EDB クロンスキーマ Functions	253
8.2.1	localcopyschema	253
8.2.2	localcopyschema_nb	257
8.2.3	remotecopyschema	260
8.2.4	remotecopyschema_nb	264
8.2.5	process_status_from_log	267
8.2.6	remove_log_file_and_job	268
第 9 章	Enhanced SQL and Other Miscellaneous Features	271
9.1	コメント	271
9.2	ファンクションバージョンの出力 ()	274
9.3	スタンバイでの論理デコード	275
第 10 章	System Catalog Tables	277
10.1	edb_dir	277
10.2	edb_all_resource_groups	277
10.3	edb_profile	279
10.4	edb_redaction_column	280
10.5	edb_redaction_policy	280
10.6	edb_resource_group	281
10.7	edb_variable	281

10.8	pg_synonym	282
10.9	product_component_version	282
第 11 章	Advanced Server Exceptions	285
11.1	Advanced Server キーワード	286

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、強化された 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、パフォーマンスのモニタリングと分析、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、Advanced Server 専用の機能について説明しています。

- **互換性機能の強化。**このセクションでは、Advanced Server がサポートする互換性機能の概要を説明します。
- **データベースアドミニストレーション。**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決定するのに役立ちます。
- **SQL Profiler** は、アプリケーションで実行が不十分な SQL クエリを見つけて診断します。
- **pgsnmpd** は、Advanced Server の現在の状態に関する階層モニタリング情報を返す SNMP エージェントです。
- **** EDB Resource Manager.** ******このセクションには、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保守する方法を示します。
- **CPU Usage Throttling** は、Advanced Server プロセスによる CPU 使用率を制御するメソッドを提供します。
- **Dirty Buffer Throttling** は、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 library** は、Advanced Server 用の C アプリケーションプログラミングインタフェース (API) 言語です。
- **** PL デバッガ.** **** PL Debugger** は、PL / pgSQL 用のグラフィカル指向のデバッグツール。
- **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、アプリケーションおよびデータベースサーバのパフォーマンスを分析および改善す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タイプに応じて、Advanced Server を迅速かつ簡単に設定する手段を提供します。
- **EDB wait states** は、パフォーマンス診断のために待機イベントおよび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 機能およびその他の柔軟性と利便性を提供す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、Advanced Server 固有のデータベースオブジェクト用に追加されたシステムカタログテーブルが追加されています。
- **** Advanced Server Exceptions.** ******このセクションには、事前定義の例外、SQLstate 値、関連するレッドウッドエラーコード、および例外の説明に関する情報が含まれています。

- **** Advanced Server Keywords.** ******このセクションには、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。

Advanced Server と PostgreSQL で共有される機能については、次の URL で入手可能な PostgreSQL コア文書を参照してください。

< <https://www.postgresql.org/docs/current/> インデックス.html >

第1章 What's New

次の機能が EDB Postgres Advanced Server 13 で変更され、Advanced Server 14 が作成されました。

- Advanced Advanced Server は、INSERT 文で被参照されるテーブル、ビュー、マテリアライズドビュー、またはサブクエリの別名を作成する AS 句をサポートするようになりました。詳細については、「[Oracle Developers SQL ガイドのデータベース互換性](#)」を参照してください。
- Advanced Advanced Server は、SET ステートメントを監査するために、`edb_audit_statement` パラメータを指定した SET コマンドをサポートするようになりました。詳細については、[EDB Audit Logging](#) を参照してください。
- Advanced Advanced Server は、DBMS_SQL ファンクションで NUMBER 型の値を受け入れるサポートを追加しました。詳細については、* [Oracle Developers Built-in Package Guide のデータベース互換性](#)*を参照してください。
- Advanced Advanced Server は、テーブルスペース名前が指定されていない場合、デフォルトデータベーステーブルスペース PG_DEFAULT を含めるサポートを追加しました。詳細については、* [Oracle Developers Catalog Views ガイドのデータベース互換性](#)*を参照してください。
- Advanced Advanced Server は、RAW データ値を読み書きするために、UTL_FILE および PUT_RAW API を UTL_FILE パッケージに追加しました。詳細については、* [Oracle Developers Built-in Package Guide のデータベース互換性](#)*を参照してください。
- Advanced Advanced Server は、PARTITION `part_name` または SUBPARTITION `subpart_name` を使用して PARTITION または SUBPARTITION にアクセスのサポートを追加しました。詳細については、[データベース互換性テーブルパーティショニングガイド](#)を参照してください。
- Advanced Advanced Server は、現在のセッションに関する情報を取得する USERENV および SYS_CONTEXT 関数をサポートするようになりました。詳細については、* [Oracle Developers Reference Database のデータベース互換性](#)*を参照してください。
- Advanced Advanced Server は、ビット単位のファンクション BITAND と BITOR、ビット単位の集約ファンクション BIT_AND_AGG、および BIT_OR_AGG をサポートするようになりました。これらは、ビット単位の AND または OR オペレーションを実行し、入力引数のデータタイプに基づいて値を返します。詳細については、* [Oracle Developers Reference Database のデータベース互換性](#)*を参照してください。
- Advanced Advanced Server は、パラメータ `default_with_rowids = true` を設定することにより、一意の ROWID でテーブルパーティションを作成するサポートを追加しました。詳細については、[データベース互換性テーブルパーティショニングガイド](#)を参照してください。Advanced Server は* [Audit Log Archiving](#) *を導入します。これにより、データベースは監査ログファイルをアーカイブできます。詳細については、[EDB Audit Logging](#) を参照してください。
- Advanced Advanced Server は、SUBPARTITION TEMPLATE のサポートを導入し、リスト、レンジ、ま

たはハッシュサブパーティションを使用して、各テーブルパーティションのデフォルトのサブパーティションを作成します。詳細については、[データベース互換性テーブルパーティショニングガイド](#)を参照してください。

- Advanced Advanced Server は、(@) または (-) パラメータで (@) または (-) を指定することにより、SELECT、UPDATE、DELETE、INSERT などの特定の DML ステートメントのオブジェクトを選択的に監査できる**オブジェクト監査**をサポートするようになりました。詳細については、[Object Auditing](#)を参照してください。
- Advanced Advanced Server は、CONNECT BY 句に拡張機能を追加しました。これは、階層の親レコードとその子レコード間のリレーションを指定します。詳細については、* オラクル Developers Guide *のデータベース互換性を参照してください。
- Advanced Advanced Server は、SELECT、INSERT、UPDATE、または DELETE ステートメントで PARTITION または SUBPARTITION にアクセスのための ALIAS のサポートを追加しました。詳細については、[データベース互換性テーブルパーティショニングガイド](#)を参照してください。Advanced Server は、オブジェクトで DDL オペレーションが実行されるときに作成および最終 DDL 時間を保存するサポートを追加しました。詳細については、* オラクル Developers Catalog Views ガイドのデータベース互換性*を参照してください。

1.1 制限

次の表は、PostgreSQL のさまざまなハード制限を示しています。ただし、絶対的なハード制限に達する前に、パフォーマンス制限や使用可能なディスクスペースなどの実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	NA
		NA
<i>number of databases</i>	4,294,950,911	NA
		NA
<i>relations per database</i>	1,431,650,303	NA
		NA
<i>relation size</i>	32 TB	デフォルトの BLCKSZ は 8192 バイトです
<i>rows per table</i>	limited by the number of tuples that can fit onto 4,294,967,295 pages	NA
		NA

次のページに続く

表 1 – 前のページからの続き

Item	Upper Limit	Comment
<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; 以下のノートを参照してください
<i>field size</i>	1 GB	NA
		NA
<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
		NA

!!! Note * -格納されるタプルが単一の 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はさらに削減されます。例、タプルヘッダーを除くと、1600 の int 列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が、1600 の 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 などのタイプの可変長フィールドは、値が必要ラージさの場合、テーブルの TOAST テーブルに行外に格納することができます。テーブルのヒープ内のタプル内に残るのは 18 バイトのポインタのみです。より短い長さフィールドの場合、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最大列制限に影響します。さらに、新しく作成されたタプルのドロップされた列の値は、タプルのヌルビットマップで null として内部的にマークされ null が、ヌルビットマップはスペースも占有します。

1.1.1 Conventions Used in this Guide

</div>

以下はこのドキュメント中で使用されるコンベンションのリストです。

- このガイドは、Linux システムと Windows システムの両方に適用されます。ディレクトリ経路は、スラッシュ付きの Linux フォーマットで表示されます。Windows システムで作業する場合は、ドライブ文字の後にコロンを付けてディレクトリパスをスタートし、スラッシュを円記号に置き換えます。
- このドキュメントの情報の一部は、PostgreSQL および EDB Postgres Advanced Server データベースシステムに互換的に適用される場合があります。* 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 Advanced Server の両方を総称する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ニーズがある場合は、特定の名前、PostgreSQL または Advanced Server が使用されます。PostgreSQL または Advanced Server 製品のインストールディレクトリパスは、PGRES_INSTALL_HOME * と呼ばれます。PostgreSQL Linux インストールの場合、これはバージョン 10 以前の/opt/PostgreSQL/x.x に

デフォルト設定されます。それ以降のバージョンでは、PostgreSQL コミュニティパッケージソフトを使用します。

- バージョン 10 以前のインタラクティブインストーラを使用して行われる Advanced Server Linux インストールの場合、これはデフォルトで `/opt/edb/asx.x` になります。
- RPM パッケージを使用して Advanced Server Linux をインストールする場合、デフォルトは `/usr/edb/asxx` になります。
- Advanced 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。製品のバージョン番号は、バージョン 10 以降では `x.x` または `xx` で表されます。

1.1.2 About the Examples Used in this Guide

</div>

このガイドの例は、以下に示すタイプと背景で示されています。

例と例からの出力は、黒の背景に固定幅の白いフォントで表示されます。

この例では、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。

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

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

ここで、`xx` は Advanced Server のバージョン番号です。

さらに、オラクル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を含むスクリプトが同じディレクトリにあります。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。
- テーブルに対するすべての権限を `PUBLIC` グループに付与します。

テーブルとプログラムは、現在のユーザがテーブルとプロシージャを作成するパーミッションを持つ検索パスの最初のスキーマに作成されます。次のコマンドを発行して、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、組織内の従業員を表します。従業員、部門、および従業員の履歴レコードの3つのタイプ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、雇用日、給与、およびマネージャがあります。一部の従業員は、給与に加えて手数料を受け取ります。すべての従業員関連情報は、`emp` テーブルに保存されます。

サンプル会社は地域的に多様であるため、部門の場所を追跡します。各会社の従業員は部門に割り当てられます。各部門は、一意の部門番号と短縮名前で識別されます。各部門は1つの場所に関連付けられています。すべての部門関連情報は `dept` テーブルに保存されます。

同社はまた、従業員が抱える仕事に関する情報も追跡しています。一部の従業員は、長期間にわたって会社に勤務しており、さまざまな位置に就いたり、昇給を受けたり、部門を切り替えたりしました。スタート日、新しいジョブ、部門、給与、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` テーブルで維持されます。

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
-- 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),
```

(次のページに続く)

(前のページからの続き)

```

*   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)
*       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');
```

(次のページに続く)

(前のページからの続き)

```

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);
--
-- 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');

```

(次のページに続く)

(前のページからの続き)

```
--
-- 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 $$
DECLARE
    *   v_empno      NUMERIC(4);
    *   v_ename      VARCHAR(10);
    *   emp_cur CURSOR FOR
    *       SELECT empno, ename FROM emp ORDER BY empno;
BEGIN
    *   OPEN emp_cur;
    *   RAISE INFO 'EMPNO ENAME';
    *   RAISE INFO '-----';
    *   LOOP
    *       FETCH emp_cur INTO v_empno, v_ename;
    *       EXIT WHEN NOT FOUND;
    *       RAISE INFO '% %', v_empno, v_ename;
    *   END LOOP;
    *   CLOSE emp_cur;
    *   RETURN;
END;
$$ 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);
```

(次のページに続く)

(前のページからの続き)

```

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;
$$ 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'.
--

```

(次のページに続く)

(前のページからの続き)

```

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
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

```

(次のページに続く)

(前のページからの続き)

```

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

```

(次のページに続く)

(前のページからの続き)

```

*   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;
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);

```

(次のページに続く)

(前のページからの続き)

```

*   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;
*       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,

```

(次のページに続く)

(前のページからの続き)

```

*           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 $$
DECLARE
*   v_action          VARCHAR(24);
*   v_text            TEXT;
BEGIN
*   IF TG_OP = 'INSERT' THEN
*       v_action := ' added employee(s) on ';
*   ELSIF TG_OP = 'UPDATE' THEN
*       v_action := ' updated employee(s) on ';
*   ELSIF TG_OP = 'DELETE' THEN
*       v_action := ' deleted employee(s) on ';
*   END IF;
*   v_text := 'User ' || USER || v_action || CURRENT_DATE;
*   RAISE INFO ' %', v_text;
*   RETURN NULL;
END;
$$ 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

```

(次のページに続く)

(前のページからの続き)

```
*      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

Advanced Server には、オラクル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。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、* オラクル Developers Guide のデータベース互換性*を参照してください。バージョン固有のガイドは次から入手できます。

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

次のセクションでは、Advanced Server でサポートされている互換性機能の一部を取り上げます。

2.1 互換性機能を有効にする

Advanced Server をインストールするいくつかの方法があり、互換性機能を利用できます。

- クラスターを初期化する前に、INITDBOPTS 変数（Advanced Server サービス構成ファイル）を使用して--redwood-like を指定します。
- `initdb` を呼び出してオラクル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、オラクル非互換モードで `cluser` を初期化します。

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、EDB ウェブサイトから入手できる* EDB Postgres Advanced Server インストールガイド*を参照してください。

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

2.2 ストアドプロシージャ言語

Advanced Server は、カスタムプロシージャ、関数、トリガー、パッケージソフトを作成できる生産性の高いプロシージャ言語をサポートしています。プロシージャ言語：

SQL 言語とビルトインパッケージソフトを補完します。 - 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。

ストアドプロシージャ言語の使用方法については、次の場所にあるデータベース互換性ストアドプロシージャ言語ガイドを参照してください。

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

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プランを生成します。これらの実行計画を分析した後、サーバーは、最短で結果セットを（一般に）結果計画を選択します。サーバーの計画の選択は、いくつかの要因に依存します。

- データハンドリング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 コマンドによって収集された列統計。

ルールとして、クエリープランナは最も安価なプランを選択します。クエリプランを選択するときに、オプティマイザヒントを使用してサーバーに影響を与えることができます。

オプティマイザのヒントは、DELETE、INSERT、SELECT、または UPDATE コマンドの直後に続くコメントのような構文に埋め込みれたディレクティブ（またはマルチブルの指示子）です。コメント内のキーワードは、結果セットを作成するときに特定の計画を採用または回避するようにサーバーに指示します。オプティマイザのヒントの使用方法については、次の場所にある「オラクル Developers Guide のデータベース互換性ガイド」を参照してください。

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

2.4 データディクショナリ Views

Advanced Server には、オラクル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。Advanced Server で使用可能なビューの詳細については、次の場所にある『オラクル Developers Catalog Views のデータベース互換性ガイド』を参照してください。

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

2.5 dblink_oracle

dblink_oracle は、Advanced Server 内からオラクルシステムに保存された SELECT、INSERT、UPDATE または DELETE データを可能にする OCI ベースのデータベースリンクを提供します。dblink_oracle の使用方法、およびサポートされている関数と手順の詳細については、次の Web サイトで入手可能な「オラクル開発者向けデータベース互換性ガイド」を参照してください。

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

2.6 プロファイル管理

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。プロファイル管理コマンドを使用すると、データベーススーパーユーザが名前付けプロファイル*を作成および管理できます。各プロファイルは、パスワードおよび 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。プロファイル内のルールは次のことができます。

- 失敗したログイン試行をカウントする
- ログイン試行の過剰な失敗により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
- 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に新しい要件を適用できます。

プロファイルを作成した後、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、各クラスターにはマルチプルのプロファイルが存在する場合があります。マルチプルのデータベースにアクセスできる 1 人のユーザは、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、次の場所にある「[Oracle Developers Guide のデータベース互換性ガイド](#)」を参照してください。

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

2.7 組み込みパッケージ

Advanced Server は、Oracle の手順および機能との互換性を提供する多数のビルトインパッケージソフトをサポートしています。

Package Name	Description
DBMS_ALERT	DBMS_ALERT パッケージは、アラートの登録、送信、および受信の機能を提供します。
DBMS_AQ	DBMS_AQ パッケージは、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、高度なキューイング機能のサポート手順を提供します。

次のページに続く

表 1 – 前のページからの続き

Package Name	Description
DBMS_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関数とプロシージャを提供します。DBMS_CRYPTO 関数を使用して、暗号的に強い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、スケジューリング、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	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関係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、メッセージバッファからメッセージを取得したりする機能を提供します。
DBMS_PIPE	DBMS_PIPE パッケージは、同じデータベースクラスタ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ためのいくつかのメソッドを提供します。
DBMS_REDACT	DBMS_REDACT パッケージは、クエリーによって返されるデータの編集またはマスキングを有効にします。
DBMS_RLS	DBMS_RLS パッケージを使用すると、特定の 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	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からの読み取りおよびファイルへの書き込み機能を提供します。
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、生のデータタイプの長さを操作または取得できます。

次のページに続く

表 1 – 前のページからの続き

Package Name	Description
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。
	NA

各パッケージ内で使用可能な手順と機能の詳細については、次の場所にある『オラクル Developers ビルトインパッケージガイドのデータベース互換性*』を参照してください。

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

2.8 クライアントライブラリを開く

Open Client Library は、オラクル Call Interface とのアプリケーションの相互運用性を提供します。以前は「ロックイン」されていたアプリケーションは、アプリケーションコードをほとんどまたはまったく変更せずに、Advanced Server またはオラクルデータベースと連携できます。Open Client Library の EDB 実装は C で記述されています。

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

Open Client Library でサポートされている機能の詳細については、次から入手できる* EDB Postgres Advanced Server OCL Connector Guide *を参照してください。

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

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、次の場所にある『オラクル Developers Tools and Utilities Guide のデータベース互換性*』を参照してください。

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

**** EDB * Plus ****

EDB * Plus は、オラクル開発者とユーザーになじみのある Advanced Server へのコマンドラインユーザインタフェー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のクエリ
- ストアドプロシージャの実行 SQL コマンドからの出力のフォーマット
- バッチスクリプトの実行
- OS コマンドの実行

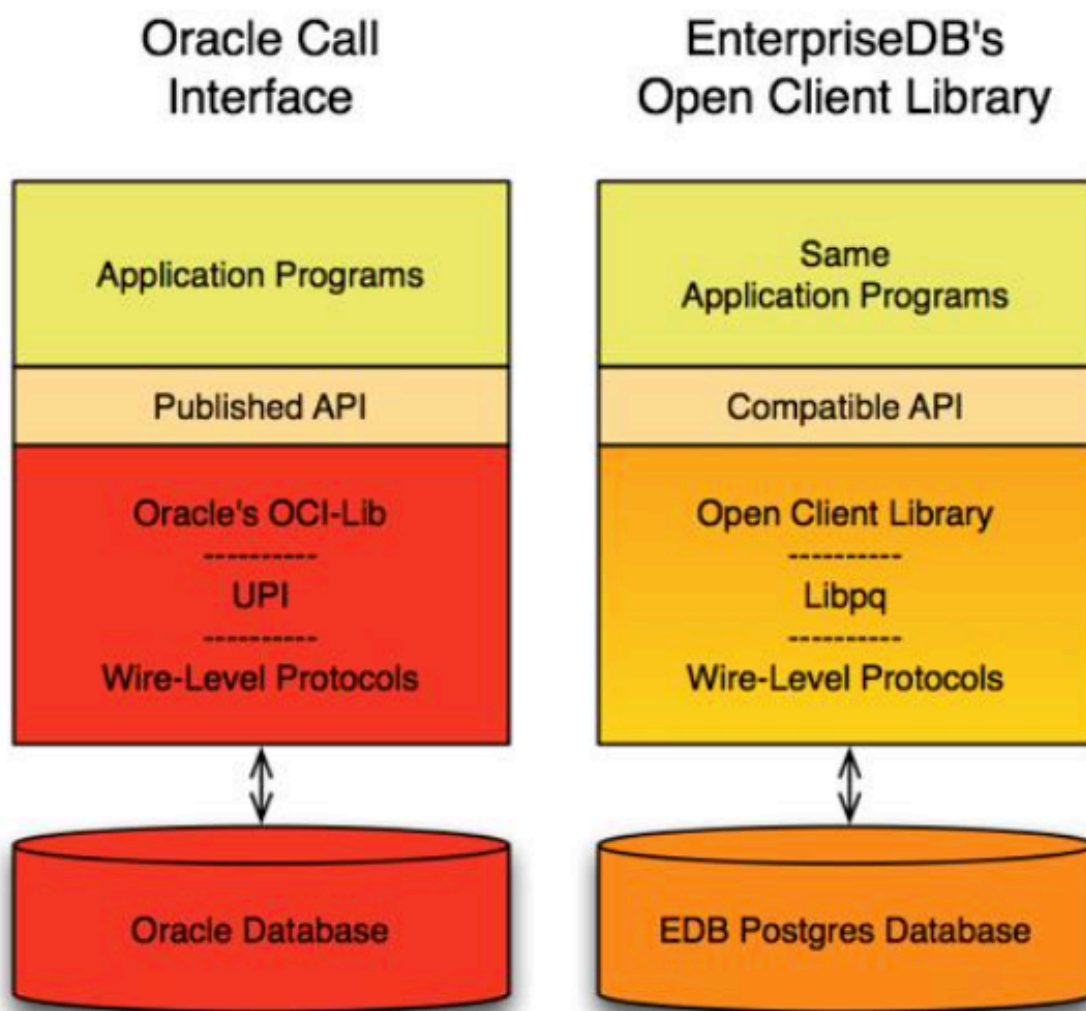


図 1: The Open Client Library

- レコード出力

EDB * Plus の詳細については、次から入手できる* EDB * Plus ユーザーガイド*を参照してください。

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

** EDB * Loader **

EDB * Loader は、Advanced Server 用のオラクルデータベースと互換性のあるインタフェースを提供する高性能バルクデータローダです。EDB * Loader コマンドラインユーティリティは、オラクル SQL * Loader が提供するパラメータ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。

オラクル SQL * Loader データローディング方法のサポート-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用のオラクル SQL * Loader 互換構文 - 区切り文字で区切られたフィールドまたは固定幅のフィールドを持つ入力データ- 拒否されたレコードを収集するための不良ファイル - マルチプルのターゲットテーブルの読み込み - 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ファイルを破棄- 標準入力からのデータローディングとリモートローディング

** 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（関数、ストアドプロシージャ、トリガー、およびパッケージソフト）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 ソースコード（プレーンテキスト）を含むファイルを、読み取りがほとんど不可能なフォームの同じコードを含むファイルに変換します。コードの難読化されたフォームを取得したら、そのコードを Advanced Server に送信すると、それらのプログラムが難読化されたフォームで保存されます。EDB * Wrap はコードを不明瞭にしますが、テーブル定義はまだ公開されています。

ラップするものはすべて難読化されたフォームで保存されます。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関数とプロシージャが難読化されたフォームで保存されます。

** Dynamic Runtime Instrumentation Tools Architecture (DRITA) **

Dynamic Runtime Instrumentation Tools Architecture (DRITA) を使用すると、DBA はカタログビューをクエリーして、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する**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* snapshots *を比較して、システムのパフォーマンスを評価します。snapshot は、特定の時点で保存されたシステムパフォーマンスデータのセットです。各 snapshot は一意の ID 番号で識別されます。DRITA レポート機能で snapshotID 番号を使用して、システムパフォーマンス統計を結果することができます。

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 コードは構文チェックされ、C に変換されます。

ECPGPlus は、Advanced Server データベースに接続されている場合、C プログラムで Pro * C 構文をサポートします。ECPGPlus は以下をサポートします：

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

ECPGPlus の使用については、EDB ウェブサイトから入手できる * EDB Postgres Advanced Server ECPG コネクタガイド*を参照してください。

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

2.11 テーブルのパーティション分割

パーティション化されたテーブルでは、論理的にラージ 1 つのテーブルが小さな物理的断片に分割されます。パーティション化にはいくつかの利点があります。

- クエリのパフォーマンスは、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に劇的に改善できます。パーティション化により、インデックスの前からパーティション列を省略できるため、インデックスサイズが小さくなり、インデックスの頻繁に使用される部分がメモリに収まる可能性が高くなります。
- クエリーまたは更新が単一のパーティションのラージ部分にアクセスする場合、インデックスおよびテーブル全体に散在するランダムアクセス読み取りを使用する代わりに、サーバーが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。
- パーティショニングのデザイン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、一括オペレーションよりもはるかに高速です。また、バルク DELETE に起因する VACUUM のオーバーヘッドを完全に回避します。
- あまり使用されないデータは、安価な（または低速の）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ニングは、テーブルが非常にラージ場合にのみ価値があります。テーブルがパーティショニング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。ルールとして、テーブルのサイズはデータベースサーバの物理的メモリを超える必要があります。

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、次の Web サイトで入手可能な * Database Compatibility Table Partitioning Guide * を参照してください。

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

第3章 Database Administration

</ div>

この章では、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。

3.1 Configuration Parameters

</ div>

このセクションでは、Advanced Server のデータベースサーバ構成パラメータについて説明します。これらのパラメータは、データファイルとログファイルの場所、接続、認証、セキュリティ設定、リソースのアロケーションと消費、アーカイビングとレプリケーションの設定、エラーロギングと統計の収集、最適化とパフォーマンスチューニングなど、データベースサーバの動作と環境のさまざまな側面を制御します。ロケールおよびフォーマット設定など。

Advanced Server にのみ適用される構成パラメータは、[Summary of Configuration Parameters](#) に記載されています。

構成パラメータに関する追加情報は、次の場所にある PostgreSQL Core Documentation にあります。

< <https://www.postgresql.org/docs/current/> 静的/runtime-config.html >

3.1.1 Setting Configuration Parameters

</ div>

</ div>

このセクションでは、構成パラメータの指定および設定方法の概要を説明します。

各設定パラメータは、**name/value** ペアを使用して設定されます。パラメータ名は大文字小文字ません。通常、パラメータ名前は、オプションの等号 (=) によって値から分離されます。

以下は、`postgresql.conf` ファイルのいくつかの設定パラメータ設定の例です。

```
# This is a comment
log_connections = yes
log_destination = 'syslog'
search_path = '$user', public'
shared_buffers = 128MB
```

パラメータ値は、5つのタイプのいずれかとして指定されます。

- **ブール値。**許容値は、on, off, true, false, yes, no, 1, 0、またはこれらの明確なプレフィックスとして記述できます。
- **整数。**小数部分のない数字。
- **浮動小数点。**10 進数で区切られたオプショナルの小数部分を持つ数値。
- **** String. ****テキスト値。値がシンプル識別子または数値ではない場合（つまり、値にスペースやその他の句読点などの特殊文字が含まれる場合）、シングルクォートで囲みます。
- **** Enum. ****文字列値の特定のセット。許可される値は、システムビュー `pg_settings.enumvals` にあります。列挙値は大文字小文字ません。

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、暗黙的な単位（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）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。別のユニット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。

設定パラメータの設定は、さまざまな方法で確立できます。

Advanced Server データベース製品のビルド時に確立されるパラメータ設定がいくつかあります。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。また、データベースの作成時に各データベースに永続的に設定されるパラメーターがいくつかあります。これらのパラメーターも読み取り専用であり、その後データベースで変更することはできません。

- データベースクラスタ全体のほとんど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スタートまたはリスタート時に有効になります。これらの初期パラメータ設定の一部は、以下の箇条書きで説明されているように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ビルトインのデフォルト設定があります。
- 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、`postgresql.conf` ファイルの構成パラメーターはオーバーライドされます。`ALTER SYSTEM` コマンドは、`postgresql.auto.conf` ファイル内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、構成ファイルでパラメーター設定を変更できます。その後、構成ファイルが再ロードされると（`SIGHUP` シグナルが発行されることを意味します）、特定のパラメータタイプに対して、変更されたパラメーター設定が直ちに有効になります。これらのパラメータタイプの一部では、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。これらの他のパラメータタイプの場合、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。さらに、他のパラメータタイプの場合、変更された設定は、データベースサーバが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 コードコメントの次のセクションを参照してください。

– <https://www.postgresql.org/docs/current/config-setting.html>

SQL コマンド `ALTER DATABASE`、`ALTER ROLE`、または `ALTER ROLE IN DATABASE` を使用して、特定のパラメータ設定を変更できます。変更されたパラメータ設定は、コマンドの実行後、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している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。これらの SQL コマンドによって確立されたパラメーター設定は、データベースサーバーの再起動後も無期限に有効であり、2 番目と 3 番目の箇条書きで説明した方法によって確立された設定を上書きします。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、次の方法でのみ変更できます。

- a) 異なるパラメータ値でこれらのコマンドを再発行する、または
- b) 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、前の箇条書きで説明した方法で確立された設定を使用するようにパラメータを変更します。これらの SQL コマンドの正確な構文については、* PostgreSQL Core Documentation * の第 VI 章「リファレンス」の「SQL コマンド」セクションを参照してください。
- <https://www.postgresql.org/docs/current/sql-commands.html>
- PGOPTIONS 環境変数を使用するか、EDB-PSQL または PSQL コマンドラインターミナルプログラム内で SET コマンドを使用して、個々のセッションの継続中に特定のパラメータ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、セッションの期間のみ、2 番目、3 番目、および 4 番目の箇条書きで説明されている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。

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 Advanced Server インストールの data ディレクトリにあります。

ポンド記号 (#) 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。パラメータを設定または変更した後、新しいパラメータ値を有効にするには、サーバーを reload または restart にする必要があります。

postgresql.conf ファイル内の一部のパラメーターには、change requires restart を示すコメントが含まれています。サーバーのリスタートが必要なパラメーターのリストをビューするには、psql コマンドラインで次のクエリーを実行します。

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

</div>

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、接続クライアントの認証時にサーバーが使用する認証メソッドを指定します。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、initdb は、そのクラスターの pg_hba.conf ファイルを作成し、クライアントの接続に必要な認証の種類を指定します。pg_hba.conf ファイルを変更するには、選択したエディタでファイルを開きます。pg_hba.conf ファイルの認証設定を変更した後、サーバーをリスタート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、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/ 静的/auth-pg-hba-conf.html](https://www.postgresql.org/docs/current/静的/auth-pg-hba-conf.html) >

接続要求を受信すると、サーバーは、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。`pg_hba.conf` ファイルエントリをログに記録してサーバーへの接続を認証するには、`postgresql.conf` ファイルで `log_connections` パラメータを ON に設定します。

レコードは、接続タイプ、データベース名前、ユーザ名前、クライアント IP アドレス、および `pg_hba.conf` ファイル内のこれらのパラメーターとマッチングしたときに接続を許可する認証メソッドを指定します。サーバーへの接続が承認されると、一致した行番号と `pg_hba.conf` ファイルの認証レコードを確認できます。

次の例は、認証成功時の有効な `pg_hba.conf` エントリー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

</div>

このセクションには、すべての Advanced Server 構成パラメーターとパラメーターのいくつかのキー属性をリストしたサマリテーブルが含まれています。

これらの属性は、サマリテーブルの次の列で説明されています。

- **** Parameter.** ****** 構成パラメータ名前。
- **効果の範囲。** 設定パラメータ設定の効果の範囲。
 - **-Cluster** – 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の作成時に設定されます。ローカ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変更できます。
 - つまり、次のエンティティに対して異なる設定を行うことができます。これにより、このリストの后者の設定が前のものを上書きします。a) データベースクラスタ全体、b) データベースデータベースクラスタ内の特定のデータベース、c) 特定のロール、d) 特定のロール特定のデータベースに接続、e) 特定のセッション。
- **有効になる時期** 変更されたパラメータ設定が有効になる時期。
 - **-Preset** – Advanced Server 製品の構築時または特定のデータベースの作成時に確立されます。これは読み取り専用パラメータであり、変更できません。**-Restart** – データベースサーバーを再起動する必要があります。**-Reload** – 構成ファイルを再ロードする必要があります（またはデー

データベースサーバを再起動できます)。-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の設定を変更すると、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すると、新しいセッションで有効になります。

- **** Authorized User.** ******パラメータ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タイプ。
 - EPAS service account – EDB Postgres Advanced Server サービスアカウント（オラクル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 –スーパーユーザ特権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（ALTER コマンドで変更するデータベースまたはロール）に対する権限を持つデータベースロール。-n/a –パラメーター設定はどのユーザも変更できません。
- **説明。** 設定パラメータの簡単な説明。
- **** EPAS のみ.** ****** ‘X’ –構成パラメータは EDB Postgres Advanced Server のみに適用されます。この列のエントリは、設定パラメータが PostgreSQL にも適用されることを示しています。

!!! Note * 変更しないパラメーターがいくつかあります。これらは、[説明] 列で「**注：内部使用のみ**」として指定されています。

次の表に、構成パラメーターの要約を示します。

Parameter	Scope of Effect	When Takes Effect	Authorized User	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	統計および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に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	'archive_command' を使用して WAL ファイルのアーカイビングを許可します。	
<i>archive_timeout</i>	Cluster	Reload	EPAS service account	新しいファイルが N 秒以内に開始されなかった場合、次の xlog ファイルへのスイッチを強制します。	

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 User	Description	EPAS Only
<i>array_nulls</i>	Session	Immediate	User	配列内の 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ための最大許容時間を設定します	
<i>autovacuum</i>	Cluster	Reload	EPAS service account	オートバキュームサブプロセスを開始します。	
<i>autovacuum_analyze_scale_factor</i>	Cluster	Reload	EPAS service account	`reltuples`の一部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 の周回を防ぐためにテーブルをオートバキュームする年齢。	
<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プロセスの最大数を設定します。	
<i>autovacuum_multixact_freeze_max_age</i>	Cluster	Restart	EPAS service account	multixact 周回を防ぐためにテーブルを自動バキュームするオートバキューム年齢。	
<i>autovacuum_naptime</i>	Cluster	Reload	EPAS service account	オートバキューム実行の間にスリープする時間。	
<i>autovacuum_vacuum_cost_delay</i>	Cluster	Reload	EPAS service account	オートバキューム真空の場合、ミリ秒単位の真空コスト遅延。	

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 User	Description	EPAS Only
<i>autovacuum_vacuum_cost_limit</i>	Cluster	Reload	EPAS service account	オートバキューム真空の場合、昼寝前に利用可能な真空コスト額。	
<i>autovacuum_vacuum_scale_factor</i>	Cluster	Reload	EPAS service account	`reltuples`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	各バキュームワークプロセスで使用され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	ラウンド間のバックグラウンドライタのスリープ時間。	
<i>bgwriter_lru_maxpages</i>	Cluster	Reload	EPAS service account	バックグラウンドライタのラウンドごとにフラッシュするLRUページの最大数。	
<i>bgwriter_lru_multiplier</i>	Cluster	Reload	EPAS service account	ラウンドごとにフリーする平均バッファ使用量の倍数。	
<i>block_size</i>	Cluster	Preset	n/a	ディスクブロックのサイズを表示します。	
<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。	

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 User	Description	EPAS Only
<i>bonjour_name</i>	Cluster	Restart	EPAS service account	Bonjour サービス名前を設定します。	
<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確認します。	
<i>checkpoint_completion_target</i>	Cluster	Reload	EPAS service account	チェックポイントインターバルの一部として、チェックポイント中にダーティバッファのフラッシュに費やされた時間。	
<i>checkpoint_segments</i>	Deprecated	Deprecated	Deprecated	パラメータに値を指定すると、サーバーが起動しなくなります。	
<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頻繁に入力された場合の警告を有効にします。	
<i>client_encoding</i>	Session	Immediate	User	クライアントのキャラクタ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。	

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 User	Description	EPAS Only
<i>commit_delay</i>	Session	Immediate	Superuser	トランザクションコミットから WAL をディスクにフラッシュするまでの遅延をマイクロ秒単位で設定します。	
<i>commit_siblings</i>	Session	Immediate	User	`commit_delay` を実行する前の最小同時オープ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	プランナがコンストレイン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ーの見積もりを設定します。	
<i>cpu_operator_cost</i>	Session	Immediate	User	各演算子またはファンクション呼び出しの処理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。	

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 User	Description	EPAS Only
<i>cursor_tuple_fraction</i>	Session	Immediate	User	取得されるカーソルの行の割合のプランナーの見積もりを設定します。	
<i>custom_variable_classes</i>	Cluster	Reload	EPAS service account	Advanced Server 9.2 では非推奨です。	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になっている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フォーマットを設定します。	
<i>db_dialect</i>	Session	Immediate	User	ビルトインの名前付けの優先順位を設定します。	X
<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	データベースユーザ名を有効にします。	
<i>deadlock_timeout</i>	Session	Immediate	Superuser	デッドロックロックをチェックする前にロックを待機する時間を設定します。	

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 User	Description	EPAS Only
<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	インデントはツリー表示を解析および計画ツリーします。	
<i>debug_print_parse</i>	Session	Immediate	User	各クエリの構文解析ツリーをログに記録します。	
<i>debug_print_plan</i>	Session	Immediate	User	各クエリの実行計画をログに記録します。	
<i>debug_print_rewritten</i>	Session	Immediate	User	各クエリを書き直された構文解析ツリーをログに記録します。	
<i>default_heap_fillfactor</i>	Session	Immediate	User	デフォルトでは、このヒープフィルファクターを使用して新しいテーブルを作成しデフォルト。	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	でテーブルとインデックスを作成するデフォルトテーブルスペースを設定します。	
<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。	

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 User	Description	EPAS Only
<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 <i>de-ferrable</i> なステータスを設定します。	
<i>default_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 をサポートする新しいテーブル (インデックス付きの ROWID) を作成しデフォルト。	X
<i>dynamic_library_path</i>	Session	Immediate	Superuser	動的にロード可能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	使用する動的共有メモリの実装を選択します。	
<i>edb_audit</i>	Cluster	Reload	EPAS service account	EDB 監査を有効にして、XML または CSV フォーマットの監査レポートを作成します。	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

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 User	Description	EPAS Only
<code>edb_audit_archiver_timeout</code>	Cluster	Reload	EPAS service account	<code>`edb_audit_archiver_timeout`</code> パラメータに基づいて監査ログファイルをチェックします。	X
<code>edb_audit_archiver_filename_prefix</code>	Cluster	Reload	EPAS service account	<code>`edb_audit_directory`</code> に <code>`edb_audit_archiver_filename_prefix`</code> が含まれる監査ログファイルが圧縮または有効期限、あるいはその両方に適格であるかどうかを判断します。パラメータは <code>`edb_audit_filename`</code> と整列する必要があります。	X
<code>edb_audit_archiver_compress_timeout</code>	Cluster	Reload	EPAS service account	監査ログが圧縮の対象となるまでの時間を秒単位で示します。	X
<code>edb_audit_archiver_compress_size_limit</code>	Cluster	Reload	EPAS service account	監査ログが圧縮の対象となるまでの合計サイズをメガバイト単位で表示します。	X
<code>edb_audit_archiver_compress_command</code>	Cluster	Reload	EPAS service account	監査ログファイルを圧縮するコマンド。	X
<code>edb_audit_archiver_compress_strategy</code>	Cluster	Reload	EPAS service account	すでに圧縮されたログファイルのサフィックスを決定します; パラメータは <code>`edb_audit_archiver_compress_command`</code> と整列する必要があります。	X
<code>edb_audit_archiver_expire_timeout</code>	Cluster	Reload	EPAS service account	監査ログが有効期限が切れるまでの時間を秒単位で示します。	X
<code>edb_audit_archiver_expire_size_limit</code>	Cluster	Reload	EPAS service account	合計サイズをメガバイト単位で表示します。その後、監査ログは有効期限が切れます。	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 User	Description	EPAS Only
<i>edb_audit_archiver_expire_command</i>	Cluster	Reload	EPAS service account	期限切れの監査ログを削除する前に実行コマンド。	X
<i>edb_audit_archiver_sort_file</i>	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`mtime`に基づいて並べ替えます。	X
<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`edb_audit_directory`または`syslog`を設定します。`syslog`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前パターンを設定します。	X
<i>edb_audit_rotation_day</i>	Cluster	Reload	EPAS service account	曜日に基づく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

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 User	Description	EPAS Only
<i>edb_connectby_order</i>	Session	Immediate	User	`ORDER BY`を使用しない`CONNECT BY`クエリの結果を深さ 1 次にオーダー替えます。 注：内部使用のみ。	X
<i>edb_custom_plan_tries</i>	Session	Immediate	User	プランナが汎用実行計画を選択する前に、プランナ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	データの編集を有効にします。	X
<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	プランナがテーブルパーティション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 パラメーターをハンドリングする方法を決定します。	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 User	Description	EPAS Only
<i>edb_redwood_raw_names</i>	Session	Immediate	User	Redwood インターフェイスから PostgreSQL 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場合、`NULL` を空の文字列として扱います。	X
<i>edb_resource_group</i>	Session	Immediate	User	現在のプロセスで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
<i>edb_sql_protect.level</i>	Cluster	Reload	EPAS service account	イベントが見つかったときの SQL / Protect の動作を定義します。	X
<i>edb_sql_protect.max_protected_queries</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ヨンの最大数を設定します。	X
<i>edb_sql_protect.max_protected_roles</i>	Cluster	Restart	EPAS service account	SQL/ Protect で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_to_cache</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サポートを要求する代わりに、エラーを続行できます。	X
<i>edb_wait_states.directory</i>	Cluster	Restart	EPAS service account	EDB 待機状態ログはこのディレクトリに保存されます。	X
<i>edb_wait_states.retention_period</i>	Cluster	Reload	EPAS service account	EDB 待機状態のログファイルは、保存期間後に自動的に削除されます。	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 User	Description	EPAS Only
<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インターバル。	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	ディスク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	ディスクサブシステムで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。	
<i>enable_hashagg</i>	Session	Immediate	User	プランナーがハッシュ集計プランを使用できるようにします。	
<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	プランナーがインデックスのみのスキャンプランを使用できるようにします。	
<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。	

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 User	Description	EPAS Only
<i>enable_material</i>	Session	Immediate	User	プランナーによるマテリアライゼーションの使用を有効にします。	
<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	プランナーがネステッドループの結合プランを使用できるようにします。	
<i>enable_seqscan</i>	Session	Immediate	User	プランナーがシーケンシャルスキャンプランを使用できるようにします。	
<i>enable_sort</i>	Session	Immediate	User	プランナーが明示的なソートステップ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スラッシュエスケープについて警告します。	
<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前を設定します。	
<i>exit_on_error</i>	Session	Immediate	User	エラーが発生したらセッションを終了します。	

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 User	Description	EPAS Only
<i>external_pid_file</i>	Cluster	Restart	EPAS service account	ポストマスター PID を指定されたファイルに書き込みます。	
<i>extra_float_digits</i>	Session	Immediate	User	浮動小数点値に表示される桁数を設定します。	
<i>from_collapse_limit</i>	Session	Immediate	User	サブクエリが折りたたまれない `FROM`-リストサイズを設定します。	
<i>fsync</i>	Cluster	Reload	EPAS service account	ディスクへの更新の同期を強制します。	
<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	遺伝的クエリーの最適化を有効にします。	
<i>geqo_effort</i>	Session	Immediate	User	GEQO：他の GEQO パラメーターのデフォルトを設定するために労力が使用されます。	
<i>geqo_generations</i>	Session	Immediate	User	GEQO：アルゴリズムの反復回数。	
<i>geqo_pool_size</i>	Session	Immediate	User	GEQO：母集団内の個人の数。	
<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：母集団内の選択的圧力。	

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 User	Description	EPAS Only
<i>geqo_threshold</i>	Session	Immediate	User	GEQO が使用される `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 による正確な検索で許可される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	リカバリ中に接続とクエリを許可します。	
<i>hot_standby_feedback</i>	Cluster	Reload	EPAS service account	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、クエリーの競合を回避します。	
<i>huge_pages</i>	Cluster	Restart	EPAS service account	Linux での huge pages の使用。	
<i>icu_short_form</i>	Database	Preset	n/a	ICU の照合順序オーダーの構成を示します。	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	チェックサムが失敗した後も処理を続行します。	

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 User	Description	EPAS Only
<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。（セッションスタートに `PGOPTIONS` で設定することもできます。）	
<i>index_advisor.enabled</i>	Session	Immediate	User	IndexAdvisor 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	インターバル値の表示フォーマットを設定します。	
<i>join_collapse_limit</i>	Session	Immediate	User	<i>FROM</i> -リストのサイズを設定します。これを超えると、 <i>JOIN</i> 構成がフラット化されません。	
<i>krb_caseins_users</i>	Cluster	Reload	EPAS service account	Kerberos および GSS-API ユーザ名を大文字小文字しないものとして扱う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所を設定します。	
<i>lc_collate</i>	Database	Preset	n/a	照合順序オーダーロケールを表示します。	
<i>lc_ctype</i>	Database	Preset	n/a	文字分類と大文字小文字変換ロケールを表示します。	
<i>lc_messages</i>	Session	Immediate	Superuser	メッセージを表示する言語を設定します。	

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 User	Description	EPAS Only
<i>lc_monetary</i>	Session	Immediate	User	金額をフォーマットするための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	監視するホスト名前または 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一覧表示します。(セッションスタートに `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。	
<i>lo_compat_privileges</i>	Session	Immediate	Superuser	ラージオブジェクトの権限チェックの下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	オート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。	

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 User	Description	EPAS Only
<i>log_connections</i>	Cluster/Session	Reload/Immediate	EPAS service account/User	成功した各接続をログに記録します。(セッションスタートに `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。(セッションスタートに `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	ログに記録されたメッセージの冗長性を設定します。	
<i>log_executor_stats</i>	Session	Immediate	Superuser	エグゼキュータ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。	

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 User	Description	EPAS Only
<i>log_filename</i>	Cluster	Reload	EPAS service account	ログファイルのファイル名前パターンを設定します。	
<i>log_hostname</i>	Cluster	Reload	EPAS service account	ホスト名前を接続ログに記録します。	
<i>log_line_prefix</i>	Cluster	Reload	EPAS service account	各ログ行の前に付けら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	このレベル以上で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	プランナ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。	

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 User	Description	EPAS Only
<i>log_rotation_size</i>	Cluster	Reload	EPAS service account	ログファイルの自動ローテーションは、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。	
<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	サブプロセスを開始して、stderr 出力や csv ログをログファイルにキャプチャします。	
<i>maintenance_work_mem</i>	Session	Immediate	User	メンテナンス操作に使用す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。	

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 User	Description	EPAS Only
<i>max_files_per_process</i>	Cluster	Restart	EPAS service account	サーバ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	ファンクション引数の最大数を表示します。	
<i>max_identifier_length</i>	Cluster	Preset	n/a	識別子の最大長さを表示します。	
<i>max_index_keys</i>	Cluster	Preset	n/a	インデックスキーの最大数を表示します。	
<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。	
<i>max_stack_depth</i>	Session	Immediate	Superuser	最大スタック深度をキロバイトで設定します。	
<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。	

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 User	Description	EPAS Only
<i>max_standby_streaming_delay</i>	Cluster	Reload	EPAS service account	ホットスタンバイサーバ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 が拡大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	同時ワーカープロセスの最大数。	
<i>min_wal_size</i>	Cluster	Reload	EPAS service account	WAL ログが削除されるのではなくリサイクルされるしきい値を設定します。デフォルトはメガバイトです。	
<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	オラクル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 User	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	`autoprewarm`バックグラウンドワークを有効にします。	X
<i>pg_prewarm.autoprewarm_interval</i>	Cluster	Reload	EPAS service account	`autoprewarm`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、接続のスタートアップを N 秒待機します。（セッションスタートに `PGOPTIONS` で設定することもできます。）	
<i>pre_auth_delay</i>	Cluster	Reload	EPAS service account	接続のスタートアップを N 秒間待ってから認証します。	
<i>qreplace_function</i>	Session	Immediate	Superuser	クエリ置換ファンクションで使用する機能。 注：内部使用のみ。	X
<i>query_rewrite_enabled</i>	Session	Immediate	User	子テーブルのスキャンは、クエリーにマッチする行がないことがコンストレインによって保証されている場合はスキップされたりされます。	X
<i>query_rewrite_integrity</i>	Session	Immediate	Superuser	問い合わせ書き換えを強制する度合いを設定します。	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 User	Description	EPAS Only
<i>quote_all_identifiers</i>	Session	Immediate	User	SQL フラグメントを生成するときは、すべての識別子をクォートで囲んでください。	
<i>random_page_cost</i>	Session	Immediate	User	順次フェッチされない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、サーバーを再初期化します。	
<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オーダーを設定します。	
<i>segment_size</i>	Cluster	Preset	n/a	ディスクファイルあたりのページ数を表示します。	
<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）のキャラクタ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	サーババージョンを表示します。	
<i>server_version_num</i>	Cluster	Preset	n/a	サーババージョンを整数で表示します。	

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 User	Description	EPAS Only
<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一覧表示します。	
<i>session_replication_role</i>	Session	Immediate	Superuser	トリガーと書き換え規則の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一覧表示します。	
<i>sql_inheritance</i>	Session	Immediate	User	サブテーブルがデフォルトでさまざまなコマンドに含まれるようにします。	
<i>ssl</i>	Cluster	Restart	EPAS service account	SSL 接続を有効にします。	
<i>ssl_ca_file</i>	Cluster	Restart	EPAS service account	SSL 証明書権限ファイルの場所。	
<i>ssl_cert_file</i>	Cluster	Restart	EPAS service account	SSL サーバー証明書ファイルの場所。	
<i>ssl_ciphers</i>	Cluster	Restart	EPAS service account	許可される SSL 暗号のリストを設定します。	

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 User	Description	EPAS Only
<i>ssl_crl_file</i>	Cluster	Restart	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	Restart	EPAS service account	ECDH に使用する曲線を設定します。	
<i>ssl_key_file</i>	Cluster	Restart	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	Restart	EPAS service account	サーバーの暗号スイートのオーダーを優先します。	
<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受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	'...' 文字列でバックスラッシュを文字通り処理します。	
<i>statement_timeout</i>	Session	Immediate	User	ステートメントの最大許容期間を設定します。	
<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してい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。	

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 User	Description	EPAS Only
<i>synchronous_commit</i>	Session	Immediate	User	コミット時に即時 fsync を設定します。	
<i>synchronous_standby_names</i>	Cluster	Reload	EPAS service account	潜在的な同期的スタンバイの名前のリスト。	
<i>syslog_facility</i>	Cluster	Reload	EPAS service account	syslog が有効な場合に使用される syslog *機能*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前を設定します。	
<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の時間。	
<i>tcp_keepalives_interval</i>	Session	Immediate	User	TCP キープアライブが再送信する間の時間。	
<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 User	Description	EPAS Only
<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	`LISTEN`および`NOTIFY`のデバッグ出力を生成します。	
<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	並べ替えの際のリソース使用量に関する情報を出力します。	
<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	`pg_stat_activity.current_query`用に予約しているサイズをバイト単位で設定します。	
<i>track_counts</i>	Session	Immediate	Superuser	データベース活動状況に関する統計を収集します。	
<i>track_functions</i>	Session	Immediate	Superuser	データベース活動状況に関する機能レベルの統計を収集します。	

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 User	Description	EPAS Only
<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	読み取り専用のシリアライズブルトランザクションを、シリアル化の失敗が発生することなく実行できるようになるまで延期するかどうか。	
<i>transaction_isolation</i>	Session	Immediate	User	現在のトランザクションの隔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。	
<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。	

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 User	Description	EPAS Only
<i>update_process_title</i>	Session	Immediate	Superuser	プロセス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.uudecode_redwood</i>	Session	Immediate	User	Oracle で作成された <code>uuencode</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> パッケージを介して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	ミリ秒単位の真空コスト遅延。	
<i>vacuum_cost_limit</i>	Session	Immediate	User	昼寝前に利用可能な真空コスト額。	
<i>vacuum_cost_page_dirty</i>	Session	Immediate	User	真空で汚れたページのバキュームコスト。	
<i>vacuum_cost_page_hit</i>	Session	Immediate	User	バッファキャッシュで見つかったページの真空コスト。	
<i>vacuum_cost_page_miss</i>	Session	Immediate	User	バッファキャッシュに見つからないページの真空コスト。	
<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する必要があるトランザクションの数（ある場合）。	
<i>vacuum_freeze_min_age</i>	Session	Immediate	User	<code>VACUUM</code> がテーブル行をフリーズする最小年齢。	

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 User	Description	EPAS Only
<i>vacuum_freeze_table_age</i>	Session	Immediate	User	<code>VACUUM</code> がテーブル全体をスキャンしてタプルをフリーズする年齢。	
<i>vacuum_multixact_freeze_min_age</i>	Session	Immediate	User	<code>VACUUM</code> 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	<code>VACUUM</code> がタプルをフリーズするためにテーブル全体をスキャン必要がある Multixact の年齢。	
<i>wal_block_size</i>	Cluster	Preset	n/a	先行書き込みログにブロックサイズを表示します。	
<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	スタンバイサーバ用に保持される 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	重要でない変更の場合でも、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。	

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 User	Description	EPAS Only
<i>wal_receiver_status_interval</i>	Cluster	Reload	EPAS service account	WAL レシーバステータスレポートからプライマリへの最大インターバルを設定します。	
<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あたりのページ数を表示します。	
<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。	
<i>wal_sync_method</i>	Cluster	Reload	EPAS service account	ディスクに WAL 更新を強制するために使用されるメソッドを選択します。	
<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	クエリー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。	

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 User	Description	EPAS Only
<i>xmloption</i>	Session	Immediate	User	暗黙的なパースおよびシリアル化操作のXMLデータをドキュメントと見なすかコンテンツフラグメントと見なすかを設定します。	
<i>zero_damaged_pages</i>	Session	Immediate	Superuser	破損したページヘッダーを超えて処理を続行します。	

3.1.3 Configuration Parameters by Functionality

</div>

このセクションでは、構成パラメーターの特定のグループについて詳しく説明します。

各パラメータのセクション見出しの後に、属性のリストが続きます。

- **Parameter Type.** パラメータが受け入れることができる値のタイプ。パラメータタイプの値については、[Setting Configuration Parameters](#) セクションを参照してください。
- **Default Value.** 構成ファイルで値が明示的に設定されていない場合のデフォルト設定。
- **Range.** 許可される値のレンジ。
- **Minimum Scope of Effect.** 個別の設定を行うことができる最小のスコープ。通常、個別の設定の最小スコープは、**cluster** 全体（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）、または **per session**（設定はロール、データベース、または個別のセッションによって異なります）です。（この属性の意味は、表の **Scope of Effect** 列と同じです）。
- **When Value Changes Take Effect.** パラメーターの値への変更を有効にするために必要な最小侵襲アクション。構成ファイルで行われたすべてのパラメータ設定の変更は、データベースサーバをリスタートすると有効になります。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** を使用して有効にできます。最後に、他のパラメータ設定の変更は、結果が **immediate** であるクライアントサイドアクションで有効にすることができます。（この属性は、表の **When Takes Effect** 列と同じ意味を持ちます）。
- **Required Authorization to Activate.** パラメーターの設定の変更を有効にするユーザ認証のタイプ。データベースサーバのリスタートまたは構成ファイルのリロードが必要な場合、ユーザはEPAS サービスアカウント（Advanced Server 互換性インストールモードに応じて **enterprisedb** または **postgres**）でなければなりません。（この属性の意味は、表の **Authorized User** 列と同じです）。

Top Performance Related Parameters

</ div>

このセクションでは、パフォーマンスに最も直接的インパクトを与える構成パラメーターについて説明します。

shared_buffers

</ div>

パラメータタイプ：整数

デフォルト値：32MB

範囲：128kB からシステムに依存

効果の最小範囲：クラスター

値の変更が有効になるとき：再起動

有効化に必要な承認：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量を設定します。デフォルトは通常 32 メガバイト (32MB) ですが、カーネル設定でサポートされていない場合は (initdb で決定されたように) 少なくなる場合があります。この設定は少なくとも 128 キロバイトでなければなりません。(BLCKSZ のデフォルト以外の値は最小値を変更します。) ただし、パフォーマンスを向上させるには通常、最小値よりも大幅に高い設定が必要です。

1GB 以上の RAM を備えた専用の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。shared_buffers のラージ設定でも効果的なワークロードがいくつかありますが、Advanced Server はオペレーティングシステムキャッシュにも依存しているため、shared_buffers への RAM の 40 % 以上のアロケーションが少ない量よりもうまく機能することはほとんどありません。

RAM が 1GB より小さいシステムでは、オペレーティングシステム用に十分なスペースを確保するために、RAM の割合を減らすのが適切です (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 のラージ値は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 の有効なレンジは、一般に 64MB から 512MB です。

このパラメータを大きくすると、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。必要に応じてこれらのパラメーターを調整する方法については、* PostgreSQL Core Documentation * のセクション Shared Memory and Semaphores を参照してください。

work_mem

</ div>

パラメータタイプ：整数

デフォルト値：1MB

範囲：64kB から 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

一時ディスクファイルに書き込む前に、内部ソート操作とハッシュテーブルによって使用されるメモリの量を指定します。値のデフォルトは1メガバイト（1MB）です。複雑なクエリーでは、いくつかの並べ替えまたはハッシュ操作が並行して実行される場合があることに注意してください。各オペレーションは、一時ファイルへのデータの書き込みを開始する前に、この値が指定するだけのメモリを使用できます。また、実行中の複数のセッションがこのような操作を同時に実行している可能性があります。したがって、使用される合計メモリは `work_mem` の値の何倍にもなります。値を選択するときは、この事実に留意する必要があります。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約、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。

通常、妥当な値は4MB～64MBで、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よって異なります。

maintenance_work_mem

</ div>

パラメータタイプ：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

VACUUM、CREATE INDEX、ALTER TABLE ADD FOREIGN KEY などのメンテナンス操作で使用されるメモリの最大量を指定します。デフォルトは16メガバイト（16MB）です。データベースセッションで一度に実行できるのはこれらの操作の1つだけであり、通常、インストールではそれらの多くが同時に実行されないため、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、バキューム処理およびデータベースダンプの復元のパフォーマンスが向上する場合があります。

オートバキュームを実行すると、このメモリが `autovacuum_max_workers` 倍まで割り当てられる可能性があるため、デフォルト値を高く設定しないように注意してください。

ルールとして、これをシステムメモリの約 5 % に設定することをお勧めしますが、約 512MB を超えないようにしてください。値を大きくしても、必ずしもパフォーマンスが向上するとは限りません。

wal_buffers

</ div>

パラメータタイプ：整数

デフォルト値：64kB

範囲：32kB からシステムに依存

効果の最小範囲：クラスター

値の変更が有効になるとき：再起動

有効化に必要な承認：EPAS サービスアカウント

WAL データ用の共有メモリで使用されるメモリの量。デフォルトは 64 キロバイト (64kB) です。データはトランザクションコミットされるたびにディスクに書き出されるため、設定は 1 つの典型的なトランザクションによって生成される WAL データの量を保持するのに十分なラージさである必要があります。

このパラメータを大きくすると、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。必要に応じてこれらのパラメーターを調整する方法については、* PostgreSQL Core Documentation * のセクション「共有メモリとセマフォ」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とは限りませんが、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場合があります。この値を 1MB 程度に増やすと、この問題を軽減できます。非常にビジーなシステムでは、最大約 16MB まで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、このパラメータは Advanced Server の初期共有メモリアロケーションを増やします。そのため、これを増やすと Advanced Server のスタートエラーが発生する場合、オペレーティングシステムの制限を増やす必要があります。

checkpoint_segments

</ div>

廃止されました。このパラメータは Advanced Server でサポートされていません。

checkpoint_completion_target

</ div>

パラメータタイプ：浮動小数点

デフォルト値：0.5

範囲：0 から 1

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：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一部として指定します。これにより、システムが次のチェックポイントに向けて作業を開始する間にチェックポイントの書き込みが広がります。

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意味します。値 0.9 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終了することを意味するため、より長い時間にわたってチェックポイントの書き込みを調整し、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

</ div>

パラメータタイプ：整数

デフォルト値：5 分

範囲：30 秒から 3600 秒

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：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）。デフォルトは 5 分（5min）です。このパラメータを増やすと、クラッシュリカバリに必要な時間を増やすことができます。

checkpoint_timeout を 15 分などの大きな値に増やすと、特に shared_buffers にラージ値を使用する場合に、システムの I/O ロードを減らすことができます。

前述のチェックポイントパラメータの調整の欠点は、システムが適度な追加のディスクスペースを使用し、クラッシュが発生したイベントの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小さな価格です。

max_wal_size

</ div>

パラメータタイプ：整数

デフォルト値：1 ギガバイト

範囲：2 から 2147483647

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：EPAS サービスアカウント

`max_wal_size` は、WAL が自動 WAL チェックポイント間で到達する最大サイズを指定します。これはソフト制限です。WAL サイズは、特別な状況（高ロード、`archive_command` の失敗、または `wal_keep_segments` の設定が高い場合）で `max_wal_size` を超えることがあります。

このパラメータを増やすと、クラッシュリカバリに必要な時間を増やすことができます。このパラメータは、`postgresql.conf` ファイルまたはサーバコマンドラインでのみ設定できます。

`min_wal_size`

</div>

パラメータタイプ：整数

デフォルト値：80 メガバイト

範囲：2 から 2147483647

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：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将来使用するためにリサイクルされます。これにより、WAL 使用量の急増にハンドルするために十分な WAL スペースが予約れます（ラージバッチジョブの実行時など）。このパラメータは、`postgresql.conf` ファイルまたはサーバコマンドラインでのみ設定できます。

`bgwriter_delay`

</div>

パラメータタイプ：整数

デフォルト値：200ms

範囲：10ms から 10000ms

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：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（`bgwriter_lru_maxpages` および `bgwriter_lru_multiplier` パラメータで制御可能）に書き込みを発行します。その後、`bgwriter_delay` ミリ秒スリープし、繰り返します。

デフォルト値は 200 ミリ秒（200ms）です。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 のマルチプルではない値に設定すると、次に高い 10 のマルチプル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` をチューニングするときは、デフォルト値から減らす必要があります。このパラメータは、使用率が非常に低いシステムでの電力消費を節約することを除いて、ほとんど増加しません。

seq_page_cost

</ div>

パラメータタイプ：浮動小数点

デフォルト値： 1

範囲： 0 から 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

シリーズのシーケンシャルフェッチのパートであるディスクページフェッチのコストに関するプランナの見積もりを設定します。デフォルトは 1.0 です。この名前は、同じ名前のテーブルスペースパラメータを設定することにより、特定のテーブルスペースに対してオーバーライドできテーブルスペース。（* PostgreSQL Core Documentation *の ALTER TABLESPACE コマンドを参照してください）。

デフォルト値はキャッシングがほとんどないことを前提としているため、頻繁に減らすことをお勧めします。データベースが物理的メモリよりも大幅に大きい場合でも、このパラメータを 1 より小さい（デフォルト値 1 ではなく）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できます。データベースがメモリ内に完全に収まる場合、この値をさらに大きく、おそらく 0.1 に下げることができます。

random_page_cost

</ div>

パラメータタイプ：浮動小数点

デフォルト値： 4

範囲： 0 から 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

順次にフェッチされないディスクページのコストに関するプランナの見積もりを設定します。デフォルトは 4.0 です。この名前は、同じ名前のテーブルスペースパラメータを設定することにより、特定のテーブルスペースに対してオーバーライドできテーブルスペース。（* PostgreSQL Core Documentation *の ALTER TABLESPACE コマンドを参照してください）。

seq_page_cost に対してこの値を小さくすると、システムはインデックススキャンを優先します。値を大きく make と、インデックススキャンのコストが比較的高くなります。両方の値を一緒に増減して、cpu_tuple_cost パラメーターと cpu_index_tuple_cost パラメーターで記述される 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。

デフォルト値はキャッシングがほとんどないことを前提としているため、頻繁に減らすことをお勧めします。データベースが物理的メモリよりもかなり大きい場合でも、このパラメータを（デフォルトの4ではなく）2に設定して、その方法でより良いクエリープランが得られるかどうかを確認できます。データベースがメモリ内に完全に収まる場合、この値をさらに大きく、おそらく 0.1 に下げることができます。

システムで許可されますが、`random_page_cost` を `seq_page_cost` より小さい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内に収まるシーケンスは、それらを等しい（または非常に近い）に設定することはペナルティにかなっていません。また、大量にキャッシュされたデータベースでは、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小さいため、CPU パラメータに比べて両方の値を低くする必要があります。

`effective_cache_size`

</ div>

パラメータタイプ：整数

デフォルト値：128MB

範囲：8kB から 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

単一のクエリーで使用可能なディスクキャッシュの有効サイズに関するプランナーの仮定を設定します。これは、インデックスを使用するコストの見積もりに組み込まれます。値が大き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、値が小さいほどシークエンシャルスキャンが使用される可能性が高くなります。このパラメータを設定するときは、Advanced Server の共有バッファと、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部分の両方を考慮する必要があります。また、使用可能なスペースを共有する必要があるため、異なるテーブルで予想される同時クエリ数をアカウントしてください。このパラメータは、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も予約しません。推定目的でのみ使用されます。デフォルトは 128 メガバイト（128MB）です。

このパラメータの設定が低すぎると、プランナは、有益な場合でもインデックスを使用しないことを決定する場合があります。`effective_cache_size` を物理的メモリの 50 % に設定することは、通常の保守的な設定です。より積極的な設定は、物理的メモリの約 75 % です。

`synchronous_commit`

</ div>

パラメータタイプ：ブール値

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

コマンドがクライアントに `success` 指示を返す前に、WAL レコードがディスクに書き込まれるのをトランザクションコミットが待機するかどうかを指定します。デフォルトの安全な設定はオンです。オフの場合、成功がクライアントに報告されてから、トランザクションがサーバ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可能性があります。（最大遅延は `wal_writer_delay` の 3 倍です。）

`fsync` とは異なり、このパラメータをオフに設定しても、データベースの不整合のリスクは発生しません。オペレーティングシステムまたはデータベースクラッシュにより、コミットされた最近のトランザクションが失われる可能性があります。データベースの状態はそれらのトランザクションが失われた場合と同じになりますきれいに中止しました。

そのため、トランザクションの永続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、`synchronous_commit` をオフにすることは有用な代替手段となります。詳細については、* PostgreSQL Core Documentation *のセクション `Asynchronous Commit` を参照してください。

このパラメータ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。例、デフォルトが反対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コミットするには、トランザクション内で `SET LOCAL synchronous_commit TO OFF` を発行します。

`edb_max_spins_per_delay`

</div>

パラメータタイプ：整数

デフォルト値：1000

範囲：10 から 1000

効果の最小範囲：クラスターごと

値の変更が有効になるとき：再起動

有効化に必要な承認：EPAS サービスアカウント

`edb_max_spins_per_delay` を使用して、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。`edb_max_spins_per_delay` の代替値を指定しない場合、サーバーは **1000** のデフォルト値を適用します。

これは、NUMA（非均一メモリアクセス）アーキテクチャを使用するシステムに役立つ場合があります。

pg_prewarm.autoprewarm

</div>

パラメータタイプ：ブール値

デフォルト値：true

範囲：{true | false}

効果の最小範囲：クラスター

値の変更が有効になるとき：再起動

有効化に必要な承認：EPAS サービスアカウント

このパラメータは、データベースサーバが **autoprewarm** を実行するかどうかを制御します。**autoprewarm** は、シャットダウン前に共有バッファをディスクに自動的にダンプするバックグラウンドワーカプロセスです。次に、サーバーの次回起動時に共有バッファ **prewarms** を使用します。つまり、ディスクからバッファプールにブロックをロードします。

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ディングすることにより、サーバーの再起動後のウォームアップ時間を短縮することです。

pg_prewarm.autoprewarm がオンに設定されている場合、**autoprewarm** ワーカーが有効になります。パラメータがオフに設定されている場合、**autoprewarm** は無効になります。パラメータはデフォルトでオンになっています。

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** 設定パラメータにリストされているライブラリーに **\$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リスタートします。その後、サーバーが一貫した状態になった直後に、**autoprewarm** バックグラウンドワーカが起動します。

autoprewarm プロセスは、バッファプールにフリーバッファスペースがなくなるまで、**\$PGDATA/autoprewarm.blocks** に以前に記録されたブロックのローディングをスタートします。この方法では、リカバリ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ロードされた新しいブロックは置き換えられません。

autoprewarm プロセスがディスクからのバッファのローディングを完了すると、**pg_prewarm.autoprewarm_interval** パラメータで指定されたインターバルで共有バッファをディスクに定期的にダンプします。**autoprewarm** バックグラウンドワーカの詳細については、**pg_prewarm.autoprewarm_interval** を参照してください。次のサーバーのリスタートに、プレウォームプロセスは、ディスクに最後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アップします。

pg_prewarm.autoprewarm_interval

</div>

パラメータタイプ：整数

デフォルト値：300s

範囲：0 秒から 2147483 秒

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：EPAS サービスアカウント

これは、autoprewarm バックグラウンドワーカ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です。デフォルトは 300 秒です。0 に設定すると、共有バッファは定期的な間隔ではなく、サーバーがシャットダウンされたときにのみダンプされます。

autoprewarm バックグラウンドワーカの詳細については、[pg_prewarm.autoprewarm](#) を参照してください。

Resource Usage / Memory

</div>

このセクションの構成パラメータは、メモリに関連するリソース使用量を制御します。

edb_dynatune

パラメータタイプ：整数

デフォルト値：0

範囲：0 から 100

効果の最小範囲：クラスター

値の変更が有効になるとき：再起動

有効化に必要な承認：EPAS サービスアカウント

ホストマシンの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システムがホストシステムのリソースをどれだけ使用するかを決定しホスト。

Advanced Server が最初にインストールされるとき、edb_dynatune パラメータは、インストールされたホストマシン（開発マシン、混在マシン、または専用サーバー）の選択された使用法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 `postgresql.conf` ファイルのさまざまな構成パラメーターを調整してパフォーマンスを向上さオーダー必要はありません。

edb_dynatune パラメータは、0～100 の整数値に設定できます。値を 0 にすると、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のリソース使用量が `postgresql.conf` ファイル内の他の構成パラメーターの制御下に完全に残ります。

ゼロ以外の低い値（1-33 など）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専用にします。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で使用されます。

34～66 のレンジの値は、適度な量のリソースをデータベースサーバ専用にします。この設定は、Advanced Server と同じマシンで実行されている他のアプリケーションの固定数を持つ専用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。

最も高い値（67～100 など）は、サーバーのリソースのほとんどをデータベースサーバ専用にします。この設定は、Advanced Server の実行専用のホストマシンに使用されます。

`edb_dynatune` の値が選択されると、`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のパフォーマンスをさらに微調整できます。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。パラメータの値を変更するには、設定パラメータのコメントを外し、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

パラメータタイプ：列挙型

デフォルト値：oltp

範囲：{oltp | reporting | mixed}

効果の最小範囲：クラスター

値の変更が有効になるとき：再起動

有効化に必要な承認：EPAS サービスアカウント

このパラメータは、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するために使用されます。

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ときに推奨されます。
- **reporting**. 大量のデータレポートに使用されるデータベースサーバーに推奨。
- **mixed**. トランザクション処理とデータレポートの両方を提供するサーバーに推奨。

Resource Usage / EDB Resource Manager

</ div>

このセクション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

パラメータタイプ：整数

デフォルト値：16

範囲：0 から 65536

効果の最小範囲：クラスター

値の変更が有効になるとき：再起動

有効化に必要な承認：EPAS サービスアカウント

このパラメータは、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` で指定された値よりも多くのリソースグループを作成できますが、これらのグループのプロセスがアクティブに使用し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。

パラメータ `edb_max_resource_groups` は、不足しないように維持する予定のグループ数よりも十分に大きく設定する必要があります。

edb_resource_group

パラメータタイプ：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

`edb_resource_group` パラメータを、グループのリソースタイプ設定に従って、EDB Resource Manager によって現在のセッションが制御されるリソースグループの名前に設定します。

パラメータが設定されていない場合、現在のセッションは EDB Resource Manager を使用しません。

Query Tuning

</ div>

このセクションでは、オプティマイザのヒントに使用される構成パラメーターについて説明します。

enable_hints

パラメータタイプ：ブール値

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

enable_hints がオンの場合、SQL コマンドに埋め込みれたオプティマイザーヒントが利用されます。このパラメータがオフの場合、オプティマイザーのヒントは無視されます。

Query Tuning / Planner Method Configuration

</ div>

このセクションでは、プランナメソッドの構成に使用される構成パラメーターについて説明します。

edb_enable_pruning

パラメータタイプ：ブール値

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

TRUE に設定すると、edb_enable_pruning により、クエリープランナはテーブルパーティションを早期にブルーニングできます。Early-pruning は、クエリープランナがクエリープランを生成するクエリー before で検索されないパーティションを prune (つまり無視) できることを意味します。これにより、検索されないパーティションのクエリープランの生成が排除されるため、パフォーマンス時間が向上します。

逆に、late-pruning は、クエリープランナがパーティション after をブルーニングして、各パーティションのクエリープランを生成することを意味します。(constraint_exclusion 設定パラメータは、遅延ブルーニングを制御します。)

早期ブルーニングの機能は、WHERE 句のクエリーの性質に依存します。アーリーブルーニングは、WHERE column = literal 型のコンストレインを持つシンプルクエリ (たとえば、WHERE deptno = 10) など) のみ使用できます。

アーリーブルーニングは、WHERE column = expression (WHERE deptno = 10 + 5) など) などのより複雑な問い合わせには使用されません。

Reporting and Logging / What to Log

</div>

このセクションの構成パラメータは、レポートとロギングを制御します。

trace_hints

パラメータタイプ：ブール値

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

ヒントがプランナによって使用されたかどうかに関するより詳細な情報を提供するには、オプティマイザのヒント機能とともに使用します。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;
```

以下に示す NO_INDEX ヒント付きの SELECT コマンドは、前述の構成パラメーターが設定されたときに生成される追加情報を示しています。

```
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.

*                                QUERY PLAN

-----

Seq Scan on accounts (cost=0.00..14461.10 rows=1 width=97)
  * Filter: (aid = 100)
(2 rows)
```

edb_log_every_bulk_value

</div>

パラメータタイプ: ブール値**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 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ギン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 設定パラメータによって制御できます。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、ログメッセージは一括処理ごとに 1 回、処理された行数とともに記録されます。さらに、期間はバルク処理ごとに 1 回発行されます。デフォルトは false に設定されています。

Auditing Settings

</div>

このセクションでは、Advanced Server データベース監査機能で使用する構成パラメーターについて説明します。

edb_audit

</div>

パラメータタイプ: 列挙型**デフォルト値:** none**範囲:** {none | csv | xml}**効果の最小範囲:** クラスター**値の変更が有効になるとき:** 再読み込み**有効化に必要な承認:** 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。値 xml または csv は、データベース監査を有効にします。これらの値は、監査情報がキャプチャされ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、これもデフォルト。

edb_audit_directory

</div>

パラメータタイプ：文字列

デフォルト値：edb_audit

範囲：該当なし

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：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。ディレクトリのパスは、Advanced Server data ディレクトリに対する絶対パスまたは相対パスにすることができます。

edb_audit_filename

</div>

パラメータタイプ：文字列

デフォルト値：audit-%Y%m%d_%H%M%S

範囲：該当なし

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：EPAS サービスアカウント

監査情報が保存される監査ファイルのファイル名前を指定します。デフォルトのファイル名前は 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応じて適切な現在の値に置き換えられます。

edb_audit_rotation_day

</div>

パラメータタイプ：文字列

デフォルト値：every

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

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：EPAS サービスアカウント

監査ファイルをローテート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。ローテーションを無効にするには、値を none に設定します。ファイルを毎日ローテートするには、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トするには、値を目的の曜日に設定します。

edb_audit_rotation_size

</ div>

パラメータタイプ：整数

デフォルト値：0MB

範囲：0MB から 5000MB

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：EPAS サービスアカウント

ファイルのローテーションが強制されるとき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。パラメータがコメント化されているか、0 に設定されている場合、サイズごとのファイルのローテーションは発生しません。

edb_audit_rotation_seconds

</ div>

パラメータタイプ：整数

デフォルト値：0s

範囲：0 秒から 2147483647 秒

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：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、このパラメータを 0 に設定します。

edb_audit_connect

</ div>

パラメータタイプ：列挙型

デフォルト値：failed

範囲：{none | failed | all}

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：EPAS サービスアカウント

ユーザーによるデータベースコネクション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。失敗したすべての接続試行を監査するには、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

</ div>

パラメータタイプ：列挙型

デフォルト値：none

範囲：{none | all}

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：EPAS サービスアカウント

接続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

</ div>

パラメータタイプ：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} ...

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：EPAS サービスアカウント

この設定パラメータは、特定の SQL コマンドに関連するステートメントだけでなく、SQL ステートメントのさまざまなカテゴリーの監査を指定するために使用されます。エラーをログに記録するには、パラメータ値を **error** に設定します。CREATE TABLE、ALTER TABLE などのすべての DDL ステートメントを監査するには、パラメータ値を **ddl** に設定します。特定のタイプの DDL ステートメントを監査するために、パラメータ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。さらに、**create table**、**create view**、**drop role** などのコマンドの後にオブジェクトタイプを指定できます。INSERT、UPDATE、DELETE、または TRUNCATE などのすべての変更ステートメントは、**edb_audit_statement** を **dml** に設定して監査できます。特定のタイプの DML ステートメントを監査するために、パラメータ値に特定の SQL コマンド、**insert**、**update**、**delete**、または **truncate** を含めることができます。これらの SQL コマンドに関するステートメントを監査するには、パラメータ値 **select**、**grant**、**revoke**、または **rollback** を含めます。SET ステートメントを監査するには、SET にパラメータ値を含めます。値を **all** に設定すると、すべてのステートメントが監査されますが、**none** はこの機能を無効にします。オブジェクトごとのレベルの監査では、SELECT、UPDATE、DELETE、INSERT ステートメント、(@) などのオブジェクト権限で許可された操作を監査します。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** パラメータには、(@) **include** および (-) **exclude** シンボルを持つグループ名前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

</ div>

パラメータタイプ：文字列**デフォルト値：**なし**最小効果範囲：**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：**ユーザー

edb_audit パラメータが **csv** または **xml** に設定されている場合、**edb_audit_tag** を使用して、監査ログに含まれる文字列値を指定します。

edb_audit_destination

</ div>

パラメータタイプ：列挙型**デフォルト値：**file**範囲：**{file | syslog}**効果の最小範囲：**クラスター**値の変更が有効になるとき：**再読み込み**有効化に必要な承認：**EPAS サービスアカウント

監査ログ情報を、`edb_audit_directory` パラメータで指定されたディレクトリに記録するか、`syslog` プロセスで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（デフォルト設定）。

`/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 Advanced Server でのみ有効であり、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。

`edb_log_every_bulk_value`

</div>

`edb_log_every_bulk_value` 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

</div>

このセクションでは、ロケールとフォーマットに影響する構成パラメーターについて説明します。

`icu_short_form`

パラメータタイプ：文字列

デフォルト値：なし

範囲：該当なし

最小効果範囲：データベース

値の変更が有効になるとき：n/a

有効化に必要な承認：該当なし

設定パラメータ `icu_short_form` は、データベースまたは Advanced Server インスタンスに割り当てられたデフォルトの ICU 短縮名前を含むパラメータです。ICU の短いフォームと Unicode 照合アルゴリズムに関する一般情報については、[Unicode Collation Algorithm](#) を参照してください。

この設定パラメータは、`CREATE DATABASE` コマンドを `ICU_SHORT_FORM` パラメータと一緒に使用する場合に設定されます。この場合、このデータベースに接続するときに、指定された短いフォーム名前が設定されて `icu_short_form` 設定パラメータに表示されるか、`initdb` で Advanced Server インスタンスが作成されます--`icu_short_form` オプションと一緒に使用されるコマンド。この場合、指定された短いフォーム名前が設定され、その Advanced Server インスタンスのデータベースに接続するときに `icu_short_form` 設定パラメータに表示されます。データベースは、異なる short を持つ独自の `ICU_SHORT_FORM` パラメータでそれをオーバーライドしませんフォーム

説明した方法で設定すると、`icu_short_form` 設定パラメータは変更できません。

Client Connection Defaults / Statement Behavior

</div>

このセクションでは、ステートメントの動作に影響する構成パラメーターについて説明します。

default_heap_fillfactor

パラメータタイプ：整数

デフォルト値：100

範囲：10 から 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

CREATE TABLE コマンドで FILLFACTOR ストレージパラメータが省略されている場合に、テーブルの fillfactor を設定します。

テーブルの fillfactor は 10～100 の割合です。100 (完全なパッキング) がデフォルト。より小さい fillfactor が指定されている場合、INSERT 操作は、指定されたパーセンテージにのみテーブルページをパックします。各ページの残りのスペースは、そのページの行を更新ために予約れています。これにより、UPDATE 行の更新されたコピーをオリジナルのページと同じページに配置できます。これは、別のページに配置するよりも効率的です。エントリが決して更新されないテーブルの場合、完全なパッキングが最良の選択ですが、頻繁に更新されるテーブルでは、より小さい fillfactor が適切です。

edb_data_redaction

パラメータタイプ：ブール値

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

データ編集とは、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露出を制限するポリシーのサポートです。

デフォルト設定は TRUE であるため、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データ修正が適用されます。

- 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は、墨消し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。

パラメータを FALSE に設定して無効にすると、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を受け取ります。

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

Client Connection Defaults / Other Defaults

</ div>

このセクションのパラメーターは、その他のさまざまな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：クラスター

値の変更が有効になるとき：再起動

有効化に必要な承認：EPAS サービスアカウント

オラクルサーバーへのオラクル Call Interface (OCI) データベースリンクを作成する前に、Advanced Server を正しいオラクルホームディレクトリに転送する必要があります。Linux の LD_LIBRARY_PATH 環境変数（または Windows の PATH）をオラクルクライアント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。

Windows の場合のみ、代わりに postgresql.conf ファイルの oracle_home 設定パラメータの値を設定できます。oracle_home 設定パラメータで指定された値は、Windows の PATH 環境変数をオーバーライドします。

Advanced Server をスタートするたびに、Linux の LD_LIBRARY_PATH 環境変数（Windows の PATH 環境変数または oracle_home 設定パラメータ）を適切に設定する必要があります。

**** Windows のみ：*** postgresql.conf ファイルで oracle_home 設定パラメータを設定するには、ファイルを編集し、次の行を追加します。

```
oracle_home = 'lib_directory'
```

lib_directory を oci.dll を含む Windows ディレクトリの名前に置き換えます。

oracle_home 設定パラメータを設定した後、変更を有効にするにはサーバーをリスタートする必要があります。Windows サービスコンソールからサーバーを再起動します。

odbc_lib_path

パラメータタイプ：文字列

デフォルト値：なし

範囲：該当なし

効果の最小範囲：クラスター

値の変更が有効になるとき：再起動

有効化に必要な承認：EPAS サービスアカウント

ODBC ドライバマネージャを使用し、標準以外の場所にインストールする場合、`postgresql.conf` ファイルの `odbc_lib_path` 設定パラメータ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、ドライバマネージャ共有ライブラリ（通常は `libodbc.so`）への完全なパス名を含める必要があります。

Compatibility Options

</ div>

このセクションで説明する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

パラメータタイプ：ブール値

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

DATE がコマンドの列のデータタイプとして表示される場合、設定パラメータ `edb_redwood_date` が TRUE に設定されている場合、テーブル定義がデータベースに格納されるときに、TIMESTAMP に変換されます。したがって、時間コンポーネントも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タイプは、PostgreSQL のネイティブ DATE データタイプのままであり、データベースにそのまま格納されます。PostgreSQL の DATE データタイプは、列に時間コンポーネントのない日付のみを格納します。

`edb_redwood_date` の設定に関係なく、コンテキストが SPL 宣言セクションの変数のデータタイプデータタイプ、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のデータタイプ、または SPL ファン

クシヨンの戻り値の型は、常に内部的に `TIMESTAMP` に変換されるため、時間コンポーネントが存在する場合はハンドルできます。

`edb_redwood_greatest_least`

パラメータタイプ：ブール値

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

`GREATEST` ファンクションは、パラメーターのリストから最大値を持つパラメータを返します。`LEAST` ファンクションは、パラメーターのリストから値が最小のパラメータを返します。

`edb_redwood_greatest_least` が `TRUE` に設定されている場合、`GREATEST` および `LEAST` 関数は、パラメーターの少なくとも 1 つが `null` の場合に `null` を結果ます。

```
SET edb_redwood_greatest_least TO on;
```

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

```
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` である場合を除き、`null` パラメーターは無視されます。その場合、関数によって `null` が返されます。

```
SET edb_redwood_greatest_least TO off;
```

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

```
greatest
-----
      *      3
(1 row)
```

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

```
greatest
-----
(1 row)
```

edb_redwood_raw_names

パラメータタイプ：ブール値

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

プログラムがデフォルト値のデータベースにユーザされている場合、Redwood カタログ（つまり、システムカタログのプレフィックスが ALL_、DBA_、または USER_）。さらに、引用符は、引用符で囲まれて作成された名前を囲み引用符。

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。したがって、引用符で囲まずに作成された名前は、PostgreSQL で期待されるように小文字で表示されます。引用符で囲んで作成された名前は、作成されたとおりに表示されますが、引用符は表示されません。

例、次のユーザ名前が作成され、そのユーザとの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 に設定した Redwood カタログ USER_TABLES から表示すると、引用符名前を除き、名前は大文字で表示されます。

```
edb=> SELECT * FROM USER_TABLES;
```

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)

edb_redwood_raw_names を TRUE に設定して表示すると、Mixed_Case 名前を除き、名前は小文字で表示され引用符。

```
edb=> SET edb_redwood_raw_names TO true;
SET
edb=> SELECT * FROM USER_TABLES;
 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 カタログから見ると大文字と小文字がマッチするようになりました。

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

edb_redwood_strings

パラメータタイプ：ブール値

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

edb_redwood_strings パラメータが TRUE に設定されている場合、文字列が null 変数または null 列と連結されると、結果はオリジナル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 の動作が維持されます。これは、文字列と null 変数または null 列の結合が null 結果をもたらすことです。

次の例は、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 という名前付けの列があります。次のクエリーは、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結合すると、最終結果 null が生成されるため、クエリー結果には歩合のある従業員のみが表示されます。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;
*      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ーです。ここでは、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結合すると、空でない文字列が生成され文字列。

```
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;
*      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
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

パラメータタイプ：ブール値

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

statement level transaction isolation という用語は、SQL コマンドでランタイムエラーが発生したときに、その単一のコマンドによって引き起こされ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示します。例、1つの UPDATE コマンドが5行を正常に更新したが、6行を更新しようとする例外が発生する場合、この UPDATE コマンドによって行われた6行すべて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。

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スタート以降のデータベース上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状態のままになり、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、例外は以前のコミットされていないデータベース更新を自動的に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、例外はコミットされていないデータベース更新をロールバックします。

!!! Note * edb_stmt_level_tx を TRUE に設定するのは、どうしても必要な場合のみにしてください。パフォーマンスにインパクトを与える可能性があります。

次の PSQL での実行例は、edb_stmt_level_tx が FALSE である場合、2 番目の INSERT コマンドのアポートによって最初の INSERT コマンドもロールバックされ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。そうしないと、すべてのステートメントが自動的にコミットし、edb_stmt_level_tx の効果のこのデモの目的に反します。

```

\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;

empno  |  ename  | deptno

```

(次のページに続く)

(前のページからの続き)

```
-----+-----+-----
(0 rows)
```

次の例では、`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ていません。この時点で、最初の `INSERT` コマンドは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;

empno | ename | deptno
-----+-----+-----
* 9001 | JONES |    40
(1 row)

COMMIT;
```

`COMMIT` コマンドの代わりに `ROLLBACK` コマンドが発行された場合、従業員番号 9001 の挿入もロールバックされます。

db_dialect

パラメータタイプ：列挙型

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

ネイティブ PostgreSQL システムカタログ `pg_catalog` に加えて、Advanced Server には拡張カタログビューが含まれています。これは、展開カタログビューの `sys` カタログです。`db_dialect` パラメータは、これらのカタログが名前解決のために検索されるオーダーを制御します。

`postgres` に設定すると、ネームスペースの優先順位は `pg_catalog`、次に `sys` になり、PostgreSQL カタログに最高の優先順位が与えられます。`redwood` に設定すると、ネームスペースの優先順位は `sys`、次に `pg_catalog` になり、展開カタログビューの優先順位が最も高くなります。

default_with_rowids

パラメータタイプ：ブール値

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

on に設定すると、CREATE TABLE には新しく作成されたテーブルに ROWID 列が含まれ、SQL コマンドで被参照できます。以前のバージョンの Advanced Server ROWIDs は OIDs にマップされていましたが、Advanced Server バージョン 12 以降では、ROWID は 1 で始まり、ROWIDs オプションで作成されたテーブルの行に割り当てられたシーケンスに基づく自動インクリメント値です。デフォルトでは、ROWID 列に一意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。

Advanced Server 11 以前のバージョンから 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 が含まれている場合、--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、Advanced Server バージョン 12 以降でテーブル作成を続行します。pg_upgrade：エラーが発生しました。ただし、テーブルに OIDs または ROWIDs が含まれている場合は、次の手順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. テーブルを削除してから、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後、ターゲットデータベースに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復元します。

optimizer_mode

パラメータタイプ：列挙型

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

オプティマイザ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。

次の表は、可能な値を示しています。

Hint	Description
<i>ALL_ROWS</i>	結果セットのすべての行を取得するために最適化します。
<i>CHOOSE</i>	結果セットから取得されると想定される行数に基づくデフォルトの最適化は行われません。これがデフォルト。
<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 行の取得用に最適化します。
<i>FIRST_ROWS_100</i>	結果セットの最初の 100 行の取得用に最適化します。
<i>FIRST_ROWS_1000</i>	結果セットの最初の 1000 行の取得用に最適化します。
	NA

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、結果セットの残りを放棄することに関心があるという仮定に基づいています。クエリーに割り当てられたリソースは、そのように調整されます。

Customized Options

</ div>

Advanced Server の以前のリリースでは、アドオンモジュール（手続き言語など）によって追加されることが通常知られていないパラメーターには `custom_variable_classes` が必要でした。

`custom_variable_classes`

`custom_variable_classes` パラメータは、Advanced Server 9.2 では非推奨です。以前にこのパラメータに依存していたパラメーターは、そのサポートを必要としなくなりました。

`dbms_alert.max_alerts`

パラメータタイプ：整数

デフォルト値：100

範囲：0 から 500

効果の最小範囲：クラスター

値の変更が有効になるとき：再起動

有効化に必要な承認：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

パラメータタイプ：整数

デフォルト値：30 Kb

範囲：30 Kb～256 Kb

効果の最小範囲：クラスター

値の変更が有効になるとき：再起動

有効化に必要な承認：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

パラメータタイプ：ブール値

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

EDB-PSQL または PSQL セッションで Index Advisor を一時的に中断する機能を提供します。
index_advisor.enabled 設定パラメータを使用オーダーには、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、次の例に示す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
```

次の例に示すように、SET コマンドを使用してパラメータ設定を変更し、Index Advisor が代替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;
*
          QUERY PLAN
```

(次のページに続く)

(前のページからの続き)

```

-----
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;
      *
      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

パラメータタイプ：ブール値

デフォルト値：false

範囲：{true | false}

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：EPAS サービスアカウント

SQL/Protect が保護されたロールをアクティブにモニタリングするかどうかを制御するには、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します。SQL/Protect でモニタリングを開始する準備ができれば、このパラメータをオンに設定します。

edb_sql_protect.レベル

パラメータタイプ：列挙型

デフォルト値：passive

範囲：{learn | passive | active}

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：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 によって実行されるアクションを設定します。

`edb_sql_protect.level` 設定パラメータを以下の値のいずれかに設定して、学習モード、パッシブモード、またはアクティブモードのいずれかを使用できます。

- **学習**保護されたロールのアクティビティを追跡し、ロールで使用されるリレーションを記録します。これは、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、最初に SQL / Protect を構成するときに使用されます。
- **** 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を停止しません。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ステップです。これは基本的に侵入検知モードで動作し、適切に監視されている場合は稼働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。これは、危険なクエリの実行を防ぐ SQL ファイアウォールとして動作します。これは、攻撃者がアプリケーションの背後にある脆弱点とデータベースのタイプを特定しようとしている場合の早期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て、攻撃者がシステムに侵入する別のメソッドを見つける前に管理者に警告することを可能にします。

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

`edb_sql_protect.max_protected_relations`

パラメータタイプ：整数

デフォルト値：1024

範囲：1 から 2147483647

効果の最小範囲：クラスター

値の変更が有効になるとき：再起動

有効化に必要な承認：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リレーションの総数は、保護されたリレーションの数に保護されたロールの数を掛けたものになることにノートしてください。すべての保護されたリレーションは、共有メモリのスペースを消費します。可能な限り最大の保護されたリレーションのためのスペースは、データベースサーバのスタートアップに予約れます。

`edb_sql_protect.max_protected_relations` が有効なレンジ外の値（例、2,147,483,648 の値）に設定されているときにサーバーが起動されると、ワーニングメッセージがデータベースサーバのログファイルに記録されます。

```
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は正常に起動しますが、`edb_sql_protect.max_protected_relations` はデフォルト値の 1024 に設定されています。

パラメータのレンジは整数データタイプ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スタートアップに使用されるパラメータ値によって異なります。

必要なスペース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。共有メモリ内のスペースを予約できないような値の場合、データベースサーバのスタートアップは失敗し、次のようなエラーメッセージが表示されます。

```
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 PostgreSQL's
request for a shared memory segment exceeded available memory, swap
space or huge pages. To reduce the request size (currently 2070118400
bytes), reduce PostgreSQL's shared memory usage, perhaps by reducing
shared_buffers or max_connections.
```

このような場合は、データベースサーバを正常に起動できるようになるまでパラメータ値を減らします。

`edb_sql_protect.max_protected_roles`

パラメータタイプ: 整数

デフォルト値: 64

範囲: 1 から 2147483647

効果の最小範囲: クラスター

値の変更が有効になるとき: 再起動

有効化に必要な承認: 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。

保護されたロールはすべて、共有メモリのスペースを消費します。サーバーは、保護されたロールの数に保護されたリレーションの数 (`edb_sql_protect.max_protected_relations`) を掛けたスペースを確保することにノートしてください。データベースサーバのスタートアップに、保護されたロールの最大可能スペースが予約れます。

`edb_sql_protect.max_protected_roles` が有効なレンジ外の値 (例、2,147,483,648 の値) に設定されているときにデータベースサーバが起動すると、データベースサーバのログファイルにワーニング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` はデフォルト値の 64 に設定されています。

パラメータのレンジは整数データタイプ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スタートアップに使用されるパラメータ値によって異なります。

必要なスペース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。共有メモリ内のスペースを予約できないような値の場合、データベースサーバのスタートアップは失敗し、次のようなエラーメッセージが表示されます。

```
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 PostgreSQL's
request for a shared memory segment exceeded available memory, swap
space or huge pages. To reduce the request size (currently 2070118400
bytes), reduce PostgreSQL's shared memory usage, perhaps by reducing
shared_buffers or max_connections.
```

このような場合は、データベースサーバを正常に起動できるようになるまでパラメータ値を減らします。

edb_sql_protect.max_queries_to_save

パラメータタイプ：整数

デフォルト値：5000

範囲：100 から 2147483647

効果の最小範囲：クラスター

値の変更が有効になるとき：再起動

有効化に必要な承認：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。

保存されるすべてのクエリーは、共有メモリのスペースを消費します。保存可能なクエリの最大可能スペースは、データベースサーバのスタートアップに予約れます。

`edb_sql_protect.max_queries_to_save` が有効なレンジ外の値（例、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すると、データベースサーバのログファイルにワーニング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` はデフォルト値の 5000 に設定されています。

パラメータのレンジは整数データタイプ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スタートアップに使用されるパラメータ値によって異なります。

必要なスペース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。共有メモリ内のスペースを予約できないような値の場合、データベースサーバのスタートアップは失敗し、次のようなエラーメッセージが表示されます。

```

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 PostgreSQL's
request for a shared memory segment exceeded available memory, swap
space or huge pages. To reduce the request size (currently 2070118400
bytes), reduce PostgreSQL's shared memory usage, perhaps by reducing
shared_buffers or max_connections.

```

このような場合は、データベースサーバを正常に起動できるようになるまでパラメータ値を減らします。

edb_wait_states.ディレクトリ

パラメータタイプ：文字列

デフォルト値：edb_wait_states

範囲：該当なし

効果の最小範囲：クラスター

値の変更が有効になるとき：再起動

有効化に必要な承認：EPAS サービスアカウント

EDB 待機状態ログファイルが保存されるディレクトリパスを設定します。指定されたパスは、絶対パスであり、相対パスではありません。ただし、デフォルト設定はedb_wait_statesであり、\$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](#) の下の EDB Wait States を参照してください。

edb_wait_states.retention_period

パラメータタイプ：整数

デフォルト値：604800s

範囲：86400s から 2147483647s

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：EPAS サービスアカウント

EDB 待機状態のログファイルが削除されるまでの期間を設定します。デフォルトは 604800 秒、つまり 7 日です。EDB の待機状態については、[Performance Analysis and Tuning](#) の下の EDB Wait States を参照してください。

edb_wait_states.sampling_interval

パラメータタイプ：整数

デフォルト値： 1s

範囲： 1 秒から 2147483647 秒

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：EPAS サービスアカウント

EDB 待機状態の 2 つのサンプリングサイクル間のタイミングインターバルを設定します。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](#) の下の **EDB Wait States** を参照してください。

edbldr.empty_csv_field

パラメータタイプ：列挙型

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

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。edbldr.empty_csv_field パラメータの有効な値は次のとおりです。

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 does not contain quotes and as an empty string if it contains quotes.

EDB * Loader の edbldr.empty_csv_field パラメータの詳細については、EDB ウェブサイトの*『[Oracle Developers Tools and Utilities Guide](#) のデータベース互換性*』を参照してください。

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

utl_encode.uudecode_redwood

パラメータタイプ：ブール値

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

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

FALSE のデフォルト設定に設定すると、Advanced Server の UTL_ENCODE.UUDECODE 関数は、Advanced Server の UTL_ENCODE.UUENCODE ファンクションによって作成された uuencode データをデコードできファンクション。

utl_file.umask

パラメータタイプ：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

utl_file.umask パラメータは、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または単にマスクを設定します。これは、Advanced Server UTL_FILE パッケージ内でのみ使用されます。

!!! Note * utl_file.umask パラメータは、Windows システムではサポートされていません。

utl_file.umask に指定される値は、Linux の umask コマンドに有効な 3 文字または 4 文字の 8 進数文字列です。この設定により、UTL_FILE 関数およびプロシージャによって作成されたファイルの権限が決まります。(ファイル許可と umask コマンドの使用法については、Linux または Unix システムに関する情報ソースを参照してください。)

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に対するすべての許可が拒否されることに注意してください。ユーザ enterprisedb のみがファイルに対する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

</div>

このセクションの構成パラメーターは Advanced Server のみに適用され、特定の限定された目的のためのものです。

nlslength_semantics

パラメータタイプ：列挙型

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

このパラメータは Advanced Server では効果がありません。

例、ALTER SESSION コマンドのフォームは、構文エラーをスローすることなく Advanced Server で受け入れられますが、セッション環境は変更しません。

```
ALTER SESSION SET nlslength_semantics = char;
```

!!! Note * このパラメータの設定はサーバー環境に影響を与えないため、システムビュー pg_settings には表示されません。

query_rewrite_enabled

パラメータタイプ：列挙型

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

このパラメータは Advanced Server では効果がありません。

例、次のフォームの ALTER SESSION コマンドは Advanced Server で構文エラーをスローすることなく受け入れられますが、セッション環境は変更されません。

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

!!! Note * このパラメータの設定はサーバー環境に影響を与えないため、システムビュー `pg_settings` には表示されません。

query_rewrite_integrity

パラメータタイプ：列挙型

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

このパラメータは Advanced Server では効果がありません。

例、次のフォームの `ALTER SESSION` コマンドは Advanced Server で構文エラーをスローすることなく受け入れられますが、セッション環境は変更されません。

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

!!! Note * このパラメータの設定はサーバー環境に影響を与えないため、システムビュー `pg_settings` には表示されません。

timed_statistics

パラメータタイプ：ブール値

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

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。

!!! Note * Advanced Server がインストールされている場合、`postgresql.conf` ファイルには、`timed_statistics` をオフに設定する明示的なエントリーが含まれています。このエントリーがコメントアウトされて、`timed_statistics` がデフォルトになり、構成ファイルが再ロードされると、時間指定統計収集がオンになります。

Audit Archiver

</div>

このセクションでは、Advanced Server データベース監査アーカイビング機能で使用する構成パラメーターについて説明します。

edb_audit_archiver

パラメータタイプ：列挙型

デフォルト値：false

範囲：{true | false}

効果の最小範囲：クラスター

値の変更が有効になるとき：再起動

有効化に必要な承認：EPAS サービスアカウント

データベース監査アーカイビングを有効または無効にします。

edb_audit_archiver_timeout

パラメータタイプ：整数

デフォルト値：300s

範囲：30 秒から 1 日

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：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強制します。有効なレンジは 30 秒から 1 日です。

edb_audit_archiver_filename_prefix

パラメータタイプ：文字列

デフォルト値：audit-

範囲：該当なし

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：EPAS サービスアカウント

アーカイブニーズがある監査ログファイルのファイル名前を指定します。ファイル名前は `edb_audit_filename` パラメータと一致する必要があります。 `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` が含まれる監査ファイルは、圧縮および/または期限切れの対象となります。

edb_audit_archiver_compress_time_limit

パラメータタイプ：整数

デフォルト値： -1

許容値： 0、-1、または秒単位の正の数値

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認： EPAS サービスアカウント

監査ログが圧縮に適格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いとすぐに圧縮が開始され、-1 = ログファイルのタイムリーな圧縮は行われません。

edb_audit_archiver_compress_size_limit

パラメータタイプ：整数

デフォルト値： -1

許容値： 0、-1、またはメガバイト単位の正の数値

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認： EPAS サービスアカウント

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。その後、監査ログは圧縮の対象になります。パラメータが-1 に設定されている場合、サイズごとのログファイルの圧縮は行われません。

edb_audit_archiver_compress_command

パラメータタイプ：文字列

デフォルト値： `gzip %p`

範囲：該当なし

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認： 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的なメソッドを提供します。文字列の `%p` は、アーカイブするファイルのパス名前に置き換えられます。

edb_audit_archiver_compress_suffix

パラメータタイプ：文字列

デフォルト値： `.gz`

範囲：該当なし

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：EPAS サービスアカウント

すでに圧縮されているログファイルのファイル名前を指定します。ファイル名前は `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前は `.gz` です。

edb_audit_archiver_expire_time_limit

パラメータタイプ：整数

デフォルト値： `-1`

許容値： `0`、`-1`、または秒単位の正の数値

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：EPAS サービスアカウント

監査ログが有効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い場合はすぐに有効期限が開始され、`-1` = ログファイルのタイムリーな有効期限は発生しません。

`edb_audit_archiver_expire_size_limit`

パラメータタイプ：整数

デフォルト値： `-1`

許容値： `0`、`-1`、またはメガバイト単位の正の数値

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：EPAS サービスアカウント

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。その後、監査ログは期限切れの対象になります。パラメータが-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

パラメータタイプ：文字列

デフォルト値：''

範囲：該当なし

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：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コマンドを指定します。

```
# edb_audit_archiver_sort_file
```

パラメータタイプ：文字列

デフォルト値：mtime

範囲：該当なし

効果の最小範囲：クラスター

値の変更が有効になるとき：再読み込み

有効化に必要な承認：EPAS サービスアカウント

アルファベット順または `mtime` に基づいてソートされる最も古いログファイルを識別します。`mtime` は、ファイルの変更時間に基づいてファイルを並べ替えことを示します。`alphabetic` は、ファイル名前に基づいてファイルをアルファベット順に並べ替えことを示します。

3.2 Index Advisor

</div>

インデックスアドバイザーユーティリティは、特定のワークロードでパフォーマンスを向上させるためにインデックスを作成する列を決定するのに役立ちます。インデックスアドバイザーは B-tree（単一列または複合）インデックスタイプを考慮し、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（GIN、GiST、ハッシュ）を識別しません。Index Advisor は EDB Postgres Advanced Server とともにインストールされます。

インデックスアドバイザーは Advanced Server のクエリープランナと連携し、クエリープランナ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**仮定のインデックス**を作成します。インデックスアドバイザーは、ワークロードで提供される SQL クエリを分析することにより、インデックスを識別します。

Index Advisor を使用して SQL クエリを分析するには、3 つの方法があります。

- Index Advisor ユーティリティプログラムを呼び出して、分析する SQL クエリを含むテキストファイルを提供します。インデックスアドバイザーは、推奨インデックスの 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。
- インデックスアドバイザーに分析させるクエリを EDB-PSQL コマンドラインで提供します。

Postgres Enterprise Manager クライアントから Index Advisor にアクセスします。PEM クライアント経由でアクセスすると、インデックスアドバイザーは 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のインデックス作成に関する推奨事項を提供します。PEM で SQL プロファイラーとインデックスアドバイザーを使用する方法の詳細については、次の EDB ウェブサイトから入手できる* PEM 入門ガイド*を参照してください。

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

インデックスアドバイザーは、INSERT、UPDATE、DELETE、および SELECT ステートメントでインデックスの推奨事項を作成しようとします。インデックスアドバイザーを呼び出すとき、クエリセット（SQL ファイルでコマンドを提供する場合）または EXPLAIN ステートメント（psql コマンドラインで SQL ステートメントを指定する場合）のフォームでワークロードを提供します。インデックスアドバイザーは、指定されたクエリーのクエリプランと推定実行コストを表示しますが、実際にはクエリを実行しません。

分析中に、インデックスアドバイザーは、クエリー実行コストと仮定のインデックスの有無を比較します。仮定のインデックスを使用した実行コストがそれを使用しない実行コストより小さい、両方のプランが EXPLAIN ステートメントの出力で報告され、改善を定量化するメトリックが計算され、インデックスアドバイザー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定のインデックスが見つからない場合、Index Advisor は EXPLAIN ステートメントのオリジナルのクエリプラン出力のみを表示します。

- インデックスアドバイザーは、実際にはテーブルにインデックスを作成しません。インデックスアドバイザーが提供する CREATE INDEX ステートメントを使用して、推奨インデックスをテーブルに追加します。*

Advanced Server に付属のスクリプトは、インデックスアドバイザーが分析によって生成されたインデックス付けの推奨事項を保存するテーブルを作成します。また、スクリプトはファンクションの作成とテーブルのビューを作成して、結果の取得と解釈を簡素化します。

スクリプトの実行を中止することを選択した場合、インデックスアドバイザーは、インデックスアドバイザーセッションの間のみ利用可能な一時テーブルに推奨事項をログします。

3.2.1 Index Advisor Components

</ div>

インデックスアドバイザーの共有ライブラリは、クエリープランナと対話し make、インデックス作成の推奨事項を作成します。Windows では、Advanced Server インストーラは Advanced Server ホームディレクトリサブディレクトリに次の共有ライブラリを作成します。Linux の場合、edb-asxx-server-indexadvisor RPM パッケージをインストールします。xx は Advanced Server のバージョン番号です。

Linux の場合：

index_advisor.so

Windows :

`index_advisor.dll`

`libdir` ディレクトリ内のライブラリは、スーパーユーザのみがロードできることにノートしてください。データベース管理者は、インデックスアドバイザーファイルを `libdir` ディレクトリから `libdir/plugins` ディレクトリ (Advanced Server ホームディレクトリの下) に手動でコピーすることにより、非スーパーユーザーがインデックスアドバイザーを使用できるようにすることができます。プラグインへのアクセスを許可するのは、信頼できる非スーパーユーザーのみにしてください。これは、稼動環境では安全ではありません。

インストーラは、インデックスアドバイザー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`

`pg_advise_index` は、SQL クエリを含むユーザー提供の入力ファイルを読み取り、Index Advisor が推奨するインデックスの作成に使用できる `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です。`pg_advise_index` プログラムは、Advanced Server ホームディレクトリサブディレクトリにあります。

`index_advisor.sql`

`index_advisor.sql` は、ログテーブルからの推奨事項のレポートを容易にするためのファンクションとビューとともに永続的なインデックスアドバイザーログテーブルを作成するスクリプトです。スクリプトは、Advanced Server ディレクトリの `share/contrib` サブディレクトリにあります。

`index_advisor.sql` スクリプトは、`index_advisor_log` テーブル、`show_index_recommendations()` ファンクション、および `index_recommendations` ビューを作成します。これらのデータベースオブジェクトは、インデックスアドバイザーを呼び出すロールの検索パスにアクセス可能なスキーマに作成する必要があります。

`index_advisor_log`

インデックスアドバイザーは、インデックスの推奨事項を `index_advisor_log` テーブルに記録します。インデックスアドバイザーが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、インデックスアドバイザーは同じ名前の一時テーブルにインデックスの推奨事項を保存します。一時テーブルは、現在のセッションの間だけ存在します。

`show_index_recommendations()`

`show_index_recommendations()` は、特定の Index Advisor セッション中に作成された推奨事項を解釈して表示する PL/pgSQL ファンクションです (バックエンドプロセス ID で識別されます)。

`index_recommendations`

インデックスアドバイザーは、クエリー分析中に `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()` 関数と同じフォーマットで出力を生成しますが、`show_index_recommendations()` ファンクション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ファンクションが、保存されたすべてのセッションに対するインデックスアドバイザーの推奨事項が含まれます。

3.2.2 Index Advisor Configuration

</ div>

インデックスアドバイザーは、現在のセッションの間のみ利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。マルチプルのセッションの結果を保存するには、`index_advisor_log` テーブルを作成する必要があります（Advanced Server が Index Advisor の推奨事項を保存する場所）。`index_advisor_log` テーブルを作成するには、`index_advisor.sql` スクリプトを実行する必要があります。

インデックスアドバイザーのテーブル、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、インデックスアドバイザーを呼び出す（および結果セットをクエリーする）すべてのユーザーがスキーマに対する `USAGE` 特権を持っている必要があることに注意してください。スキーマは、インデックスアドバイザーと対話するすべてのユーザーの検索パスに存在する必要があります。

1. 選択したスキーマを `search_path` パラメータのスタートに配置します。例、検索パスが現在の場合：

```
* search_path=public, accounting
```

- インデックスアドバイザーオブジェクトを `advisor` 名前付けのスキーマに作成するには、次のコマンドを使用します。

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

2. `index_advisor.sql` スクリプトを実行して、データベースオブジェクトを作成します。psql クライアントを実行している場合、次のコマンドを使用できます。

```
* \i full_pathname/index_advisor.sql
```

- `full_pathname` の代わりに `index_advisor.sql` スクリプトへのパス名を指定します。

3. `index_advisor_log` テーブル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、またはこれらのデータベースオブジェクトの所有者である場合、この手順は不要です。

- `-index_advisor_log` 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 を起動できるようにします。`-index_advisor_log` テーブルに対する `DELETE` 特権を付与して、指定し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。`-index_recommendations` ビューで `SELECT` 権限を付与します。

次の例は、`ia` 名前付けのスキーマでのインデックスアドバイザーデータベースオブジェクトの作成を示しています。このオブジェクトには、ユーザー名前 `ia_user` のインデックスアドバイザーユーザーがアクセスできます。

```
$ edb-psql -d edb -U enterprisedb
psql.bin (14.0.0, server 14.0.0)
Type "help" for help.

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

(次のページに続く)

(前のページからの続き)

```

SET
edb=# \i /usr/edb/as14/share/contrib/index_advisor.sql
CREATE TABLE
CREATE INDEX
CREATE INDEX
CREATE FUNCTION
CREATE FUNCTION
CREATE VIEW
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

```

インデックスアドバイザーを使用する場合、指定されたスキーマ (ia) を ia_user の search_path パラメータに含める必要があります。

3.2.3 Using Index Advisor

</ div>

インデックスアドバイザーを呼び出すときは、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、クエリー（コマンドラインで指定）、またはクエリーのセットを含むファイル（pg_advise_index() ファンクションによって実行）です。ワークロードの分析後、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。インデックスアドバイザーによって生成されたインデックスの推奨事項を確認し、インデックスアドバイザーによって生成された CREATE INDEX ステートメントを使用して、推奨インデックスを作成できます。

!!! Note * 読み取り専用トランザクションでインデックスアドバイザーを実行しないでください。

次の例では、スーパーユーザ enterprisedb がインデックスアドバイザーユーザであり、インデックスアドバイザーデータベースオブジェクトがスーパーユーザ enterprisedb の search_path のスキーマに作成されていることを前提としています。

次のセクションの例では、次のステートメントで作成されたテーブルを使用しています。

```

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。

```

a      | b
-----+-----
* 0    | 99999

```

(次のページに続く)

(前のページからの続き)

```

* 1 | 99998
* 2 | 99997
* 3 | 99996
*   .
*   .
*   .
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々の行にありますが、各クエリー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は、EXPLAIN キーワードで始まってはなりません。

次の例は、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;

```

以下のコードサンプルに示すように、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 */

```

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

-s

-s は、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です。インデックスアドバイザーが結果セットを結果ない場合、-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じて、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);
```

ファイル内の `CREATE INDEX` ステートメントを使用して `psql` コマンドラインで推奨インデックスを作成するか、`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
```

psql コマンドラインでのインデックスアドバイザーの使用

インデックスアドバイザーを使用して、`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。次の手順では、Index Advisor プラグインのローディングと Index Advisor の使用について詳しく説明します。

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、インデックスアドバイザーに分析させたい各 SQL コマンドを呼び出します。インデックスアドバイザーは、クエリの推奨事項を `index_advisor_log` テーブルに格納します。`index_advisor_log` テーブルがユーザーの `search_path` に存在しない場合、同じ名前の一時テーブルが作成されます。この一時テーブルは、ユーザーのセッションの間のみ存在します。

Index Advisor プラグインをローディングした後、Index Advisor はすべての SQL ステートメントを分析し、セッション中のインデックス作成に関する推奨事項をログします。

実際にクエリーを実行せずに、インデックスアドバイザーでクエリーを分析（およびインデックス `make` の推奨事項）したい場合は、SQL ステートメントの前にはじめにキーワードを付けます。

ステートメントの前にはじめに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、後で確認するためにインデックスの推奨事項を `index_advisor_log` テーブルに書き込みます。

次の例では、クエリーが推奨仮定のインデックスを使用している場合、`EXPLAIN` ステートメントは通常のクエリプランを表示し、その後に同じクエリーのクエリプランを表示します。

```

edb=# EXPLAIN SELECT * FROM t WHERE a < 10000;
*
          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 プラグインをローディングした後、`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` パラメータを使用して、psql セッションを中断することなく、インデックスアドバイザーを一時的に無効にできます。

```

edb=# SET index_advisor.enabled TO off;
SET

```

インデックスアドバイザーを有効にするには、パラメータを `on` に設定します。

```

edb=# SET index_advisor.enabled TO on;
SET

```

3.2.4 Reviewing the Index Advisor Recommendations

</ div>

Index Advisor によって生成されたインデックスの推奨事項を確認するには、いくつかの方法があります。次のことができます。

- `index_advisor_log` テーブルをクエリします。
- `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 ()` ファンクションの使用

`show_index_recommendations()` ファンクションを使用してインデックスアドバイザー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 );
```

ここで、`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結果れます。

次のコードは、結果セットのフラグメントの例を示し行います。

```
edb=# SELECT show_index_recommendations(null);
*
          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。結果セットの行には以下が表示されます。

- 推奨インデックスの作成に必要なコマンド。
- インデックスの推定最大サイズ。
- インデックスを使用して計算された利点。
- 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、すべてのインデックスアドバイザー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

インデックスアドバイザーは、インデックス作成の推奨事項を `index_advisor_log` 名前付けのテーブルに格納します。`index_advisor_log` テーブルの行には、クエリの実行コストを削減するために仮定のインデックスを推奨できると Index Advisor が判断したクエリーの結果が含まれるクエリー。

Column	Type	Description
<i>reloid</i>	<i>oid</i>	インデックスのベーステーブルの OID
<i>relname</i>	<i>name</i>	インデックスのベーステーブルの名前
<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>	このクエリーのインデックスの計算された利益
<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>	この推奨事項を生成するプロセスのプロセス ID
<i>timestamp</i>	<i>timestamp</i>	推奨が生成された日付/時刻
		NA

`psql` コマンドラインで `index_advisor_log` テーブルをクエリーできます。次の例は、2つの Index Advisor セッションから生じる `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、異なる `backend_pid` 値によって（下の表で）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;
 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)
```

インデックスアドバイザーは、`pg_advise_index` ユーティリティによって実行される次の2つのクエリを分析した後、テーブルに最初の2行を追加しました。

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

列 `backend_pid` の 3442 の値は、これらの結果をプロセス ID 3442 のセッションからのものとして識別します。

最初の行の列 `attrs` の 1 の値は、仮定のインデックスがテーブルの最初の列（テーブル `t` の列 `a`）にあることを示しています。

2 行の `attrs` 列の 2 の値は、仮定のインデックスがテーブルの 2 列目（テーブル `t` の列 `b`）にあることを示しています。

インデックスアドバイザーは、次の 2 つのクエリ（`psql` コマンドラインで実行）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;
*
          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_log` テーブルの `benefit` 列の値は、次の式を使用して計算されます。

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

`index_advisor_log` テーブルの最後の行（例に示す）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され、比較されます。

$$\text{benefit} = (\text{Seq Scan on t cost}) - (\text{Index Scan using <hypothetical-index>})$$

また、表に利点が追加されます。

```
benefit = 1693.00 - 8.28
```

```
benefit = 1684.72
```

行に格納された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 ビューのクエリ

`index_recommendations` ビューには、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インデックスを作成するための計算されたメトリックと `CREATE INDEX` ステートメントが含まれています。以下に示すように、`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;
```

前のセクション (Querying the `index_advisor_log` Table) に示されている例を使用すると、`index_recommendations` ビューには次が表示されます。

```
edb=# SELECT * FROM index_recommendations;
 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)
```

各セッション内で、同じ推奨インデックスの恩恵を受けるすべてのクエリの結果が組み合わされて、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` 名前付け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、例、`backend_pid` が 3506 であるプロセスからの次のクエリー結果を使用します。

```
reloid | relname | attrs | benefit | index_size | backend_pid | timestamp
-----+-----+-----+-----+-----+-----+-----
-----
```

(次のページに続く)

(前のページからの続き)

```

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。

```

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

```

そして

```

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派生し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。ゲイン値が大きいほど、インデックスから得られるコスト効果は、インデックスのディスクスペース消費の可能性と比較検討されます。

3.2.5 Index Advisor Limitations

</ div>

Index Advisor は Index Only スキャンを考慮しません。推奨を行う際にインデックススキャンを考慮します。

インデックスアドバイザーは、WHERE 句で見つかった計算を無視します。事実上、推奨事項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。フィールドはシンプル列名前になります。

仮定のインデックスを推奨する場合、インデックスアドバイザーは継承を考慮しません。クエリーが親テーブルを参照する場合、インデックスアドバイザーは子テーブルのインデックスを推奨 make ません。

index_advisor_log テーブル、またはインデックス作成の推奨事項が作成されて index_advisor_log テーブルに格納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、index_advisor_log テーブルと index_advisor_log テーブルの行によって被参照される復元テーブルとの間に「リンク切れ」が発生する場合がありますオブジェクト識別子 (OID) の変更。

バックアップの前に作成さ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 を被参照テーブルの新しい 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

</div>

非効率的な SQL コードは、データベースパフォーマンスの問題の主な原因ではないにしても、その 1 つです。データベース管理者および開発者にとっての課題は、ラージ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 Profiler の特定の機能と利点は次のとおりです。

- **オンデマンドトレース。** パラメータを手動で設定し、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。** 都合の悪い時間に、トレースパラメータを指定して、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。** データベースおよびユーザごとに SQL キャプチャを選択的にフィルター処理するか、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。** グラフィカルテーブルにより、クエリを期間またはステートメントごとにすばやく並べ替えたり、フィルタリングしたりできます。グラフィカルまたはテキストベースの EXPLAIN プランは、クエリーと結合をレイアウトします。
- **インデックスアドバイザーの統合遅いクエリ**を見つけて最適化したら、パフォーマンスをさらに向上させるために、インデックスアドバイザーに基礎となるテーブルインデックスの作成を推奨させることもできます。

次に、インストールプロセスについて説明します。

ステップ 1: SQL プロファイラーのインストール

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

ステップ 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` パラメータファイルを変更して、`shared_preload_libraries` 設定パラメータに SQL プロファイラーライブラリを含めます。

Linux インストールの場合、パラメータ値には以下を含める必要があります。

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

Windows では、パラメータ値には次のものが含まれます。

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

ステップ 3: SQL プロファイラーが使用する関数を作成する

SQL Profiler インストールプログラムは、次の場所に SQL スクリプト (`sql-profiler.sql` 名前付け) を配置します。

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フェースを使用して、プロファイリングするサーバー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。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付けは `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付けは `postgres` です。

次のコマンドは、Linux システムで `psql` コマンドラインを使用して `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
```

ステップ 4: サーバーを停止してリスタートし、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 ウェブサイトにアクセスしてください。

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

トラブルシューティング

(SQL プロファイラーの新しいバージョンへのアップグレードを実行した後に) 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、既存のクエリーセットを新しいクエリーセットに置き換える必要があります。まず、`uninstall-sql-profiler.sql` スクリプトを呼び出して SQL プロファイラーをアンインストールし、次に `sql-profiler.sql` スクリプトを呼び出して SQL プロファイラーを再インストールします。

3.4 pgsnmpd

</ div>

pgsnmpd は、Linux システム上の Advanced Server の現在の状態に関する階層情報を結果ことのできる SNMP エージェントです。pgsnmpd は、xx RPM が Advanced Server バージョン番号である 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。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。

Advanced Server のインストール後、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 のディストリビューションの文書を参照してください。

以下の例は、読み取り専用アクセスを実装する pgsnmpd の最も単純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、次を参照してください。

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

3.4.1 pgsnmpd の構成

pgsnmpd 構成ファイルの名前付けは 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) を参照してください。

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ます。

net-snmp は、以下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込むように求められる場合があります。プロンプト **none** と入力して、新しい構成ファイルを生成します（既存の構成ファイルに基づいていません）。

snmpconf はメニュー方式のウィザードです。メニューアイテム 1: **snmpd.conf** を選択して、構成ウィザードをスタートします。snmpconf によって表示される各トップレベルメニューオプションを選択すると、ウィザードはシリーズの質問を順に実行し、構成ファイルの構築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 名前付けの構成ファイルを生成します。ファイルを次の場所にコピーします。

```
/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 をリッスンします。リスナポートが既に別のサービスで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することにより、代替リスナポートを指定できます。

```
agentaddress $host_address:2000
```

この例では、pgsnmpd が UDP ポート 2000 を監視するように指示します。\$host_address はサーバーの IP アドレス (127.0.0.1 など) です。

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

Advanced Server のインスタンスが稼働中であることを確認します (pgsnmpd はこのサーバーに接続します)。次のフォームのコマンドで 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継ぎます。

```
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 Advanced Server のインストールディレクトリを指定します。

-b オプションを含めて、pgsnmpd をバックグラウンドで実行することを指定します。

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、Advanced Server のインストールの接続情報 (libpq 接続文字列のフォーム) 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して他の pgsnmpd コマンドラインオプションをビューする場合は、`--help` オプションを含めます。

```
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 からの情報の要求

`net-snmp` コマンドを使用して、pgsnmpd サービスをクエリーできます。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、`snmpgetnext` クライアントに要求を SNMP バージョン 2c フォーマットで送信するように指示します。

`-c public` はコミュニティ名前を指定します。

`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`、インクリメント、`.1.3` など）。

特定のオブジェクトのクエリー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モニタできます。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）のフォームで記述します。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

</ div>

Advanced Server を使用すると、データベースおよびセキュリティの管理者、監査者、およびオペレーターは、* EDB Audit Logging *機能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。監査ログは、次のような情報をレコードするように構成できます。

- ロールが Advanced Server データベースへの接続を確立するとき Advanced Server への接続時にロールが作成、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。

3.5.1 Audit Logging Configuration Parameters

</ div>

以下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、設定パラメータの変更をすぐに有効にするか、構成を再ロードするニーズがあるか、またはデータベースサーバを再起動ニーズがあるかを判断します。

`edb_audit`

データベース監査を有効または無効にします。値 `xml` または `csv` は、データベース監査を有効にします。これらの値は、監査情報がキャプチャされ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、これもデフォルト。

`edb_audit_archiver`

データベース監査アーカイビングを有効または無効にします。

`edb_audit_archiver_timeout`

このパラメーターを使用して、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強制します。有効なレンジは 30 秒から 1 日です。

`edb_audit_archiver_filename_prefix`

アーカイブニーズがある監査ログファイルのファイル名前を指定します。ファイル名前は `edb_audit_filename` パラメータと一致する必要があります。`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` が含まれる監査ファイルは、圧縮および/または期限切れの対象となります。

`edb_audit_archiver_compress_time_limit`

監査ログが圧縮に適格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いとすぐに圧縮が開始され、`-1` =ログファイルのタイムリーな圧縮は行われません。

edb_audit_archiver_compress_size_limit

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。その後、監査ログは圧縮の対象になります。パラメータが-1に設定されている場合、サイズごとのログファイルの圧縮は行われません。

edb_audit_archiver_compress_command

監査ログファイルの圧縮を実行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的なメソッドを提供します。文字列の**%p** は、アーカイブするファイルのパス名前に置き換えられます。

edb_audit_archiver_compress_suffix

すでに圧縮されているログファイルのファイル名前を指定します。ファイル名前は **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前は **.gz** です。

edb_audit_archiver_expire_time_limit

監査ログが有効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い場合はすぐに有効期限が開始され、-1 = ログファイルのタイムリーな有効期限は発生しません。

edb_audit_archiver_expire_size_limit

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。その後、監査ログは期限切れの対象になります。パラメータが-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

削除する前に期限切れの監査ログファイルで実行コマンドを指定します。

edb_audit_archiver_sort_file

アルファベット順または **mtime** に基づいてソートされる最も古いログファイルを識別します。**mtime** は、ファイルの変更時間に基づいてファイルを並べ替えことを示します。**alphabetic** は、ファイル名前に基づいてファイルをアルファベット順に並べ替えことを示します。

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、データフォルダに対して相対パスまたは絶対パスにすることができます。デフォルトは **PGDATA/edb_audit** ディレクトリです。

edb_audit_filename

監査情報が保存される監査ファイルのファイル名前を指定します。デフォルトのファイル名前は **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の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ト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。ローテーションを無効にするには、値を **none** に設定します。ファイルを毎日ローテートするには、**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トするには、値を目的の曜日に設定します。**every** はデフォルト値です。

edb_audit_rotation_size

ファイルのローテーションが強制されるとき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0メガバイトです。パラメータがコメント化されているか、0に設定されている場合、サイズごとのファイルのローテーションは発生しません。

`edb_audit_rotation_seconds`

新しい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。この機能を無効にするには、このパラメータをデフォルトの0に設定します。

`edb_audit_connect`

ユーザーによるデータベースコネクション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。これはデフォルト。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`

接続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。これはデフォルト。

`edb_audit_statement`

この設定パラメータは、SQL ステートメントのさまざまなカテゴリの監査を指定するために使用されます。`none`、`dml`、`insert`、`update`、`delete`、`truncate`、`select`、`error`、`rollback`、`ddl`、`create`、`drop`、`alter`、`grant`、`revoke`、`set`、`all`、及び `{ select | update | delete | insert }@groupname`：以下の値の様々な組み合わせを指定することができます。デフォルトは `ddl` および `error` です。このパラメータの設定に関する情報については、[Selecting SQL Statements to Audit](#) を参照してください。

`edb_audit_tag`

この設定パラメータを使用して、各エントリー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 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ギン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` 設定パラメータによって制御できます。`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、ログメッセージは一括処理ごとに1回、処理された行数とともに記録されます。さらに、期間はバルク処理ごとに1回発行されます。デフォルトは `false` に設定されています。

`edb_audit_destination`

監査ログ情報を、`edb_audit_directory` パラメータで指定されたディレクトリに記録するか、`syslog` プロセスで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。デフォルト設定の `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 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。**注：**最近の Linux バージョンでは、`syslog` は `rsyslog` に置き換えられており、構成ファイルは `/etc/rsyslog.conf` にあります。

!!! Note * Advanced Server では、管理特権に関連付けられた管理ユーザーが、任意のユーザ、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。

次のセクションでは、`edb_audit_statement` パラメータを使用して監査する特定の SQL ステートメントの選択について説明します。

3.5.2 Selecting SQL Statements to Audit

</div>

`edb_audit_statement` では、監査する SQL ステートメントを制御するために、1 つ以上のコンマ区切り値を `inclusion` 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]...'
```

コンマで区切られた値には、コンマに続くスペース文字が含まれるか、省略されます。値は、小文字または大文字の任意の組み合わせで指定できます。

基本的なパラメータ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の監査とロギングを行います。
- **none** –すべての監査とロギングを無効にします。**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**)、および **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** ステートメントの監査結果。
- **set** –**SET** ステートメントの監査結果。
- **rollback** –**ROLLBACK** ステートメントの監査結果。
- **error** –発生したすべてのエラーメッセージをロギングに記録します。**error** 値が含まれていない限り、**all** が使用されている場合を除き、他の先行パラメータ値に関連する SQL ステートメントで発生したエラーに関するエラーメッセージは記録されません。
- **{ select | update | delete | insert }@groupname** –特定のテーブルの特定のデータ操作言語 (DML) ステートメント (**SELECT**、**UPDATE**、**DELETE**、および **INSERT** を含む) のオブジェクトを選択的に監査します。詳細については、[Object Auditing](#) を参照してください。

Data Definition Language and Data Control Language Statements は、監査のために特定の 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値を記述します。

Data Manipulation Language Statements は、監査のために特定の DML ステートメントを選択するための追加のパラメータ値を記述します。

サポートされていない値が `edb_audit_statement` パラメータに含まれている場合、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
```

以下のセクションでは、SQL 言語タイプ DDL、DCL、および DML の値について説明します。

データ定義言語とデータ制御言語のステートメント

このセクションでは、DDL および DCL ステートメントを監査するために `edb_audit_statement` パラメータに含めることができる値について説明します。

次の一般規則が適用されます。

- `edb_audit_statement` パラメータに `ddl` または `all` が含まれている場合、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` も監査されます。
- `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` パラメータ内に値の組み合わせを含めることにより、特定のタイプの DDL および DCL ステートメントを監査用を選択できます。

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 コマンドで使用されるオブジェクトタイプの説明は、次の場所にある PostgreSQL コア文書に記載されています。

< <https://www.postgresql.org/docs/current/ 静的/sql-commands.html> >

パラメータ値から `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` のいずれか) が監査されます。

DCL ステートメントの `edb_audit_statement` パラメータ値を指定するには、次の構文を使用します。

```
{ grant | revoke }
```

以下は、DDL と DCL の例です。

例 1

以下は、`edb_audit_connect` および `edb_audit_statement` が次のデフォルト以外の値で設定されている例です。

```
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;
   edb_audit_connect
-----
all
(1 row)

edb=# SHOW edb_audit_statement;
   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;
ALTER DATABASE
```

(次のページに続く)

(前のページからの続き)

```

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

```

結果の監査ログファイルには、次のものが含まれます。

各監査ログエントリはマルチプルの行に分割されて表示され、結果の外観をより明確にするために、監査ログエントリの間空白行が挿入されています。

```

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,00000,"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,00000,"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,00000,
"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,00000,"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,00000,"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,00000,"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。エラーは edb_audit_statement パラメータ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、または drop など、正常に処理された DROP ステートメントの監査につながる edb_audit_statement 設定がないため、DROP TABLE department ステートメントは監査ログにないことに注意してください。

例 2

以下は、`edb_audit_connect` および `edb_audit_statement` が次のデフォルト以外の値で設定されている例です。

```
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;
   edb_audit_connect
-----
all
(1 row)

auditdb=# SHOW edb_audit_statement;
   *               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

```

結果の監査ログファイルには、次のものが含まれます。

各監査ログエントリはマルチプルの行に分割されて表示され、結果の外観をより明確にするために、監査ログエントリの上に空白行が挿入されています。

```

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、または all のいずれの値も edb_audit_statement パラメータ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ないことに注意してください。

CREATE SEQUENCE および GRANT ステートメントは、それらの値が edb_audit_statement パラメータに含まれているため、監査されます。

例 3

以下は、edb_audit_connect および edb_audit_statement が次のデフォルト以外の値で設定されている例です。

```
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は、完全に監査できます。管理ユーザは、ADMINUSER としてデータベース auditdb に接続されます。次の ALTER USER コマンドは、ADMINUSER を監査することを指定します。

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

edb_audit_statement パラメータを all に設定すると、管理ユーザのすべての 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;
 edb_audit_connect
-----
all
(1 row)

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

(次のページに続く)

(前のページからの続き)

```

CREATE TABLE
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=> UPDATE dept SET loc = 'BEDFORD' WHERE deptno = 20;
UPDATE 1
auditdb=> SELECT * FROM dept;
 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;
 deptno |  dname   |   loc
-----+-----+-----
*      30 | SALES      | CHICAGO
*      20 | RESEARCH   | BEDFORD
(2 rows)

```

結果の監査ログファイルには、次のものが含まれます。

各監査ログエントリはマルチプルの行に分割されて表示され、結果の外観をより明確にするために、監査ログエントリの上に空白行が挿入されています。

```

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"

```

(次のページに続く)

(前のページからの続き)

```

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;
",,,,,,,,,,"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;

```

(次のページに続く)

(前のページからの続き)

```

,,,,,,,,,"psql","client backend",,"DELETE","",,"delete"

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"

```

例 4

以下は、`edb_audit_connect` および `edb_audit_statement` が次のデフォルト以外の値で設定されている例です。

```

logging_collector = 'on'
edb_audit_connect = 'all'
edb_audit_statement = 'all'

```

ユーザ `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";
```

!!! Note * データベース管理者は、`ALTER USER` コマンドを指定し、`edb_audit_statement` パラメータを任意の値に設定することにより、特定のユーザが SQL ステートメントを監査できるようにすることができます。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;
edb_audit_connect
-----
all
(1 row)

```

(次のページに続く)

(前のページからの続き)

```

auditdb=# SHOW edb_audit_statement;
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);

```

結果の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 パラメータを none に設定しても、carol の SQL ステートメントの監査は許可されないため、監査ログは生成されません。

各監査ログエントリはマルチプルの行に分割されて表示され、結果の外観をより明確にするために、監査ログエントリの間空白行が挿入されています。

```

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,000000,"connection_
→authorized: user=carol
database=auditdb",,,,,,,,,,"client backend",,,,,,"connect"

```

データ操作言語ステートメント

このセクションでは、edb_audit_statement パラメータに含めて DML ステートメントを監査できる値について説明します。

次の一般規則が適用されます。

- edb_audit_statement パラメータに dml または all が含まれている場合、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 パラメータ内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タイプの 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 パラメータ値を指定します。

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

例

以下は、`edb_audit_connect` および `edb_audit_statement` が次のデフォルト以外の値で設定されている例です。

```
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。すべてのエラーも監査ログに含まれます (UPDATE および DELETE コマンドに関連しないエラーも含む)。

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;
   edb_audit_connect
-----
all
(1 row)

auditdb=# SHOW edb_audit_statement;
   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);
```

(次のページに続く)

(前のページからの続き)

```

INSERT 0 1
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;
 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

```

結果の監査ログファイルには、次のものが含まれます。

各監査ログエントリはマルチプルの行に分割されて表示され、結果の外観をより明確にするために、監査ログエントリの間に空白行が挿入されています。

```

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.

```

(次のページに続く)

(前のページからの続き)

```

1e21c,3,"idle",2020-05-25 15:17:16 IST,5/16,0,AUDIT,00000,"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.
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 のいずれの値も edb_audit_statement パラメータ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ないことに注意してください。

select も all も edb_audit_statement パラメータに含まれていないため、SELECT * FROM dept ステートメントも監査されません。

error は edb_audit_statement パラメータで指定されますが、truncate 値では指定されないため、TRUNCATE employee ステートメントのエラーは監査ファイルに記録されますが、成功した TRUNCATE emp ステートメントでは記録されません。

3.5.3 Enabling Audit Logging

</div>

次の手順では、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、およびエラーになったステートメントをログに記録するように Advanced Server を構成する方法について説明します。

1. edb_audit パラメータを xml または csv.2 に設定して、監査を有効にします。パラメータ 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 edb_audit_connect を all.4 に設定します。すべての切断を監査するには、パラメータ edb_audit_disconnect を all.5 に設定します。DDL、DCL、DML およびその他の文を監査するには、[Selecting SQL Statements to Audit](#) の指示に従ってパラメータ edb_audit_statement を設定します。

構成ファイルの edb_audit_statement パラメータの設定は、データベースクラスタ全体に影響します。

edb_audit_statement パラメータによって制御されるように監査されるステートメントのタイプは、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応じてさらに絞り込むことができます。

- edb_audit_statement パラメータは、ALTER DATABASE dbname SET edb_audit_statement コマンドを使用して、指定されたデータベースの属性として設定できます。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内の edb_audit_statement パラメータをオーバーライドします。
- ALTER ROLE rolename SET edb_audit_statement パラメータを使用して、edb_audit_statement パラメータを指定されたロールの属性として設定できます。この設定は、構成ファイルの edb_audit_statement パラメータと、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

に前述の 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。

- edb_audit_statement コマンドで指定されたデータベースを使用する場合、edb_audit_statement パラメータを指定されたロールの属性として設定できます。この設定は、構成ファイルの edb_audit_statement パラメータ、前述の 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、および前述の IN DATABASE 節なしで ALTER ROLE コマンドを使用してロールに割り当てられた設定をオーバーライドします。

以下は、この手法の例です。

データベースクラスタは、postgresql.conf ファイルに示されているように、edb_audit_statement が all に設定されて確立されます。

```
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
```

データベースとロールは、edb_audit_statement パラメータの次の設定で確立されます。

- ddl、insert、update、および delete を含むデータベース auditdb
- select、truncate、および set を含むロール admin
- create table、insert、および update を使用したデータベース auditdb のロール admin

データベースとロールの作成と変更は、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_statement;
  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
edb=# ALTER ROLE admin SET edb_audit_statement TO 'select, truncate';
ALTER ROLE
```

(次のページに続く)

(前のページからの続き)

```
edb=# ALTER ROLE admin IN DATABASE auditdb SET edb_audit_statement TO 'create table,↵
↵insert, update';
ALTER ROLE
```

以下に、3つのケースで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: enterprisedb ロール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;
      *      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
 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には、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が表示されます。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マルチプルの行に分割されて表示され、結果の外観をより明確にするために、監査ログエントリの間に空白行が挿入されています。

```
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"
```

(次のページに続く)

(前のページからの続き)

```

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;
   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;
 f1 | f2
----+-----
  * 1 | Row 1
(1 row)

edb=# TRUNCATE edb_tbl;
TRUNCATE TABLE

```

監査ログファイルの継続は、次のようになりました。2 番目のケースを表す最後の 2 つのエントリは、`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を示しています。`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;
*      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

(次のページに続く)

(前のページからの続き)

```
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継続は、次のようになりました。3 番目のケースを表す最後から 2 番目のエントリは、CREATE TABLE audit_tbl_2 および INSERT INTO audit_tbl_2 ステートメントのみを示しています。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"

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]",5ecbb370.1dced,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]",5ecbb370.1dced,3,
```

(次のページに続く)

(前のページからの続き)

```
"idle",2020-05-25 17:30:48 IST,5/12,0,AUDIT,00000,"statement: INSERT INTO
audit_tbl_2 VALUES (1, 'Row 1');",,,,,,,,,,"psql","client backend","INSERT","", "insert"
```

3.5.4 Audit Log File

</div>

監査ログファイルは、`edb_audit` 設定パラメータの設定に応じて、CSV 形式または XML フォーマットで生成できます。

監査ログの情報は、PostgreSQL コア文書の「エラー報告とログ」セクションの「CSV 形式のログ出力の使用」セクション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。

< <https://www.postgresql.org/docs/current/ 静的/runtime-config-logging.html> >

次の表に、CSV 監査ログフォーマットで表示されるオーダーでフィールドを示します。テーブルには次の情報が含まれます。

- **** Field.** ******以前被参照した PostgreSQL 文書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 フォーマットの場合、XML 要素の名前とその属性（使用する場合）、値を参照します。
- **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。

ロギングの値を持たないフィールドは、CSV フォーマットで連続したコンマ (,) 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>	ステートメントの日付/時刻をログに記録します。
<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。
<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。
<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</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>	処理ステータス。
<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。

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>virtual_transaction_id</i>	<i>event/virtual_transaction_id</i>	<i>integer</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>integer</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、監査対象ステートメントの場合は `AUDIT`、結果のエラーメッセージの場合は `ERROR` 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 ログフォーマットの値が 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>	実行が試行された 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ー（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ー。（エラーの場合のみ）
<i>query_pos</i>	<i>query_pos</i>	<i>integer</i>	その中のエラー位置の文字数。（エラーの場合のみ）
<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 <code>verbose</code> に設定されている場合、 <i>location</i> フィールドにデータが入力されます。
<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>	`backend_type` は、`pg_stat_activity.backend_type` が表示するものに対応し、列として csv ログに追加されます。
<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 `audit_tag` パラメータで指定された値。
<i>type</i>	<i>event/type</i>	<i>text</i>	ログ内のメッセージを識別するための監査 `event_type` を決定します。
			NA

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'
```

(次のページに続く)

(前のページからの続き)

```
edb_audit_disconnect = 'all'
edb_audit_statement = 'ddl, dml, select, error'
edb_audit_tag = 'edbaudit'
```

XML フォーマットの生成時に、`edb_audit parameter` は `xml` に変更されます。

監査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,
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;
 deptno |   dname   |  loc
-----+-----+-----
*      10 | ACCOUNTING | BOSTON
(1 row)

edb=# \q
```

**** CSV 監査ログファイル****

以下は、監査ログファイルの CSV フォーマットです。

各監査ログエントリはマルチプルの行に分割されて表示され、結果の外観をより明確にするために、監査ログエントリの上に空白行が挿入されています。

```

2020-05-22 16:53:37.817 IST,"enterprisedb","edb",55279,"[local]",5ec7b639.
d7ef,1,"authentication",2020-05-22 16:53:37 IST,4/21,0,AUDIT,000000,
"connection authorized:user=enterprisedb database=edb",,,,,,,,,,"client
backend",,,,,,"edbaudit","connect"

2020-05-22 16:53:42.279 IST,"enterprisedb","edb",55279,"[local]",5ec7b639.
d7ef,2,"idle",2020-05-22 16:53:37 IST,4/23,0,AUDIT,000000,"statement: CREATE
SCHEMA edb;",,,,,,,,,,"psql","client backend","CREATE SCHEMA","edbaudit",
"create"

2020-05-22 16:54:07.896 IST,"enterprisedb","edb",55279,"[local]",5ec7b639.
d7ef,3,"idle",2020-05-22 16:53:37 IST,4/25,0,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","edbaudit","create"

2020-05-22 16:54:20.500 IST,"enterprisedb","edb",55279,"[local]",5ec7b639.
d7ef,4,"idle",2020-05-22 16:53:37 IST,4/26,0,AUDIT,000000,"statement: INSERT
INTO dept VALUES(10,'ACCOUNTING','NEW YORK');",,,,,,,,,,"psql","client backend
","INSERT","edbaudit","insert"

2020-05-22 16:54:34.821 IST,"enterprisedb","edb",55279,"[local]",5ec7b639.
d7ef,5,"idle",2020-05-22 16:53:37 IST,4/27,0,AUDIT,000000,"statement: UPDATE
department SET loc = 'BOSTON' WHERE deptno = 10;",,,,,,,,,,"psql","client
backend","UPDATE","edbaudit","update"

2020-05-22 16:54:34.821 IST,"enterprisedb","edb",55279,"[local]",5ec7b639.
d7ef,6,"UPDATE",2020-05-22 16:53:37 IST,4/27,0,ERROR,42P01,"relation "
"department" does not exist",,,,,,"UPDATE department SET loc = 'BOSTON'
WHERE deptno = 10;",8,,,"psql","client backend","UPDATE","edbaudit","error"

2020-05-22 16:54:51.308 IST,"enterprisedb","edb",55279,"[local]",5ec7b639.
d7ef,7,"idle",2020-05-22 16:53:37 IST,4/28,0,AUDIT,000000,"statement: UPDATE
dept SET loc = 'BOSTON' WHERE deptno = 10;",,,,,,,,,,"psql","client backend",
"UPDATE","edbaudit","update"

2020-05-22 16:55:00.774 IST,"enterprisedb","edb",55279,"[local]",5ec7b639.
d7ef,8,"idle",2020-05-22 16:53:37 IST,4/29,0,AUDIT,000000,"statement: SELECT *
FROM dept;",,,,,,,,,,"psql","client backend","SELECT","edbaudit","select"

2020-05-22 16:55:06.548 IST,"enterprisedb","edb",55279,"[local]",5ec7b639.
d7ef,9,"idle",2020-05-22 16:53:37 IST,,0,AUDIT,000000,"disconnection: session
time: 0:01:28.732 user=enterprisedb database=edb host=[local]",,,,,,,,,,"psql",

```

(次のページに続く)

(前のページからの続き)

```
"client backend", "", "edbaudit", "disconnect"
```

** XML 監査ログファイル**

以下は、監査ログファイルの XML フォーマットです。出力は、例の外観をより明確にするためにフォーマットされています。

```
<event user="enterprisedb" database="edb" process_id="5941" remote_host=
"[local]"
  *      session_id="5ec7ac4d.1735" session_line_num="1" process_status=
  *      "authentication"
  *      session_start_time="2020-05-22 16:11:17 IST" log_time="2020-05-22
  *      16:11:17.806 IST"
  *      virtual_transaction_id="4/19" type="connect" audit_tag="edbaudit"
  *      backend_type="client backend">
  *      <error_severity>AUDIT</error_severity>
  *      <message>connection authorized: user=enterprisedb database=edb</
  *      message>
</event>
<event user="enterprisedb" database="edb" process_id="5941" remote_host=
"[local]"
  *      session_id="5ec7ac4d.1735" session_line_num="2" process_status="idle"
  *      session_start_time="2020-05-22 16:11:17 IST" log_time="2020-05-22
  *      16:11:32.558 IST"
  *      virtual_transaction_id="4/21" type="create" command_tag="CREATE SCHEMA
  *      " audit_tag="edbaudit" application_name="psql" backend_type="client
  *      backend">
  *      <error_severity>AUDIT</error_severity>
  *      <message>statement: CREATE SCHEMA edb;</message>
</event>
<event user="enterprisedb" database="edb" process_id="5941" remote_host=
"[local]"
  *      session_id="5ec7ac4d.1735" session_line_num="3" process_status="idle"
  *      session_start_time="2020-05-22 16:11:17 IST" log_time="2020-05-22
  *      16:12:00.199 IST"
  *      virtual_transaction_id="4/23" type="create" command_tag="CREATE TABLE"
  *      audit_tag="edbaudit" application_name="psql" backend_type="client
  *      backend">
  *      <error_severity>AUDIT</error_severity>
  *      <message>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));
  *      </message>
</event>
```

(次のページに続く)

(前のページからの続き)

```

<event user="enterprisedb" database="edb" process_id="5941" remote_host=
"[local]"
  *      session_id="5ec7ac4d.1735" session_line_num="4" process_status="idle"
  *      session_start_time="2020-05-22 16:11:17 IST" log_time="2020-05-22
  *      16:12:10.992 IST"
  *      virtual_transaction_id="4/24" type="insert" command_tag="INSERT" audit_
  *      tag="edbaudit" application_name="psql" backend_type="client backend">
  *      <error_severity>AUDIT</error_severity>
  *      <message>statement: INSERT INTO dept VALUES
  *              (10,&apos;ACCOUNTING&apos;,&apos;NEW YORK&apos;
  *              );
  *      </message>
</event>
<event user="enterprisedb" database="edb" process_id="5941" remote_host=
"[local]"
  *      session_id="5ec7ac4d.1735" session_line_num="5" process_status="idle"
  *      session_start_time="2020-05-22 16:11:17 IST" log_time="2020-05-22
  *      16:12:21.764 IST"
  *      virtual_transaction_id="4/25" type="update" command_tag="UPDATE" audit_
  *      tag="edbaudit" application_name="psql" backend_type="client backend">
  *      <error_severity>AUDIT</error_severity>
  *      <message>statement: UPDATE department SET loc = &apos;BOSTON&apos;
  *              WHERE deptno = 10;
  *      </message>
</event>
<event user="enterprisedb" database="edb" process_id="5941" remote_host=
"[local]"
  *      session_id="5ec7ac4d.1735" session_line_num="6" process_status="UPDATE"
  *      session_start_time="2020-05-22 16:11:17 IST" log_time="2020-05-22
  *      16:12:21.765 IST"
  *      virtual_transaction_id="4/25" type="error" sql_state_code="42P01"
  *      command_tag="UPDATE" audit_tag="edbaudit" application_name="psql"
  *      backend_type="client backend">
  *      <error_severity>ERROR</error_severity>
  *      <message>relation &quot;department&quot; does not exist</message>
  *      <query>UPDATE department SET loc = &apos;BOSTON&apos; WHERE deptno =
  *      10;</query>
  *      <query_pos>8</query_pos>
</event>
<event user="enterprisedb" database="edb" process_id="5941" remote_host=
"[local]"
  *      session_id="5ec7ac4d.1735" session_line_num="7" process_status="idle"
  *      session_start_time="2020-05-22 16:11:17 IST" log_time="2020-05-22
  *      16:12:34.878 IST"
  *      virtual_transaction_id="4/26" type="update" command_tag="UPDATE" audit_

```

(次のページに続く)

(前のページからの続き)

```

*      tag="edbaudit" application_name="psql" backend_type="client backend">
*      <error_severity>AUDIT</error_severity>
*      <message>statement: UPDATE dept SET loc = &apos;BOSTON&apos; WHERE
*                      deptno = 10;
*      </message>
</event>
<event user="enterprisedb" database="edb" process_id="5941" remote_host=
"[local]"
*      session_id="5ec7ac4d.1735" session_line_num="8" process_status="idle"
*      session_start_time="2020-05-22 16:11:17 IST" log_time="2020-05-22
*      16:12:45.471 IST"
*      virtual_transaction_id="4/27" type="select" command_tag="SELECT" audit_
*      tag="edbaudit" application_name="psql" backend_type="client backend">
*      <error_severity>AUDIT</error_severity>
*      <message>statement: SELECT * FROM dept;</message>
</event>
<event user="enterprisedb" database="edb" process_id="5941" remote_host=
"[local]"
*      session_id="5ec7ac4d.1735" session_line_num="9" process_status="idle"
*      session_start_time="2020-05-22 16:11:17 IST" log_time="2020-05-22
*      16:12:53.048 IST"
*      type="disconnect" audit_tag="edbaudit" application_name="psql" backend_
*      type="client backend">
*      <error_severity>AUDIT</error_severity>
*      <message>disconnection: session time: 0:01:35.243 user=enterprisedb
*      database=edb host=[local]</message>
</event>

```

3.5.5 Using Error Codes to Filter Audit Logs

</div>

Advanced Server には、ユーザー指定のエラーコードを含むログファイルエントリを Advanced Server ログ ファイル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、`shared_preload_libraries` パラメータで指定された値に次の値を追加して、拡張機能を有効にする必要があります。

```
$libdir/edb_filter_log
```

次に、`edb_filter_log.errcodes` パラメータを使用して、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、ログファイルから除外する 1 つ以上のエラーコードを指定します。マルチプルのエラーコードをコンマ区切りリストで指定します。

例、`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結果ログエントリ：

23505（一意性制約に違反するため）

23502（非 NULL 制約に違反するため）

22012（ゼロ除算用）

ログファイルから省略されます。

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、次のコア文書を参照してください。

< <https://www.postgresql.org/docs/current/ 静的/errcodes-appendix.html> >

3.5.6 Using Command Tags to Filter Audit Logs

</div>

エラーメッセージを表示するものを除くログファイルの各エントリーには、**コマンドタグ**が含まれます。コマンドタグは、その特定のログエントリー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、後続のツールを使用してログファイルをスキャンし、特定の 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。

以下は XML フォームの例です。この例は、レビューしやすいようにフォーマットされています。コマンドタグは、例の CREATE ROLE、ALTER ROLE、および DROP ROLE の値を持つイベント要素の `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">
```

(次のページに続く)

(前のページからの続き)

```

*      <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]"
*      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 フォームの同じ例です。コマンドタグは、各エントリーの最後から 2 番目の列です。リストでは、空 (“”) で表示される列は、edb_audit_tag パラメータの値になります (指定されている場合)。

各監査ログエントリーはマルチプルの行に分割されて表示され、結果の外観をより明確にするために、監査ログエントリーの間に空白行が挿入されています。

```

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","",

```

(次のページに続く)

(前のページからの続き)

`"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,00000,"statement: DROP
ROLE newuser;",,,,,,,,,,"psql","client backend","DROP ROLE","", "drop"
```

3.5.7 Redacting Passwords in Audit Logs

</div>

postgresql.conf ファイルの下の GUC パラメータ `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編集するようにサーバーに指示できます。

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 } ]
```

パスワードのリダクションを有効にするには、まず `postgresql.conf` ファイルを変更してパラメータを有効にし、`shared_preload_libraries` パラメータで指定された値に次の値を追加する必要があります。

```
$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、サーバーをリスタート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、サーバーはパスワードを `x` に編集します。例、コマンド：

```
CREATE USER carol with login PASSWORD '1safepwd';
```

次のようにログファイルに追加されます。

```
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"
```

次のようにログファイルに追加されます。

```
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"
```

(次のページに続く)

パスワードを含むステートメントがログに記録され、サーバーはパスワードテキストを **x** に編集します。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記録されると、サーバーはその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

</div>

EDB 監査ログアーカイブを使用すると、データベース管理者は監査ログディレクトリによって消費されるスペースを制御し、監査ログファイルの管理にヘルプます。監査アーカイバは、監査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、および期限切れを処理します。`edb_audit_archiver_timeout` パラメータは、適切なタイミングで監査ログファイルの圧縮または期限切れをトリガーします。監査アーカイビング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。

Audit Archiver は次のことを支援します。

1. 圧縮用の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。パラメーター `edb_audit_archiver_compress_time_limit` および `edb_audit_archiver_compress_size_limit` に基づいて、圧縮用のログファイルを決定します。3. `edb_audit_archiver_compress_command` パラメータに基づいて圧縮コマンドを指定して、ログファイルの圧縮を実行します。`edb_audit_archiver_expire_time_limit` および `edb_audit_archiver_expire_size_limit` パラメータに基づいて、削除するログファイルを決定します。5. `edb_audit_archiver_expire_command` パラメータで指定された有効期限コマンドを実行して、ログファイルを削除します。

パラメータ `edb_audit_rotation_day` を目的の値に設定して、古い監査ログファイルをローテートすることにより、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できます。監査ログレコードのスペースが使用できないイベント、監査ログレコードはFIFO（先入れ先出し）ベースで上書きされます。

次の手順では、ログファイルの圧縮と有効期限を有効にするために Advanced Server を構成する方法について説明します。

- `postgresql.conf` ファイルで `edb_audit_archiver` パラメータを `on` に設定して、監査ログのアーカイビングを有効にします。
- ログファイルの圧縮を有効にするには、パラメータ `edb_audit_archiver_compress_size_limit` および `edb_audit_archiver_compress_time_limit` を目的の値に設定します。
- ログファイルの有効期限を有効にするには、パラメータ `edb_audit_archiver_expire_size_limit` および `edb_audit_archiver_expire_time_limit` を目的の値に設定します。
- 圧縮と有効期限の両方を有効にするには、パラメータ `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` を目的の値に設定します。

以下は、`postgresql.conf` ファイルの設定パラメータ設定の例です。

構成ファイルの `edb_audit_archiver` パラメータの設定は、データベースクラスタ全体に影響します。データベースクラスタは、`postgresql.conf` ファイルに示されているように、`edb_audit_archiver` が `on` に設定されて確立されます。監査ログファイルは、`edb_audit` 設定パラメータの設定に応じて CSV フォーマットで生成されます。

```
logging_collector = 'on'
edb_audit = 'csv'
edb_audit_archiver = 'on' # (Change requires restart)
```

例 1

以下は、`edb_audit_archiver`、`edb_audit_archiver_compress_size_limit`、および `edb_audit_archiver_compress_time_limit` が 10MB をより大きい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る例です。

```
edb_audit_archiver = 'on'
edb_audit_archiver_compress_size_limit = 10
edb_audit_archiver_compress_time_limit = 0
```

監査ログファイルを圧縮する前に、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` パラメータは、最新のファイルを除くログファイルをチェックし、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、すでに圧縮されている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
```

例 2

以下は、`edb_audit_archiver`、`edb_audit_archiver_expire_size_limit`、および `edb_audit_archiver_expire_time_limit` が 1 時間以上経過した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る例です。

```
edb_audit_archiver = 'on'
edb_audit_archiver_expire_size_limit = 0
edb_audit_archiver_expire_time_limit = 3600
```

監査ログファイルを圧縮する前に、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 パラメータは、最新のファイルを除くログファイルをチェックし、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
```

例 3

以下は、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ために、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 が設定されている例です。

```
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/'
```

監査ログファイルを圧縮する前に、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 パラメータは、最新のファイルを除くログファイルをチェックし、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

有効期限を確認する際、`edb_audit_archiver` は監査ログディレクトリに少なくとも 5MB 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 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
```

有効期限コマンドは `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、`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。圧縮と有効期限が切れると、バックアップログディレクトリが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

</div>

オブジェクトレベルの監査では、特定のテーブルの特定のデータ操作言語ステートメント (`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、(`@`) パラメータで (`@`) 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することもできます。データ操作言語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` パラメータ値を指定します。

```
{ select | update | delete | insert }@groupname
```

例

以下は、`edb_audit_connect` および `edb_audit_statement` が次のデフォルト以外の値で設定されている例です。

```
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た 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;
    edb_audit_connect
-----
all
(1 row)

edb=# SHOW edb_audit_statement;
      *      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

```

(次のページに続く)

(前のページからの続き)

```

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';
SET

edb=# SELECT reloptions FROM pg_class WHERE relname IN('emp', 'dept', 'bonus', 'sal');
      *
      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';
      *
      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;
 empno |  ename  | job   | mgr | hiredate | sal | comm | deptno
-----+-----+-----+-----+-----+-----+-----+-----
(0 rows)

edb=# SELECT * FROM dept;
 deptno |  dname  | loc
-----+-----+-----
(0 rows)

edb=# SELECT * FROM bonus;
 ename | job | sal | comm
-----+-----+-----+-----
(0 rows)

edb=# SELECT * FROM sal;
 grade | losal | hisal
-----+-----+-----

```

(次のページに続く)

(前のページからの続き)

(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;
```

(次のページに続く)

(前のページからの続き)

```
DELETE 1
edb=# DELETE FROM dept WHERE deptno = 20;
DELETE 1
edb=# DELETE FROM bonus WHERE sal = 4000;
DELETE 1
edb=# DELETE FROM sal WHERE losal = 5000;
DELETE 3
```

enterprisedb ユーザおよび edb データベースの edb_audit_statement パラメータを '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、SELECT、INSERT、UPDATE または (@) を含む (@) ステートメントを監査し、監査ログファイルのグループを除外できます。

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、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 UPDATE 監査ステートメント UPDATE _UPDATE の削除 low_security を削除し、ログファイル内のテーブルの high_security 監査グループの削除ステートメントを除外します。

結果の監査ログファイルには、次のものが含まれます。

各監査ログエントリはマルチプルの行に分割されて表示され、結果の外観をより明確にするために、監査ログエントリの上に空白行が挿入されています。

```
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"

2021-07-20 01:51:14.874 IST,"enterprisedb","edb",87490,"[local]",60f662f7.
```

(次のページに続く)

(前のページからの続き)

```

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"

2021-07-20 01:54:45.887 IST,"enterprisedb","edb",87490,"[local]",60f662f7.

```

(次のページに続く)

(前のページからの続き)

```

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

</ div>

- 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メソッドを定義する仕様 (Unicode Technical Report # 10) です。照合は、`SELECT ... ORDER BY` 句と同様にデータがどのようにソートされるかを意味します。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ファクタです。

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であるかもしれない手紙は、言語に依存して異なる順序で照合すると予想されるかもしれません。- アクセントやウムラウトのような特定の言語で文字とともに現れる文字は、言語によっては予想される照合順序にインパクトを与えます。- 一部の言語では、いくつかの連続する文字の組み合わせは、照合順序シーケンスに関して単一の文字として扱われる必要があります。- ケースに応じた文字の照合順序に関して、特定の設定があります。例、小文字の文字が同じ文字の大文字の前に照合さフォームか、またはその逆である必要があります。- アンダースコア文字、ハイフン、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、単に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ついての設定があります。

Unicode でサポートされる膨大な数の言語でこれらのバリエーションのすべてを考慮して、照合シーケンスを決定するための特定の基準を選択するメソッドの必要性があります。これが Unicode Collation Algorithm が定義するものです。

!!! Note * 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-tree インデックスの作成などの並べ替えタスクは、ICU 以外の照合順序でかかる時間の半分より小さいで完了できます。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。

次のセクションでは、Unified Collation Algorithm の概念を簡潔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、多くの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 Unicode 照合アルゴリズムの基本概念

Unicode Collation Algorithm の公式情報は、* Unicode Technical Report # 10 *で指定されています。これは、Unicode Consortium のウェブサイトで見つけることができます。

<http://www.unicode.org/reports/tr10/>

ICU –Unicode の国際コンポーネントも多くの有用な情報を提供します。照合順序の概念の説明は、次のウェブサイトにあります。

< [http://userguide.icu-project.org/ 照合順序/ concepts](http://userguide.icu-project.org/照合順序/concepts) >

Unicode Collation Algorithm の背後にある基本概念は、マルチレベル比較の使用です。これは、いくつかのレベルが定義されていることを意味し、次の箇条書きでレベル 1 からレベル 5 としてリストされています。各レベルは比較のタイプを定義します。文字列は、レベル 1 とも呼ばれるプライマリレベルを使用して最初に比較されます。

プライマリレベルに基づいてオーダーを決定できる場合、アルゴリズムが実行されます。プライマリレベルに基づいてオーダーを決定できない場合、二次的なレベル、レベル 2 が適用されます。二次的なレベルに基づいてオーダーを決定できる場合はアルゴリズムが実行され、そうでない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オーダー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-基本文字のプライマリレベル。** 文字や数字などの基本文字のオーダーによって、 $A < B$ などの違いが決まります。
- **レベル 2-アクセントのセカンダリレベル。** プライマリレベルの違いがない場合、アクセントなどの文字の有無によって $a < á$ などのオーダーが決まります。
- **レベル 3-ケースの 3 次レベル** プライマリレベルまたは二次的なレベルの違いがない場合、ケースの違いによって $a < A$ などのオーダーが決まります。
- **レベル 4-句読点の 4 次レベル。** プライマリ、二次的な、または 3 次レベルの違いがない場合、空白文字文字、制御文字、および句読点の有無によって $-A < A$ などのオーダーが決まります。
- **レベル 5-タイブレークの同一レベル。** プライマリ、二次的な、ターシャリ、またはクォータナリレベルの違いがない場合、コードポイント値などの他の違いがオーダーを決定します。

3.6.2 Unicode の国際コンポーネント

Unicode Collation Algorithm 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ます。ソフトウェアは、C / C ++および Java ライブラリのセットです。

Advanced Server を使用して ICU コンポーネントを呼び出して照合順序を生成する照合順序を作成する場合、結果は* ICU 照合順序*と呼ばれます。

ロケール照合

ロケールの照合順序を作成する場合、通常、指定されたロケールの事前定義された ICU ショートフォーム名前が提供されロケール。

- ICU ショートフォームは、照合順序のプロパティである照合順序属性を指定するメソッドです。Collation Attributes は、照合順序属性に関する追加情報を提供します。

ロケール用に事前定義された ICU ショートフォームがあります。ロケールの ICU ショートフォームには、特定のロケールで通常使用される照合順序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順序属性のリスト全体を指定する必要がなくなるため、照合順序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 の短いフォームの名前は、icu_short_form 列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;

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 ショートフォームのデフォルトの特性は、照合順序プロパティを指定してそのプロパティをオーバーライドすることで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 照合順序を作成する場合、照合順序の目的の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 ショートフォームで実行できます。ただし、より具体的な情報が必要な場合は、**照合順序属性**を使用して照合順序プロパティを指定できます。

照合属性は、テキスト文字列の照合順序シーケンス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の膨大な言語を網羅しているため、これらの照合順序属性は非常に複雑です。

照合順序属性の完全で正確な意味と使用法については、ICU – International Components for Unicode ウェブ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/ 照合順序/ concepts](http://userguide.icu-project.org/照合順序/concepts) >

以下は、照合順序属性の簡単な概要と、ICU ショートフォームメソッドを使用して指定する方法です。

各照合順序属性は大文字で表され、次の箇条書きにリストされています。各属性の有効な有効値は、カッコに示されたコードによって与えられます。一部のコードには、すべての属性に対して一般的な意味があります。**** 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。無視できない (N) に設定すると、変数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、変数文字の違いはマイナー重要ではありません (つまり、基本文字を比較する場合、変数文字は無視されます)。
- **** C** –大文字 (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、小文字の最初 (L) が必要な場合に指定されます。
- **** E** –ケースレベル (X、O、D)。******強度属性と組み合わせて設定します。ケースレベル属性は、アクセントではなく大文字小文字を無視する場合に使用されます。
- **** F** –フランス語照合 (X、O、D)。**** on** に設定すると、二次的な差異 (アクセントの存在) は、カナダのフランス語ロケールで行われるように、文字列の後ろからソートされます。
- **** H** –ひらがな第 4 級 (X、O、D)。******日本語文字列の JIS X 4061 照合順序との互換性のために、ひらがなとカタカナを区別するための追加レベルが導入されました。
- **** 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テキストの標準的な等価性の問題を処理します。そのため、そのような文字を並べ替えまたは比較するときに問題が発生します。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順序レベル。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小文字をアカウントに入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、同じ強度 (つまり、レベル 5) を表します。
- **** T** –可変トップ (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可能。16 進数は、無視可能と見なされる最高の文字シーケンスを指定します。例、空白文字は無視できるが、可視される変数文字は無視できない場合、変数上部が 0020 に設定され、代替属性が S に設定され、強度属性が 3 に設定されます (スペース文字バックスペース、タブ、改行コード、復帰などの他の非表示変数文字の値は 0020 より小さい。可視されるすべての句読点の値は 0020 をより大きいです。

照合順序属性とその値のセットは、目的の属性値と連結された照合順序属性文字で構成されるテキスト文字列で表されます。各属性/値のペアは、次の例に示すように、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

照合属性をロケールの ICU ショートフォーム名とともに指定して、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 という名前付けの ICU ショートフォームが他の多くの照合順序属性/値のペアで変更された例です。

AN_CX_EX_LR00T_NO_S3

上記の例では、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 (X) に設定されます。ケースレベル属性 (E) は、オフ (X) に設定されます。正規化属性 (N) は (O) に設定されます。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 の短いフォームです。

3.6.3 照合の使用

新しく定義された ICU 照合順序は、CREATE TABLE コマンドの列指定など、SQL コマンドで COLLATION "collation_name"句を使用したり、SELECT コマンドの ORDER BY 句の式に追加したりできる場所であればどこでも使用できます。

以下は、米国の言語 (en_US.UTF8) に基づいた ICU 照合の作成と使用の例です。

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、対応する大文字 (CL) の前に小文字の文字を強制フォームにソートします。

照合フォームは、大文字の文字を小文字 (CU) の前に強制的にソートします。

照合 icu_collate_ignore_punct により、並べ替え中に変数文字 (空白文字および句読点) が無視されます (AS)。

照合 icu_collate_ignore_white_sp により、並べ替え中に空白文字およびその他の不可視の変数文字が無視されますが、可視の変数文字 (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_LROOT'
);

CREATE COLLATION icu_collate_uppercase (
    *   LOCALE = 'en_US.UTF8',
    *   ICU_SHORT_FORM = 'AN_CU_EX_NX_LROOT'
);

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'
);

```

!!! Note * 照合の作成時に、LROOT 照合順序を変更するための属性が指定されている場合、ICU は通知およびワーニング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
*
*                               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)

```

次の表が作成され、入力されます。

```

CREATE TABLE collate_tbl (
    *   id          INTEGER,
    *   c2          VARCHAR(2)
);

```

(次のページに続く)

(前のページからの続き)

```

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');
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');

```

次のクエリーは、デフォルトの照合順序を使用して列 `c2` でソートします。9、10、および 11 の `id` 列値を持つ変数文字（空白文字および句読点）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;
 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)

```

次のクエリーは、列 `c2` で照合順序 `icu_collate_lowercase` を使用してソートします。フォームにより、同じベース文字の大文字の前に小文字の文字が強制フォームにソートされます。また、9 属性の 9、10、および 11 の値が `id` の行がすべての文字と数字の前にリストされるように、ベース文字を比較するときに、`AN` 属性が同じレベルで並べ替えオーダーに変数文字を強制的に含めることにノートしてください。

```

edb=# SELECT * FROM collate_tbl ORDER BY c2 COLLATE "icu_collate_lowercase";
 id | c2
-----+-----
 11 |  B
 10 | -B
  9 | .B
  7 |  1
  8 |  2

```

(次のページに続く)

(前のページからの続き)

```

4 | a
1 | A
5 | b
2 | B
6 | c
3 | C
(11 rows)

```

次のクエリーは、照合順序 `icu_collate_uppercase` を使用して列 `c2` でソートします。フォームにより、大文字の文字が同じベース文字の小文字の前に強制フォームにソートされます。

```

edb=# SELECT * FROM collate_tbl ORDER BY c2 COLLATE "icu_collate_uppercase";
 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)

```

次のクエリーは、列 `c2` で照合順序 `icu_collate_ignore_punct` を使用してソートします。これにより、変数文字が無視されるため、`id` 値が 9、10、および 11 の行は無視された変数文字の直後の文字 `B` でソートされます。

```

edb=# SELECT * FROM collate_tbl ORDER BY c2 COLLATE
"icu_collate_ignore_punct";
 id | c2
----+----
* 7 | 1
* 8 | 2
* 4 | a
* 1 | A
* 5 | b
* 11 | B
* 2 | B
* 9 | .B
* 10 | -B
* 6 | c

```

(次のページに続く)

(前のページからの続き)

```
* 3 | C
(11 rows)
```

次のクエリーは、照合順序 `icu_collate_ignore_white_sp` を使用して列 `c2` でソートします。照合順序の `AS` および `T0020` 属性により、コードポイントが 16 進数 `0020` 以下の変数文字は無視されますが、コードポイントが 16 進数 `0020` より大きい変数文字はソートに含まれます。

行の値が 11 であり、スペース文字（16 進 `0020`）で始まる行は、文字 `B` でソートされます。16 進数の `0020` より大きい目に可視句読点でスタート 9 および 10 の `id` 値を持つ行は、これらの特定の変数文字がベース文字を比較するときに同じレベルのソートオーダーに含まれるため、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";
 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

</ div>

- EDB Resource Manager *は、Advanced Server プロセスが使用す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 Advanced Server 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使いすぎて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 の使用に関するキーポイントです。

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、Advanced Server インスタンスのすべてのデータベースで使用可能な名前付け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 Advanced Server プロセスは、EDB Resource Manager によって制御されるため、グループ内のすべてのプロセスの総リソース使用量は、グループで定義された制限の近くに維持されます。- データ定義言語コマンドを使用して、リソースグループを作成、変更、および削除します。これらのコマンドは、スーパーユーザ特権を持つデータベースユーザ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望ましい総消費レベルは、**リソースタイプパラメーター**によって定義されます。現在 EDB Resource Manager で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があります。- 複数のリソースグループを作成できます。それぞれのリソースタイプパラメータの設定が異なるため、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に近づけます。リソースグループに設定が定義された複数のリソースタイプパラメータがある場合、特定のリソースタイプの実際のリソース消費は、定義されたリソースタイプパラメータ設定よりも大幅に低くなる可能性があります。これは、EDB Resource Manager が、**定義された制限内で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**を維持しようとするプロセスを調整するためです。- 利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、共有グローバルシステムカタログに保存されます。したがって、特定の Advanced Server インスタンス内のすべてのデータベースでリソースグループを利用できます。- `edb_max_resource_groups` 設定パラメータは、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 リソースグループです。このパラメータの変更は、データベースサーバのリスタートに有効になります。- `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。`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` コマンドを使用してデータベースに割り当てることができます。`postgresql.conf` ファイルでパラメータ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には、デフォルト設定で `pg_dumpall` バックアップユーティリティを使用します（つまり、`--globals-only`、`--roles-only`、または `--tablespaces-only` オプションのいずれも指定しないでください）。

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 リソースグループの作成

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>;
```

説明

CREATE RESOURCE GROUP コマンドは、指定された名前でもリソースグループを作成します。リソース制限は、ALTER RESOURCE GROUP コマンドを使用してグループに定義できます。リソースグループは、Advanced Server インスタンス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、スーパーユーザ権限が必要です。

パラメータ

group_name

リソースグループの名前。

例

次の例では、resgrp_a、resgrp_b、および resgrp_c 名前付けの 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
```

次のクエリーは、edb_resource_group カタログ内のリソースグループのエントリを示しています。

```
edb=# SELECT * FROM edb_resource_group;
 rgrpname | rgrpcpuratelimit | rgrpdirtyratelimit
-----+-----+-----
 resgrp_a |                0 |                0
 resgrp_b |                0 |                0
 resgrp_c |                0 |                0
(3 rows)
```

4.1.2 リソースグループの変更

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>;
```

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 句を含む最初のフォームは、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 句を含む 2 番目のフォームは、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対する定義済みの制限がありません。

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、スーパーユーザ権限が必要です。

パラメータ

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られる新しい名前。

resource_type

使用量の値が設定され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。

value | DEFAULT

value が指定されている場合、resource_type に割り当てられるリテラル値。DEFAULT を指定すると、リソースグループの resource_type の割り当てがリセットされます。

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
```

次のクエリーは、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;
 rgrpname | rgrpcpuratelimit | rgrpdirtyratelimit
-----+-----+-----
newgrp   | 0                | 0
resgrp_b | 0.5              | 6144
resgrp_c | 0                | 0
(3 rows)
```

4.1.3 ドロップリソースグループ

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>;
```

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、スーパーユーザ権限が必要です。

パラメータ

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ないでください。この場合、通知が発行されます。

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
```

4.1.4 リソースグループへのプロセスの割り当て

以下に示すように、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;
     edb_resource_group
-----
      resgrp_b
(1 row)
```

グループのリソースタイプの設定は、現在のプロセスですぐに有効になります。コマンドを使用して現在のプロセスに割り当てられた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。

デフォルト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インスタンス全体に割り当てることにより、デフォルトでプロセスを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 コア文書を参照してください。

< <https://www.postgresql.org/docs/current/sql-alterrole.html> >

デフォルトのリソースグループは、ALTER DATABASE ... SET コマンドによって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 コア文書を参照してください。

< <https://www.postgresql.org/docs/current/sql-alterdatabase.html> >

以下に示すように、postgresql.conf ファイルで edb_resource_group 設定パラメータ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。

```
# - 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 を変更するには、データベースサーバインスタンスで有効になる前に、構成ファイルのリロードが必要です。

4.1.5 リソースグループからプロセスを削除する

以下に示すように、`edb_resource_group` を `DEFAULT` に設定するか、`RESET edb_resource_group` 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。

```
edb=# SET edb_resource_group TO DEFAULT;
SET
edb=# SHOW edb_resource_group;
     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` フォームを使用します。

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` フォームを使用します。

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、`edb_resource_group` ファイルの `edb_resource_group` 設定パラメータを空の文字列に設定し、構成ファイルをリロードロードします。

4.1.6 リソースグループ内のプロセスの監視

リソースグループが作成された後、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は、`edb_all_resource_groups` ビューから取得できます。

`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。

!!! Note * 列 `per_process_cpu_rate_limit` および `per_process_dirty_rate_limit` は、プロセスで使用される**実際**のリソース消費量を示しませんが、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 EDB Resource Manager が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。

以下に、リソースグループ `resgrp_a` にアクティブプロセスが含まれていない場合、リソースグループ `resgrp_b` に 2 つのアクティブプロセスが含まれ、リソースグループ `resgrp_c` にアクティブプロセスが 1 つ含まれている場合の `edb_all_resource_groups` を示します。

```
edb=# SELECT * FROM edb_all_resource_groups ORDER BY group_name;
- [ 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
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 レート制限とダーティレート制限の設定は次のとおりです。

```
edb=# SELECT * FROM edb_resource_group;
 rgrpname | rgrpcpuratelimit | rgrpdirtyratelimit
-----+-----+-----
resgrp_a | 0.5              | 12288
resgrp_b | 0.4              | 6144
resgrp_c | 0.3              | 3072
(3 rows)
```

edb_all_resource_groups ビューでは、per_process_cpu_rate_limit と 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値であることにノートしてください。

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を設定することで制御されます。

`cpu_rate_limit` パラメータを、グループ内のすべてのプロセスの合計の同時 CPU 使用率が超えてはならない壁時計時間に対する 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 より大きいに設定することにより、`cpu_rate_limit` を複数の CPU コアに適用できます。例、`cpu_rate_limit` が 2.0 に設定されている場合、2 つの CPU の 100 % を使用します。

`cpu_rate_limit` パラメータの有効レンジは 0 から 1.67772e+07 です。0 の設定は、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。

100 を掛けると、`cpu_rate_limit` はリソースグループの CPU 使用率としても解釈できます。

EDB Resource Manager は、*CPU スロットリング*を使用して、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、短時間中断され、スリープモードになる場合がありますインターバル。このような中断が発生するタイミングと方法は、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 使用制限は `resgrp_a` では 50 %、`resgrp_b` では 40 %、`resgrp_c` では 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、合計 CPU 使用率は約 40 % などに維持されます。

```
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
```

次のクエリーは、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;
 rgrpname | rgrpcpuratelimit
-----+-----
 resgrp_a |                0.5
 resgrp_b |                0.4
```

(次のページに続く)

(前のページからの続き)

```
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ている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はすべて、変更の影響を直ちに受けます。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は、グループの合計 CPU 使用率が 50 % ではなく 30 % 近くになるように下方に調整されます。

リソースグループの CPU レート制限を設定する効果を説明するために、次の例では、クエリー `SELECT 20000!` によって実行される CPU 集約型の 20000 階乗 (`20000 * 19999 * 19998` などの乗算) 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、前のクエリーで示した CPU レート制限設定を持つリソースグループが使用されています。

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。次に、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;
     edb_resource_group
-----
      resgrp_b
(1 row)
edb=# SELECT 20000!;
```

2 番目のセッションでは、Linux `top` コマンドを使用して、%CPU 列の下に表示される 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化するときの任意の時点での snapshot 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
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、`edb-postgres` 列の `COMMAND` 列の下に表示される行で示されます。`%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、リソースグループ `resgrp_b` に設定された 40 % の CPU 制限に近い値です。

対照的に、`psql` セッションがリソースグループから削除され、階乗計算が再度実行される場合、CPU 使用率ははるかに高くなります。

```
edb=# SET edb_resource_group TO DEFAULT;
SET
edb=# SHOW edb_resource_group;
     edb_resource_group
-----
(1 row)

edb=# SELECT 20000!;
```

`edb-postgres` の `%CPU` 列の下で、CPU 使用率は 93.6 になりました。これは、プロセスがリソースグループのパートであったときの 39.9 よりも大幅に高くなっています。

```
$ 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
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合に適用されます。この概念を次の例に示します。

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;
```

(次のページに続く)

(前のページからの続き)

```
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;
 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
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
 *
 *
 *
```

現在、%CPU という値が 19.9 と 19.6 の edb-postgres 名前付けの 2 つのプロセスがあり、その総和はリソースグループ **resgrp_b** に設定された 40 % の CPU 使用率に近くなっています。

次のコマンドシーケンスは、0.5 秒間隔でサンプリングされたすべての edb-postgres プロセスの総和を表示します。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 % 近くに維持するときに、リソースグループ内のプロセスの合計 CPU 使用率が時間とともにどのように変化するかを示します。

```
$ while [[ 1 -eq 1 ]]; do top -d0.5 -b -n2 | grep edb-postgres | awk '{ SUM
+= $9} END { print SUM / 2 }'; done
```

(次のページに続く)

(前のページからの続き)

```

37.2
39.1
38.9
38.3
44.7
39.2
42.5
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、2つの追加の `psql` セッションが前の2つのセッションとともに使用されます。3番目と4番目のセッションは、`cpu_rate_limit .3` (30%のCPU使用率) でリソースグループ `resgrp_c` 内で同じ階乗計算を実行します。

セッション3:

```

edb=# SET edb_resource_group TO resgrp_c;
SET
edb=# SHOW edb_resource_group;
   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;
   edb_resource_group
-----
    resgrp_c

```

(次のページに続く)

(前のページからの続き)

```
(1 row)
edb=# SELECT 20000!;
```

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
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使用制限は40%と30%です。最初の2つの edb-postgres プロセスの%CPU列の総和は39.5(約40%、これは resgrp_b の制限)であり、3番目と4番目の edb-postgres プロセスの%CPU列の総和は31.6(およそ30%、つまり 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使用制限の総和は70%です。次の出力は、4つのプロセスの総和が約70%に達してい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
```

対照的に、resgrp_b に2つのセッションが残っているが3つ目のセッションがリソースグループに属さ

ない 3 つのセッションが処理されている場合、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
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にないため、58.6 %の CPU 使用率を持ち、基本的には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、3 つのセッションの総和が約 95 %であることを示しています。これは、セッションの 1 つに CPU 使用率の制限が設定されていないためです。

```
$ 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を設定することにより制御されます。

`dirty_rate_limit` パラメータを、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」にする必要のある複合レートの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` パラメータの有効レンジは 0 から $1.67772e+07$ です。0 の設定は、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**を使用して、グループ内のすべてのプロセスの集約共有バッファ書き込みレートを `dirty_rate_limit` パラメータで指定された制限の近くに保ちます。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、短時間中断され、スリープモードになる場合があります。このような中断が発生するタイミングと方法は、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` では毎秒 12288 キロバイト、`resgrp_b` では毎秒 6144 キロバイト、`resgrp_c` では毎秒 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結合書き込みレートが、毎秒約 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について、共有バッファへの結合書き込みレートは約 6144 キロバイト/秒などに維持されます。

```
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
```

次のクエリーは、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;
 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ている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はすべて、変更の影響を直ちに受

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スロットルされ、グループのダーティレートは毎秒 12288 キロバイトではなく 3072 キロバイト/秒近くなります。

リソースグループのダーティレート制限の設定の効果を説明するために、次の例では、集中的な I/O 操作に対して次の表を使用します。

```
CREATE TABLE t1 (c1 INTEGER, c2 CHARACTER(500)) WITH (FILLFACTOR = 10);
```

`FILLFACTOR = 10` 句により、`INSERT` コマンドは 1 ページあたり最大 10 % の行のみをパックします。これにより、これらの例の目的のためにダーティ共有ブロックのサンプリングが大きくなります。

`pg_stat_statements` モジュールは、SQL コマンドによってダーティ化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 1 秒あたりの実際のキロバイト書き込みレートを計算し、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。

オーダー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` 設定パラメータに追加します。

```
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
```

`pg_stat_statements_reset()` ファンクションを使用して、各例をわかりやすくするために `pg_stat_statements` ビューをクリアします。

これらの例では、前のクエリーで示されたダーティレート制限設定を持つリソースグループが使用されています。

</div>

4.3.2 例-単一グループ内の単一プロセス

次のシーケンスのコマンドは、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` を使用するように設定されています。`pg_stat_statements_reset()` ファンクションを実行すると、`pg_stat_statements` ビューがクリアされます。

最後に、`INSERT` コマンドは、1~10,000 のシリーズの整数を生成してテーブルに入力し、約 10,000 ブロックをダーティにします。

```

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;
   edb_resource_group
-----
   resgrp_b
(1 row)

edb=# SELECT pg_stat_statements_reset();
   pg_stat_statements_reset
-----
(1 row)

edb=# INSERT INTO t1 VALUES (generate_series (1,10000), 'aaa');
INSERT 0 10000

```

以下は、INSERT コマンドの結果を示しています。

```

edb=# SELECT query, rows, total_time, shared_blks_dirtied FROM
pg_stat_statements;
- [ RECORD 1 ]-----+-----
query          | INSERT INTO t1 VALUES (generate_series (?,?), ?);
rows           | 10000
total_time     | 13496.184
shared_blks_dirtied | 10003

```

実際のダーティレートは次のように計算されます。

- ミリ秒 (ms) ごとにダーティ化されたブロックの数は 10003 ブロック/13496.184 ミリ秒で、これは* 0.74117247 ブロック/ミリ秒*になります。
- 結果に 1000 を掛けて、1 秒あたりのダーティ化された共有ブロック数 (1 秒= 1000 ミリ秒) を求めます。これにより、* 741.17247 ブロック/秒*が得られます。
- 結果に 8.192 を掛けて、1 秒あたりのダーティキロバイト数 (1 ブロック= 8.192 キロバイト) を求めます。これにより、約* 6072 キロバイト/秒*が生成されます。

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/秒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していなくてもステップが繰り返される場合、ダーティバッファレート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

```

(次のページに続く)

(前のページからの続き)

```

edb=# SHOW edb_resource_group;
   edb_resource_group
-----
(1 row)

edb=# SELECT pg_stat_statements_reset();
 pg_stat_statements_reset
-----
(1 row)

edb=# INSERT INTO t1 VALUES (generate_series (1,10000), 'aaa');
INSERT 0 10000

```

以下に、リソースグループを使用しない 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;
- [ RECORD 1 ]-----+-----
query              | INSERT INTO t1 VALUES (generate_series (?,?), ?);
rows                | 10000
total_time         | 2432.165
shared_blks_dirtied | 10003

```

最初に、ダーティレート制限が 6144 キロバイト/秒に設定されているリソースグループが使用された場合の 13496.184 ミリ秒と比較して、合計時間が 2432.165 ミリ秒であったことにノートしてください。

リソースグループを使用しない実際のダーティレートは、次のように計算されます。

- ミリ秒 (ms) ごとにダーティ化されたブロック数は 10003 ブロック / 2432.165 ミリ秒で、これは * 4.112797 ブロック/ミリ秒*を生成します。
- 結果に 1000 を掛けて、毎秒ダーティ化された共有ブロックの数 (1 秒= 1000 ミリ秒) を求めます。これにより、* 4112.797 ブロック/秒*が得られます。
- 結果に 8.192 を掛けて、1 秒あたりのダーティキロバイト数 (1 ブロック= 8.192 キロバイト) を求めます。これにより、約* 33692 キロバイト/秒*が生成されます。

実際のダーティレート 33692 キロバイト/秒は、ダーティレート制限が 6144 キロバイト/秒のリソースグループが使用された場合よりも大幅に高いことに注意してください。

</ div>

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、ダーティレイト制限は、リソースグループ内のすべてのプロセスの集約に適用されます。この概念を次の例に示します。

この例では、2つの異なる `psql` セッションの2つの異なるテーブルで同時に挿入が実行されます。各セッションは、`dirty_rate_limit` が毎秒 6144 キロバイトに設定されているリソースグループ `resgrp_b` に追加されます。

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;
   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;
   edb_resource_group
-----
    resgrp_b
(1 row)

edb=# SELECT pg_stat_statements_reset();
   pg_stat_statements_reset
-----
(1 row)

edb=# INSERT INTO t2 VALUES (generate_series (1,10000), 'aaa');
INSERT 0 10000
```

!!! Note * 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()` コマンドの実行後に開始されました。

以下は、2つのセッションでの `INSERT` コマンドの結果を示しています。RECORD 3 はセッション 1 の結果を示します。RECORD 2 はセッション 2 の結果を示します。

```

edb=# SELECT query, rows, total_time, shared_blks_dirtied FROM
pg_stat_statements;
- [ 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

```

最初に、合計時間はセッション1で例ミリ秒、セッション2で30591.551ミリ秒であったことにノートしてください。したがって、リソースグループ内のアクティブプロセスが多いほど、グループ内のアクティブプロセスごとのダーティレートが遅くなります。これは、次の計算に示されています。

セッション1の実際のダーティレートは、次のように計算されます。

- ミリ秒 (ms) ごとにダーティ化されたブロックの数は 10003 ブロック / 33215.334 ミリ秒で、これは * 0.30115609 ブロック / ミリ秒 * を生成します。
- 結果に 1000 を掛けて、1 秒あたりダーティ化された共有ブロック数 (1 秒 = 1000 ミリ秒) を求めます。これにより、* 301.15609 ブロック / 秒 * が生成されます。
- 結果に 8.192 を掛けて、1 秒あたりのダーティキロバイト数 (1 ブロック = 8.192 キロバイト) を求めます。これにより、約 * 2467 キロバイト / 秒 * になります。

セッション2の実際のダーティレートは、次のように計算されます。

- ミリ秒 (ms) ごとにダーティ化されたブロックの数は 10003 ブロック / 30591.551 ミリ秒で、これは * 0.32698571 ブロック / ミリ秒 * になります。
- 結果に 1000 を掛けて、1 秒あたりのダーティ化された共有ブロック数 (1 秒 = 1000 ミリ秒) を求めます。これにより、* 326.98571 ブロック / 秒 * が得られます。
- 結果に 8.192 を掛けて、1 秒あたりのダーティキロバイト数 (1 ブロック = 8.192 キロバイト) を求めます。これにより、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) からの合計ダーティレートは、5146 キロバイト / 秒になり、リソースグループの設定ダーティレート制限 (6144 キロバイト / 秒) を下回ります。

</div>

4.3.4 例-複数のグループの複数のプロセス

この例では、2つの追加の psql セッションが前の2つのセッションとともに使用されます。3番目と4番目のセッションは、リソースグループ `resgrp_c` で同じ INSERT コマンドを実行し、`dirty_rate_limit` は 3072 キロバイト/秒です。

セッション 1 および 2 は、リソースグループ `resgrp_b` を使用して、毎秒 6144 キロバイトの `dirty_rate_limit` を使用して、前の例で示したように繰り返されます。

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;
   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;
   edb_resource_group
-----
resgrp_c
(1 row)

edb=# SELECT pg_stat_statements_reset();
   pg_stat_statements_reset
-----
(1 row)

edb=# INSERT INTO t4 VALUES (generate_series (1,10000), 'aaa');
INSERT 0 10000
```

!!! Note * 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()` コマンドの実行後に開始されました。

以下は、4つのセッションでの INSERT コマンドの結果を示しています。

RECORD 3 はセッション 1 の結果を示します。RECORD 2 はセッション 2 の結果を示します。

RECORD 4 はセッション 3 の結果を示します。RECORD 5 はセッション 4 の結果を示します。

```
edb=# SELECT query, rows, total_time, shared_blks_dirtied FROM
pg_stat_statements;
- [ 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
total_time      | 56063.697
shared_blks_dirtied | 10003
```

最初に、セッション 3（ノート）とセッションの時間と比較して、セッション 1（28407.435）とセッション 2（31343.458）の時間は、両方が同じリソースグループにあり、`dirty_rate_limit` が 6144 に設定されているため、互いに近い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。後者のグループは、ダーティレート制限が遅いため、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 の実際のダーティレートは、次のように計算されます。

- ミリ秒 (ms) ごとにダーティ化されたブロックの数は 10003 ブロック / 28407.435 ミリ秒で、これは * 0.35212612 ブロック / ミリ秒* になります。
- 結果に 1000 を掛けて、1 秒あたりダーティ化された共有ブロック数（1 秒 = 1000 ミリ秒）を求めます。これにより、* 352.12612 ブロック / 秒* が生成されます。
- 結果に 8.192 を掛けて、1 秒あたりのダーティキロバイト数（1 ブロック = 8.192 キロバイト）を求めます。これにより、約 * 2885 キロバイト / 秒* が生成されます。

セッション 2 の実際のダーティレートは、次のように計算されます。

- ミリ秒 (ms) ごとにダーティ化されたブロックの数は 10003 ブロック / 31343.458 ミリ秒であり、これは $\ast 0.31914156$ ブロック / ミリ秒 $\ast$ になります。
- 結果に 1000 を掛けて、1 秒あたりダーティ化された共有ブロックの数 (1 秒 = 1000 ミリ秒) を求めます。これにより、 $\ast 319.14156$ ブロック / 秒 $\ast$ が得られます。
- 結果に 8.192 を掛けて、1 秒あたりのダーティキロバイト数 (1 ブロック = 8.192 キロバイト) を求めます。これにより、約 $\ast 2614$ キロバイト / 秒 $\ast$ が生成されます。

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) からの合計ダーティレートは、5499 キロバイト / 秒になります。これは、リソースグループの設定ダーティレート制限 (6144 キロバイト / 秒) に近い値です。

セッション 3 の実際のダーティレートは、次のように計算されます。

- ミリ秒 (ms) ごとにダーティ化されたブロックの数は 10003 ブロック / 52727.846 ミリ秒で、これは $\ast 0.18971001$ ブロック / ミリ秒 $\ast$ になります。
- 結果に 1000 を掛けて、1 秒あたりダーティ化された共有ブロック数 (1 秒 = 1000 ミリ秒) を求めます。これにより、 $\ast 189.71001$ ブロック / 秒 $\ast$ が得られます。
- 結果に 8.192 を掛けて、1 秒あたりのダーティキロバイト数 (1 ブロック = 8.192 キロバイト) を求めます。これにより、約 $\ast 1554$ キロバイト / 秒 $\ast$ が生成されます。

セッション 4 の実際のダーティレートは、次のように計算されます。

- ミリ秒 (ms) ごとにダーティ化されたブロックの数は 10003 ブロック / 56063.697 ミリ秒であり、 $\ast 0.17842205$ ブロック / ミリ秒 $\ast$ になります。
- 結果に 1000 を掛けて、1 秒あたりのダーティ共有ブロック数 (1 秒 = 1000 ミリ秒) を求めます。これにより、 $\ast 178.42205$ ブロック / 秒 $\ast$ が得られます。
- 結果に 8.192 を掛けて、1 秒あたりのダーティキロバイト数 (1 ブロック = 8.192 キロバイト) を求めます。これにより、約 $\ast 1462$ キロバイト / 秒 $\ast$ が生成されます。

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) からの合計ダーティレートは、3016 キロバイト / 秒になります。これは、リソースグループの設定ダーティレート制限 (3072 キロバイト / 秒) に近い値です。

したがって、これは、EDB Resource Manager が、グループ内のアクティブプロセスの総ダーティレートを、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。

</ div>

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示します。

Column	Type	Description
<i>group_name</i>	<i>name</i>	リソースグループの名前。
<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`は制限がないことを意味します。
<i>per_process_cpu_rate_limit</i>	<i>float8</i>	リソースグループで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`は制限がないことを意味します。
<i>per_process_dirty_rate_limit</i>	<i>float8</i>	リソースグループで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。
		NA

</div>

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示します。

Column	Type	Description
<i>rgrpname</i>	<i>name</i>	リソースグループの名前。
<i>rgrpcpuratelimit</i>	<i>float8</i>	リソースグループの最大 CPU レート制限。`0`は制限がないことを意味します。
<i>rgrpdirtyratelimit</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`は制限がないことを意味します。

第5章 libpq C Library

libpq は、Advanced Server への C アプリケーションプログラムのインタフェースです。libpq は、クライアントプログラムが 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フェースの基礎となるエンジンでもあります。そのため、libpq の動作のいくつかの側面は、これらのパッケージソフトのいずれかが使用される場合、ユーザにとって重要になります。

libpq を使用するクライアントプログラムには、ヘッダファイル `libpq-fe.h` を含める必要があり、libpq ライブラリとリンクする必要があります。

5.1 EDB SPL での libpq の使用

EDB SPL 言語は libpq インタフェースライブラリで使用でき、次のサポートを提供します。

- Procedures、関数、パッケージソフト
- 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 サポート

以前のリリースでは、Advanced Server は次の libpq 関数を通じて REFCURSOR のサポートを提供していました。これらの関数は非推奨と見なされるようになりました。

- `PQCursorResult()`
- `PQgetCursorResult()`
- `PQnCursor()`

PQexec() と PQgetvalue() を使用して、SPL（または PL / pgSQL）ファンクションによって返された 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ヌル終了文字列のフォームで返されます。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、カーソルを通じて公開されている値を取得できます。

以下のサンプルには、実際のクライアントアプリケーションで必要となるエラー処理コードが含まれていないことにノートしてください。

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;
```

このファンクションは単一のパラメータ p_deptno を予期し、OPEN ステートメントに示されている SELECT クエリー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ーを実行し、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、変数（result 名前付け名前）を格納します。次に、ファンクションは呼び出し元にカーソルの名前を返します。

libpq を使用して C クライアントからこのファンクションを呼び出すには、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));
    * result = PQexec(conn, "BEGIN TRANSACTION");
    * if (PQresultStatus(result) != PGRES_COMMAND_OK)
    *     fail(conn, PQerrorMessage(conn));
    * PQclear(result);
```

(次のページに続く)

(前のページからの続き)

```

* 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++)
    * {
    *     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)

```

(次のページに続く)

(前のページからの続き)

```

*   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すコード行が含まれ、部門 10 のすべての従業員を含む結果セットが返されます。

```
result = PQexec(conn, "SELECT * FROM getEmployees(10)");
```

`PQexec()` ファンクションは、C プログラムに結果セットハンドルを返します。結果セットには正確に 1 つの値が含まれます。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。ファンクション `fetchAllRows()` は `FETCH ALL` ステートメントを作成し、そのステートメントを実行してから、`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は次のとおりです。

```

-- 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

```

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、呼び出し元によって指定されたレンジ内の部門で働くすべての従業員を含むカーソル (`employees`) の名前が含まれます。
- 2 番目の REFCURSOR には、呼び出し元によって指定されたレンジのすべての部門を含むカーソル (`departments`) の名前が含まれます。

この例では、ファンクションは単一の REFCURSOR を返す代わりに、`SETOF REFCURSOR` (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 を期待するのではなく、それぞれが単一の値を含む 2 つの行を期待することです (最初の行には `employees` カーソルの名前、2 番目の行 `departments` カーソルの名前が含まれます)。

```

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

```

(次のページに続く)

(前のページからの続き)

```

*   SELECT * FROM dept WHERE deptno BETWEEN p_min AND p_max;
*   RETURN NEXT departments;
END;

```

前の例のように、PQexec() と 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;
*   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);

```

(次のページに続く)

(前のページからの続き)

```

* 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++)
* {
*     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 番目のカーソルを結果ます。

```

-- 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)

(前のページからの続き)

```
* 7902,FORD,ANALYST,7566,03-DEC-81 00:00:00,3000.00,,20
-- departments --
* 20,RESEARCH,DALLAS
* 30,SALES,CHICAGO
```

5.3 配列のバインド

Advanced Server の配列バインディング機能により、1 回のラウンドトリップでデータ配列をネットワークで送信できます。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。

プリペアドステートメントで一括操作を実行します。次のファンクション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、プリペアドステートメントのセクションに記載されています。

次の関数を使用して、一括操作を実行できます。

- PQBulkStart
- PQexecBulk
- PQBulkFinish
- PQexecBulkPrepared

5.3.1 PQBulkStart

PQBulkStart() は、サーバー上の一括操作を初期化します。バルクデータをサーバーに送信する前に、このファンクション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 で指定されたフォーマットでデータを受信します。

**** API 定義****

```
PGresult *PQBulkStart(PGconn *conn,
    *                const char *Stmt_Name,
    *                unsigned int nCol,
    *                const int *paramFmts);
```

5.3.2 PQexecBulk

PQexecBulk() は、PQBulkStart() を使用したバルクオペレーション用に以前に初期化されたステートメントのデータ (paramValues) を提供するために使用されます。

このファンクションは、PQBulkStart() の後に複数回使用して、マルチプル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。

**** API 定義 ****

```
PGresult *PQexecBulk(PGconn *conn,
    *                unsigned int nRows,
    *                const char *const * paramValues,
    *                const int *paramLengths);
```

5.3.3 PQBulkFinish

このファンクションは、現在の一括オペレーションを完了します。プリペアドステートメントを再準備せずに再度使用できます。

**** API 定義 ****

```
PGresult *PQBulkFinish(PGconn *conn);
```

5.3.4 PQexecBulkPrepared

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呼び出しで一括オペレーションを実行できます。PQexecBulkPrepared() は、指定されたパラメーターでプリペアド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。このファンクションは、PQBulkStart()、PQexecBulk()、および PQBulkFinish() の機能を組み合わせたものです。このファンクションを使用する場合、一括オペレーション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、一括オペレーションを開始し、データをサーバーに渡し、一括オペレーションを終了します。

stmtName の場所に事前にプリペアド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、一度だけ解析および計画されます。

**** 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);
```

5.3.5 サンプルコード (PQBulkStart、PQexecBulk、PQBulkFinish を使用)

次の例では、PGBulkStart、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] );
    *   }
    *   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" );
}
```

5.3.6 サンプルコード (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 );
    *      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

</div>

デバッガーにより、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タイプは、SPL ストアドプロシージャ、関数、トリガー、パッケージソフト、および PL/pgSQL 関数とトリガーです。

デバッガーは、* pgAdmin 4 および EDB Postgres Enterprise Manager*と統合されています。Advanced Server を Windows ホストにインストールした場合、pgAdmin 4 が自動的にインストールされます。Windows Start メニューに pgAdmin 4 アイコンがあります。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、次を入力します。

```
yum install edb-pgadmin4*
```

Linux では、edb-asxx-server-pldebugger RPM パッケージ (xx は Advanced Server のバージョン番号) 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 をインストールすると、pgAdmin4 アイコンが [アプリケーション] メニューに追加されます。

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スタート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値を指定すると、プログラムのコードをすぐに観察してステップスルーできます。スタンドアロンデバッグは、新しいプログラムおよび最初の問題調査に使用される一般的なメソッドです。
- **インコンテキストデバッグ。**テストするプログラムは、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。最初に、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。プログラムへの最初の呼び出しを行う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一時停止します。その時点で、デバッガーは呼び出されたプログラムを制御します。その後、呼び出し元アプリケーションのコンテキスト内で実行される、呼び出されたプログラムのコードを観察してステップスルーできます。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スルーした後、中断されたアプリケーションは実行を再開します。コンテキスト内デバッグは、呼び出し側アプリケーションとの複雑な相互作用のために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困難な場合に役立ちます。

デバッグツールと操作は、スタンドアロンデバッグを使用してもコンテキスト内デバッグ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起動方法にあります。

以下のセクションでは、スタンドアロンデバッグメソッドを使用したデバッガの機能について説明します。インコンテキストデバッグ用のデバッガーの起動方法については、*Setting Global Breakpoint for In-Context*

Debugging で説明しています。

6.1 デバッガーの構成

デバッガーを使用する前に、`shared_preload_libraries` 設定パラメータにリストされているライブラリーに `$libdir/plugin_debugger` を追加して、`postgresql.conf` ファイル（Advanced Server ホームディレクトリディレクトリーの `data` サブディレクトリにあります）を編集します。

```
shared_preload_libraries = '$libdir/dbms_pipe,$libdir/edb_gen,$libdir/plugin_debugger'
```

- Linux では、`postgresql.conf` ファイルは `/var/lib/edb/asxx/data` にあります。Windows では、`postgresql.conf` ファイルは `C:\Program Files\edb\asxx\data` にあります。

`xx` は 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リスタート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ー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：オブジェクトメニューからのデバッガーの起動</ div>

pgAdmin 4 Browser の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て、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：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</ div>

トリガーは、スタンドアロンデバッグを使用してデバッグ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。コンテキスト内デバッグ用のグローバルブレー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および関数の場合と同じ指示に従います。

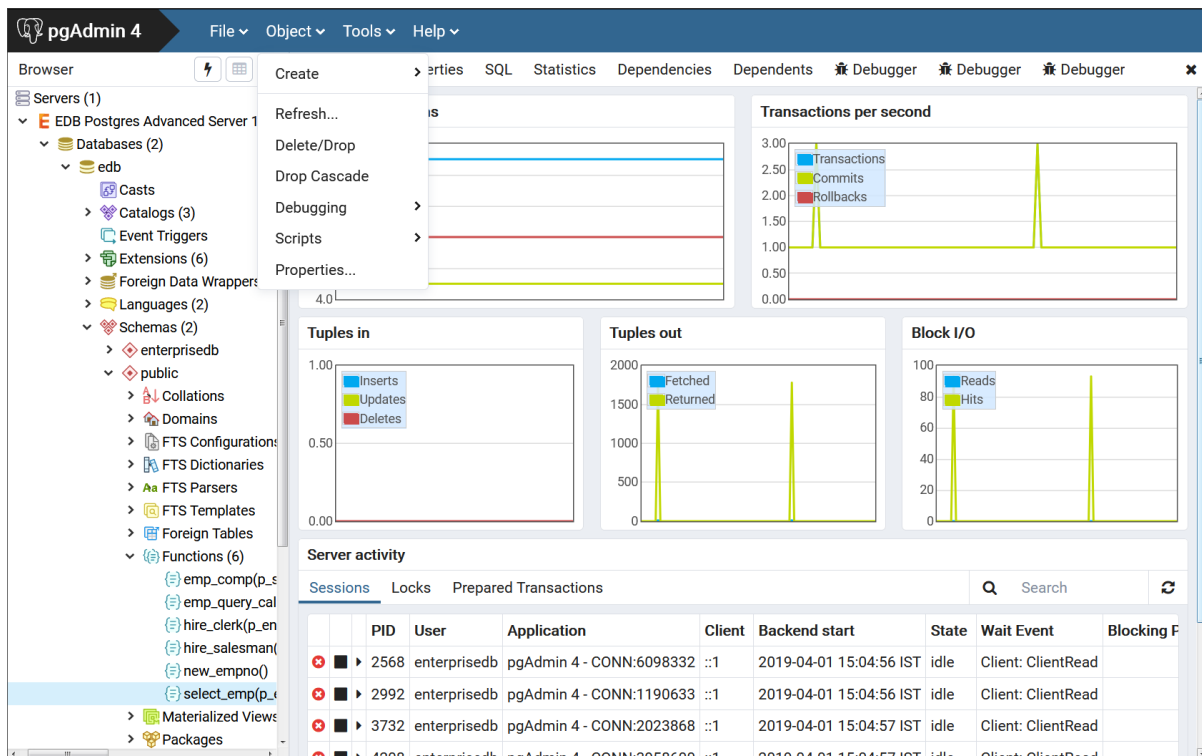


図 1: Starting the Debugger from the Object menu

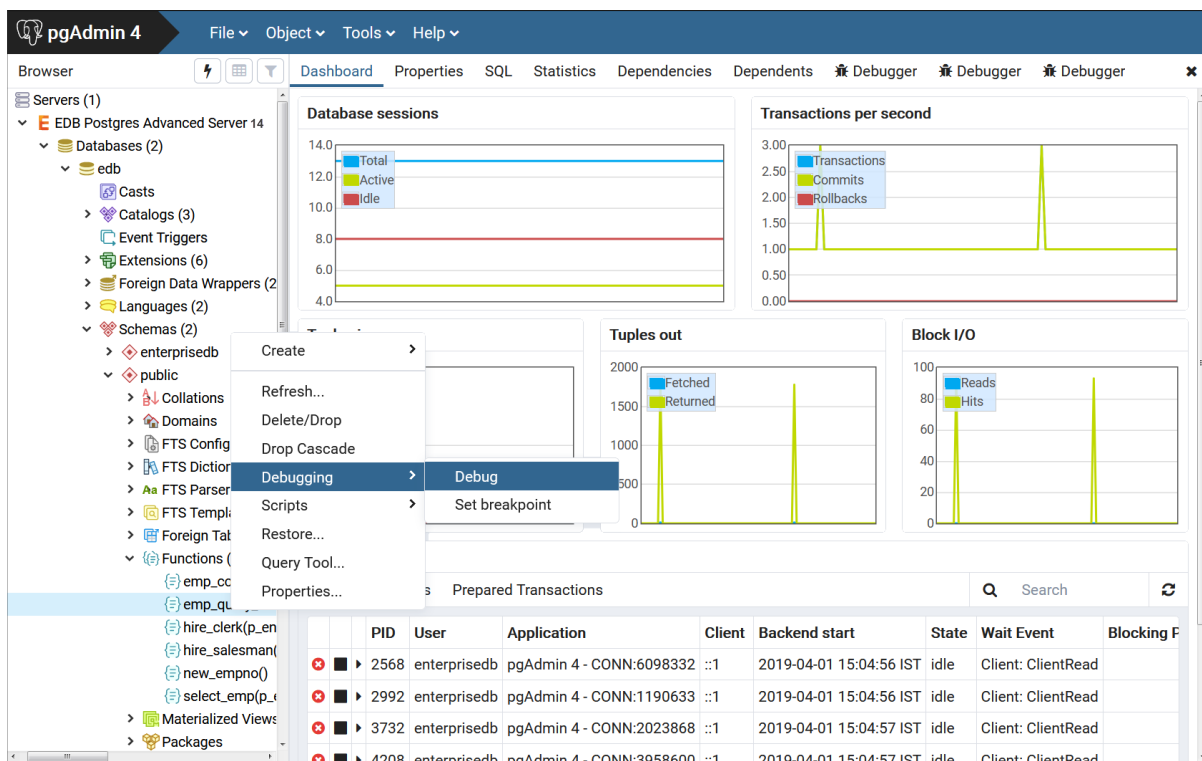


図 2: Starting the Debugger from the object's context menu

6.3 デバッガーウィンドウ

パラメーターを予期するプログラムをスタンドアロンでデバッグしている場合、Debugger ウィンドウを使用してパラメータ値を渡すことができます。デバッガをスタートすると、Debugger ウィンドウが自動的に開き、プログラムで予期される 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。

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

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、各パラメータの値を指定します。

- Name フィールドには、仮パラメータ名前が含まれます。
- Type フィールドには、パラメータデータ型が含まれています。
- Null?チェックボックスをオンにして、パラメータが NULL 値であることを示します。
- Value フィールドに式が含まれている場合は、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値が含まれます。
- 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ことを示します。
- Default Value フィールドには、パラメータのデフォルト値が含まれます。

Tab キーを押してリスト内のデータエントリー用の次のパラメータを選択するか、Value フィールドをクリックしてデータエントリー用のパラメータを選択します。

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、パッケージ初期化セクションにステップインするようデバッガに指示し、プロシージャをデバッグする前に初期化セクションコードをデバッグできるようにします。またはファンクション。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が、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。

目的のパラメータ値を入力したら、Debug ボタンをクリックしてデバッグプロセスをスタートします。Cancel ボタンをクリックして、デバッガーを終了します。

!!! Note * コンテキスト内のデバッグ中に **Debugger** ウィンドウは開きません。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値を提供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完全なデバッグサイクルを完了すると、**Debugger** ウィンドウが再び開き、新しいパラメータ値を入力してデバッグサイクルを繰り返すか、デバッグセッションを終了できます。

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のセットがあります。

デバッグ機能にアクセスするには、上部パネルにある **Tool Bar** アイコンを使用します。

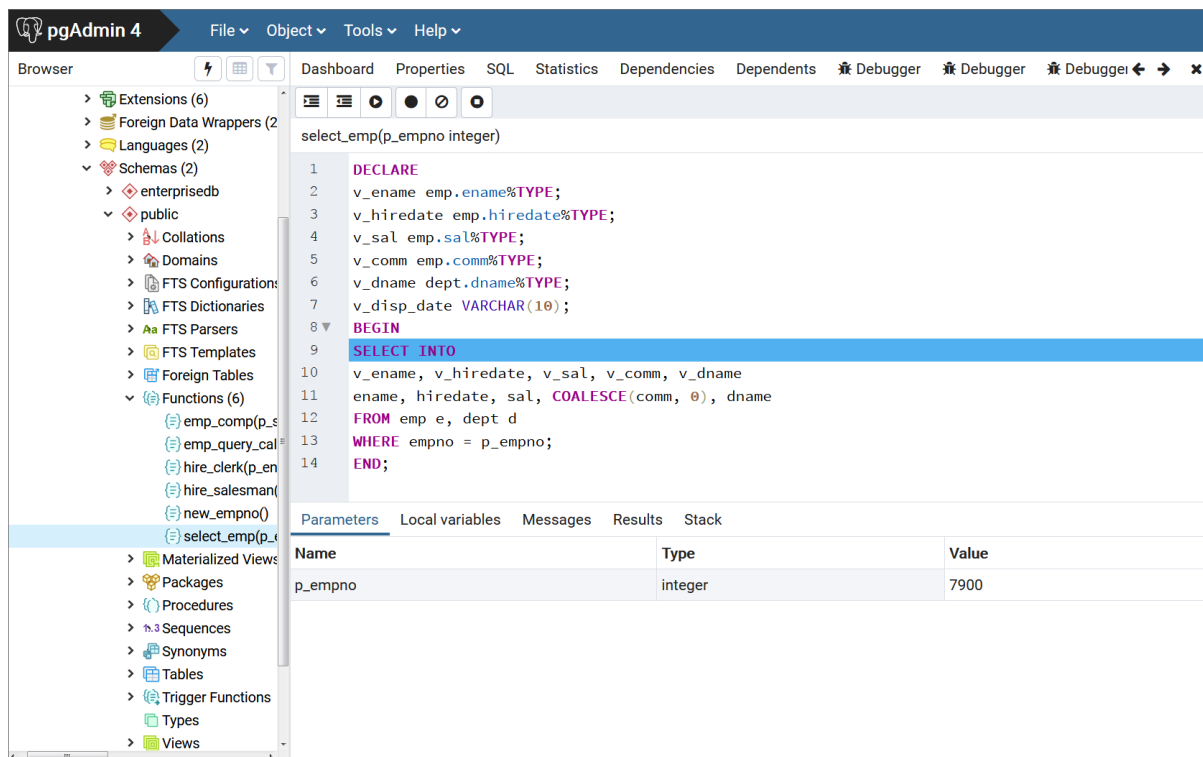


図 4: The Main Debugger window

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。

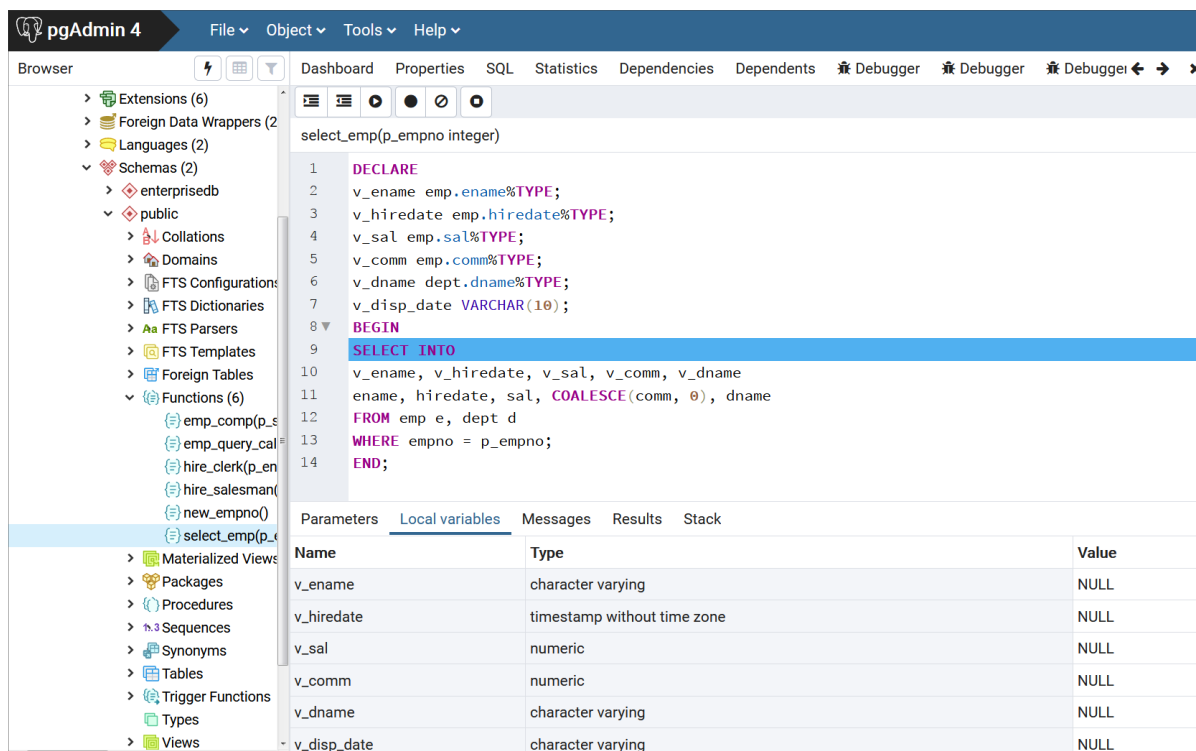


図 5: The Program Body

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いインジケータは、実行次のステートメントを強調表示します。

6.4.2 タブパネル

下部の Tabs パネルを使用して、パラメータ値またはローカル変数をビューまたは変更したり、**RAISE INFO** およびファンクションの結果によって生成されたメッセージをビューしたりできます。

以下は、パネルのタブによって表示される情報です。

- [Parameters] タブには、現在のパラメータ値が表示されます。
- [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る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のプログラム結果（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。

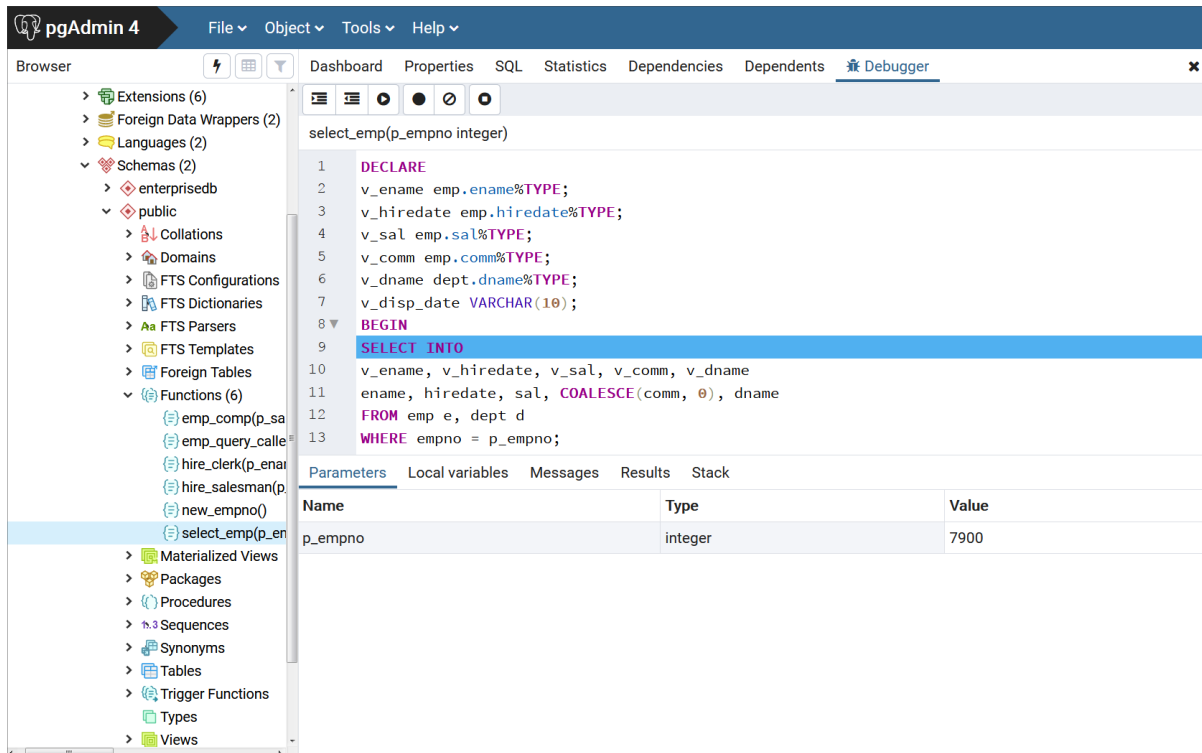


図 6: The Parameters tab

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

6.4.3 スタックタブ

</ div>

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。プログラムが呼び出されると、**Stack** タブに表示されるリストの上部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。次の情報が含まれます。

- プログラム内の呼び出しの場所
- 呼び出し回数
- 呼び出されるプログラムの名前

コールスタックを確認すると、シリーズのネストされたプログラムを介して実行コースを追跡するのにヘルプます。

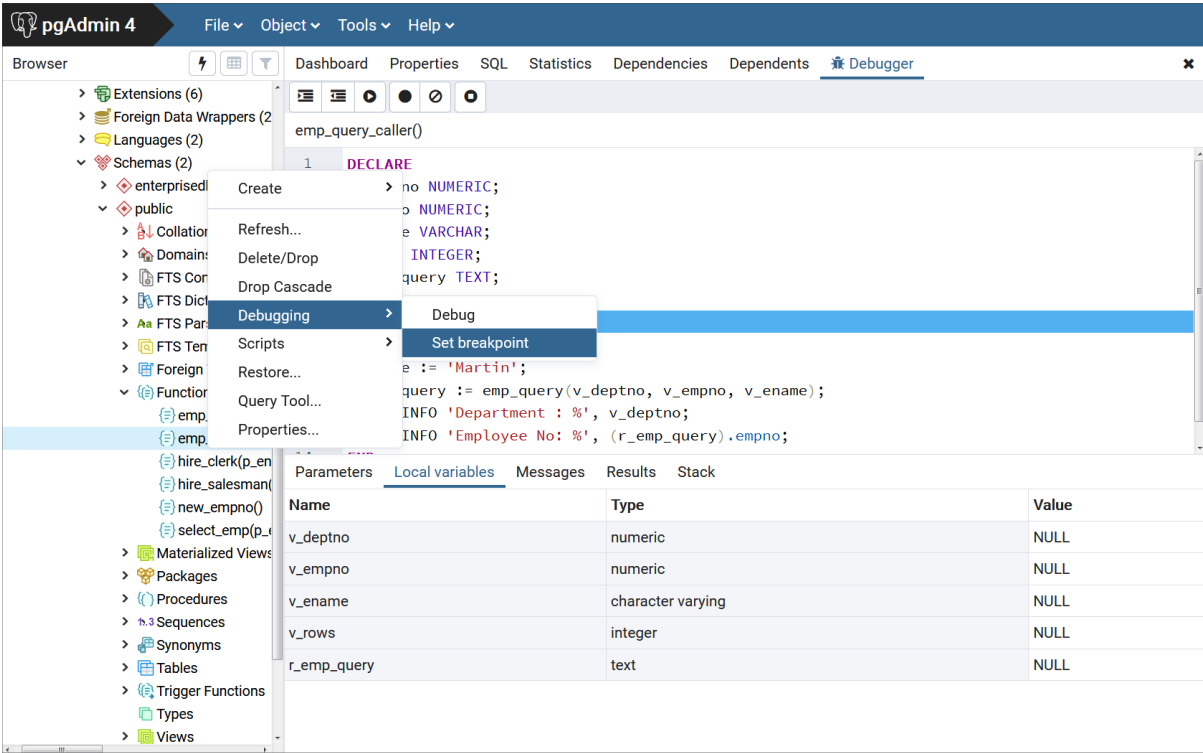


図 7: The Local variables tab

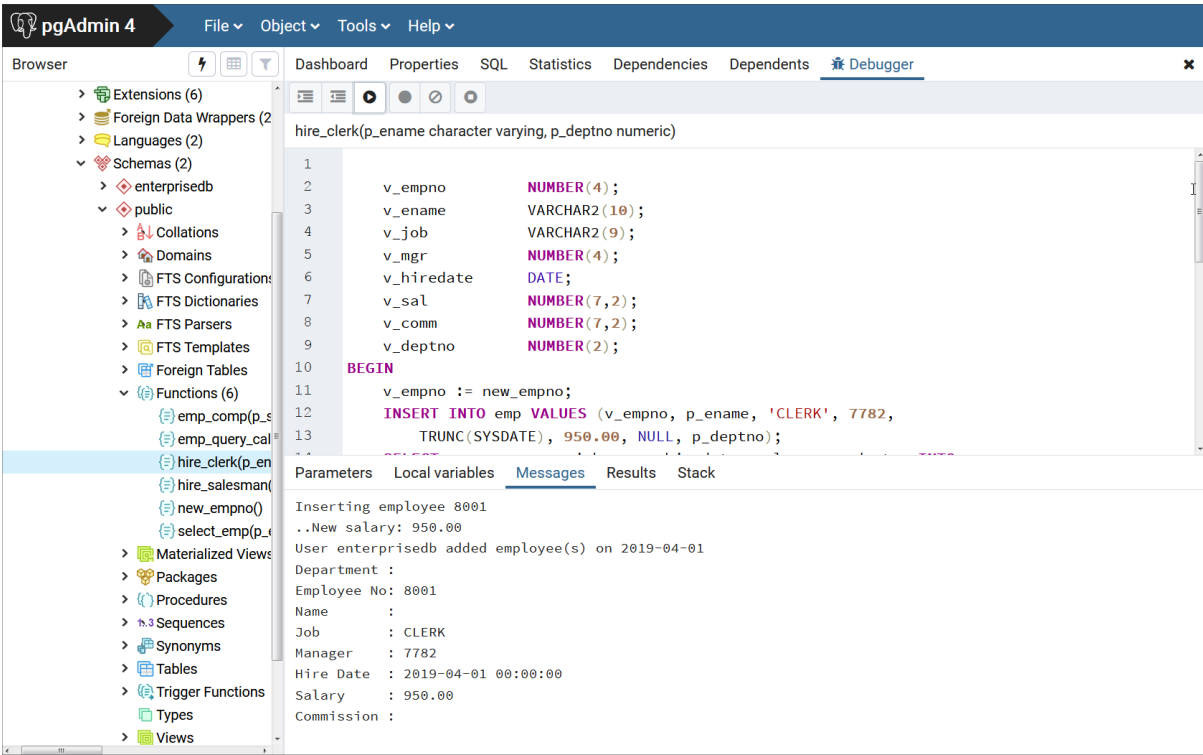


図 8: The Messages tab

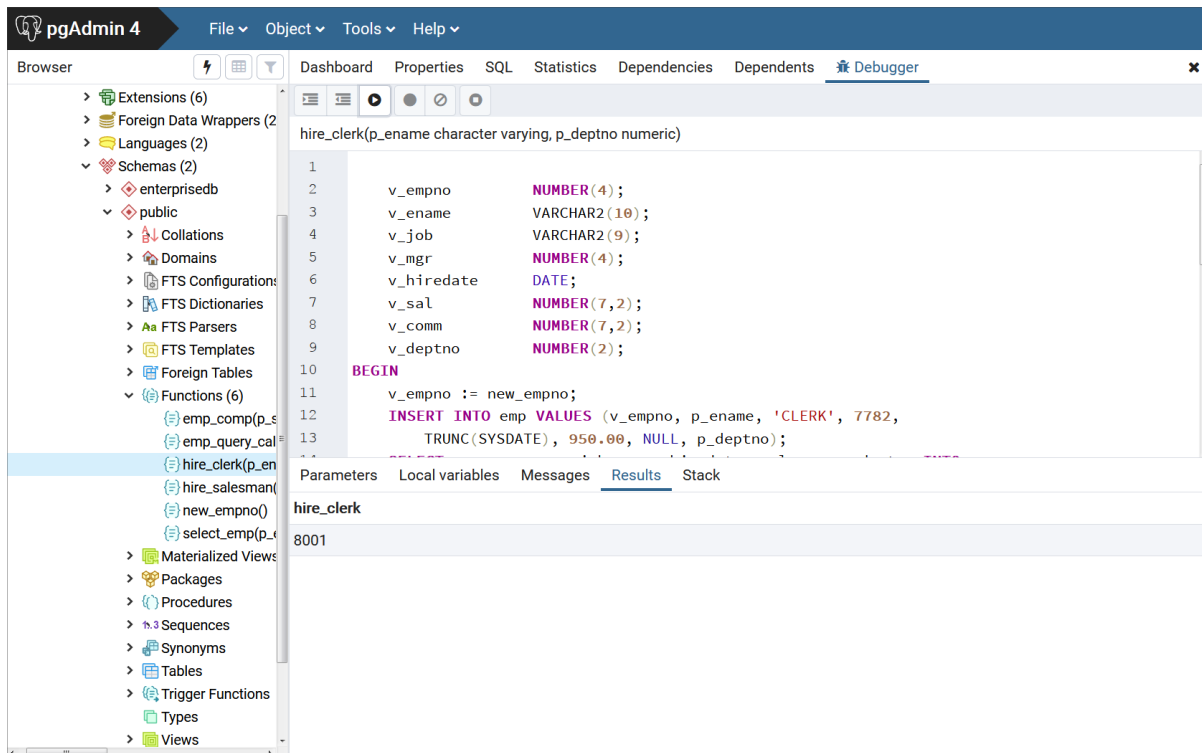


図 9: The Results tab

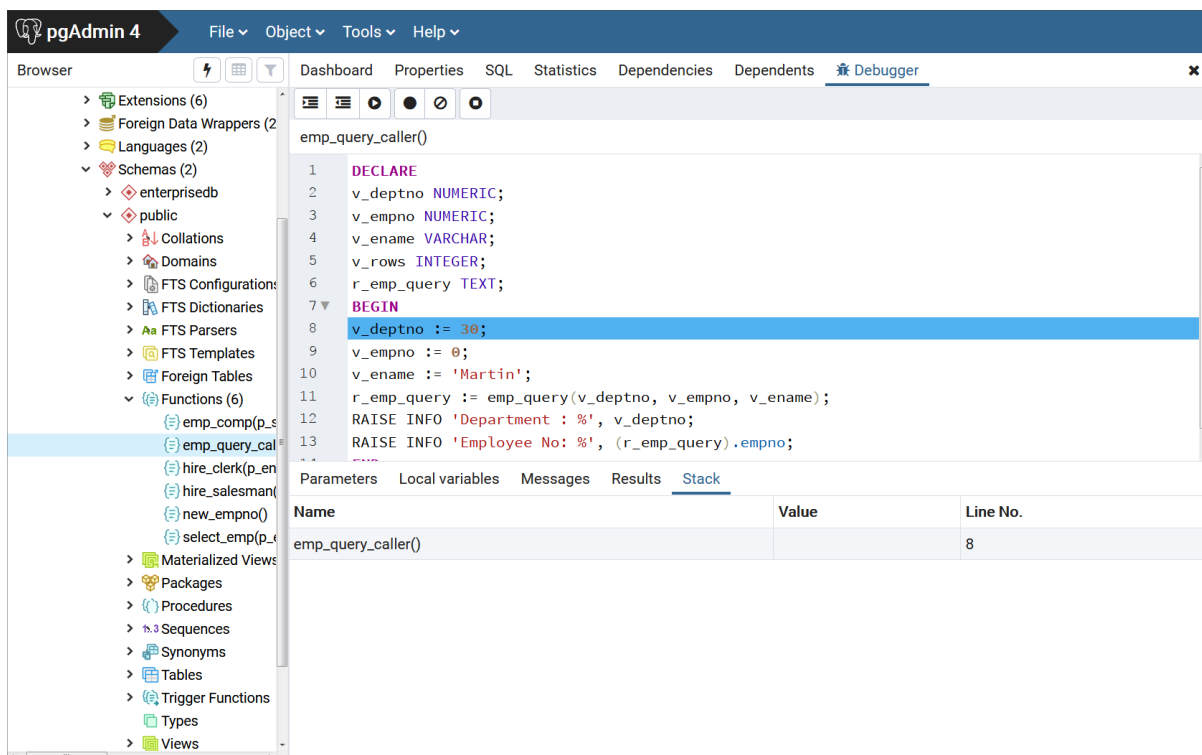


図 10: A debugged program calling a subprogram

図 10：サブプログラムを呼び出すデバッグプログラム

上の図は、`emp_query_caller` が `emp_query` 名前付け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、コールスタックの最上位にあります。

`emp_query` 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。

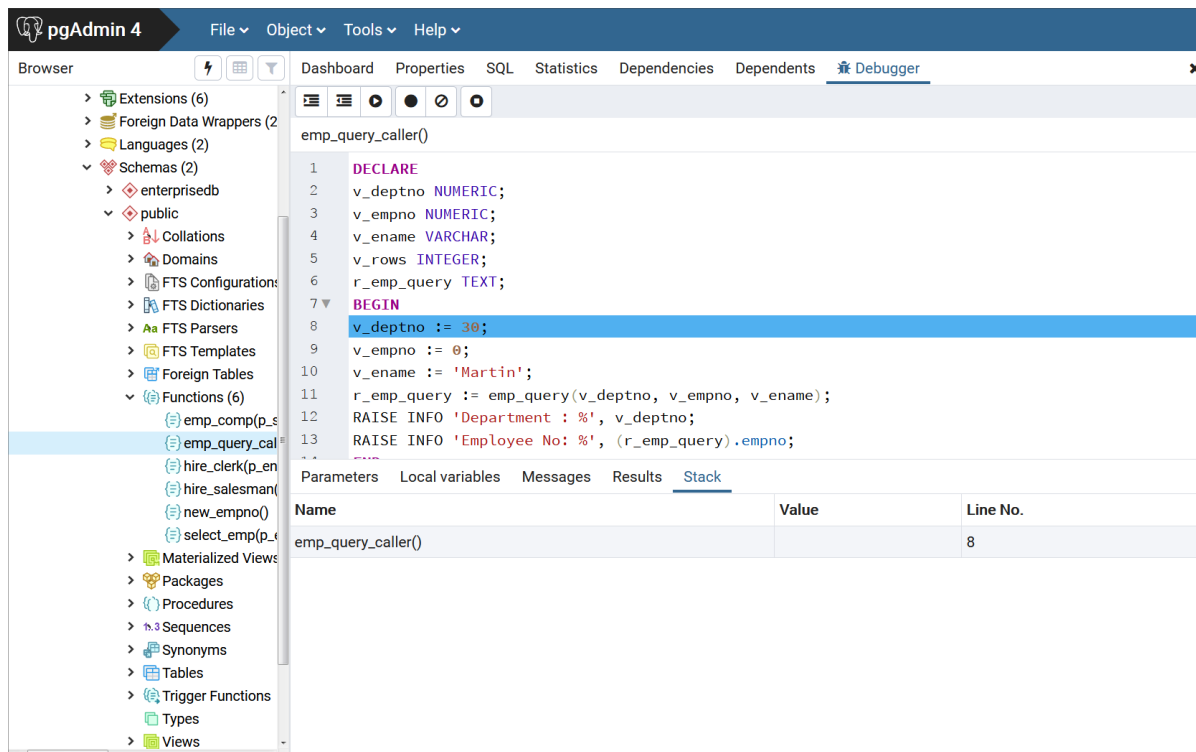


図 11: Debugging the called subprogram

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、制御は呼び出し側プログラム (`emp_query_caller`) に戻り、Stack タブの上部に表示されます。

図 12：デバッグされたサブプログラムからの制御の戻り値

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。

- 一度に 1 行ずつプログラムをステップ実行します
- ブレークポイントに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の表示と変更

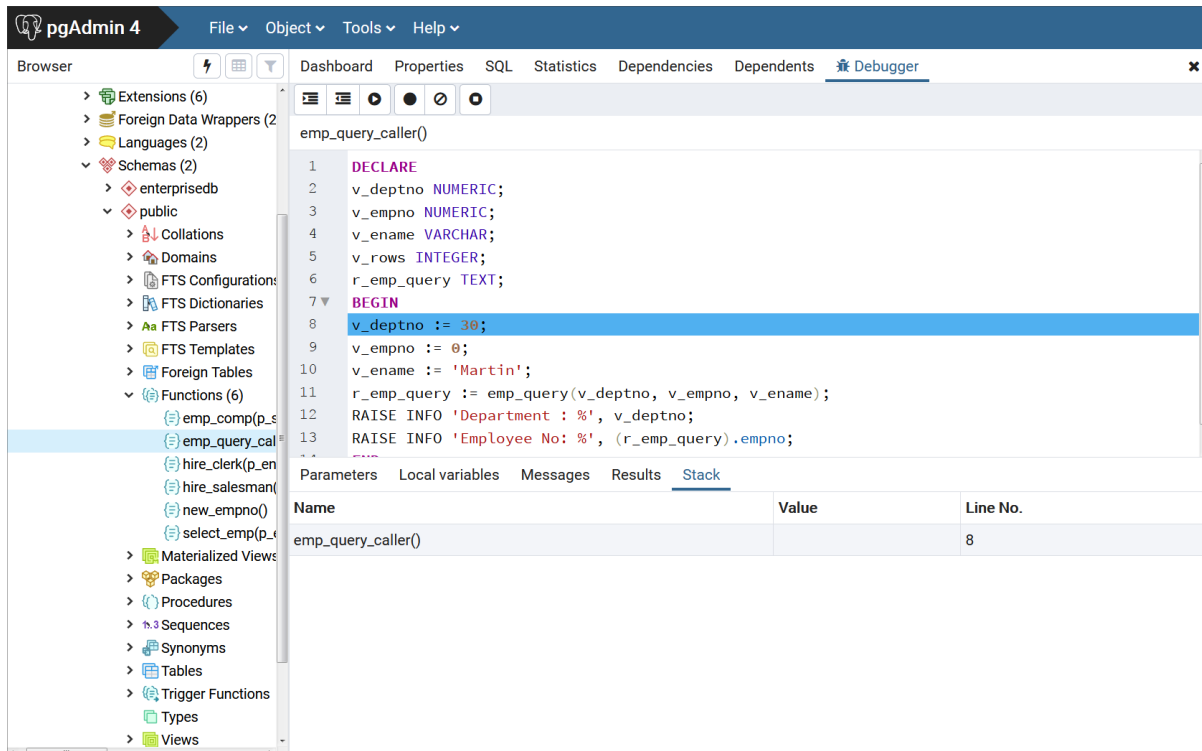


図 12: Control returns from debugged subprogram

6.5.1 コードのステップ実行

ツール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。

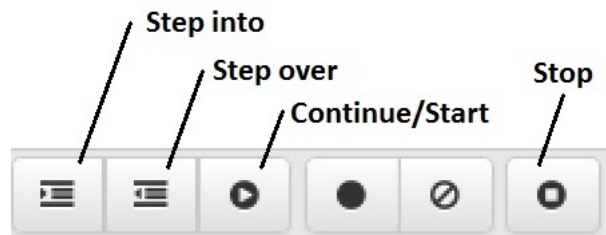


図 13: The Tool bar icons

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たサブファンクションをステップオーバーします。サブ関数は実行されますが、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 ブレークポイントを使用する

デバッガーが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すると、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ー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。操作への次のステップ、ステップオーバー、または継続オペレーションは、ブレークポイントに続くコードの次の行で実行をデバッガに強制します。ブレークポイントには2つのタイプがあります。

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-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内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。グローバルブレークポイントがプログラムに設定されると、グローバルブレークポイントを設定する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の行の左側にある灰色の影付きマージン内を左クリックします。グレーの影付きのマージンをクリックする場所は、ソースコード行 12 にブレークポイントドットが表示されるスポットのように、マージンの右側に近いはずで

作成されると、デバッガーはマージンに暗いドットを表示し、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。

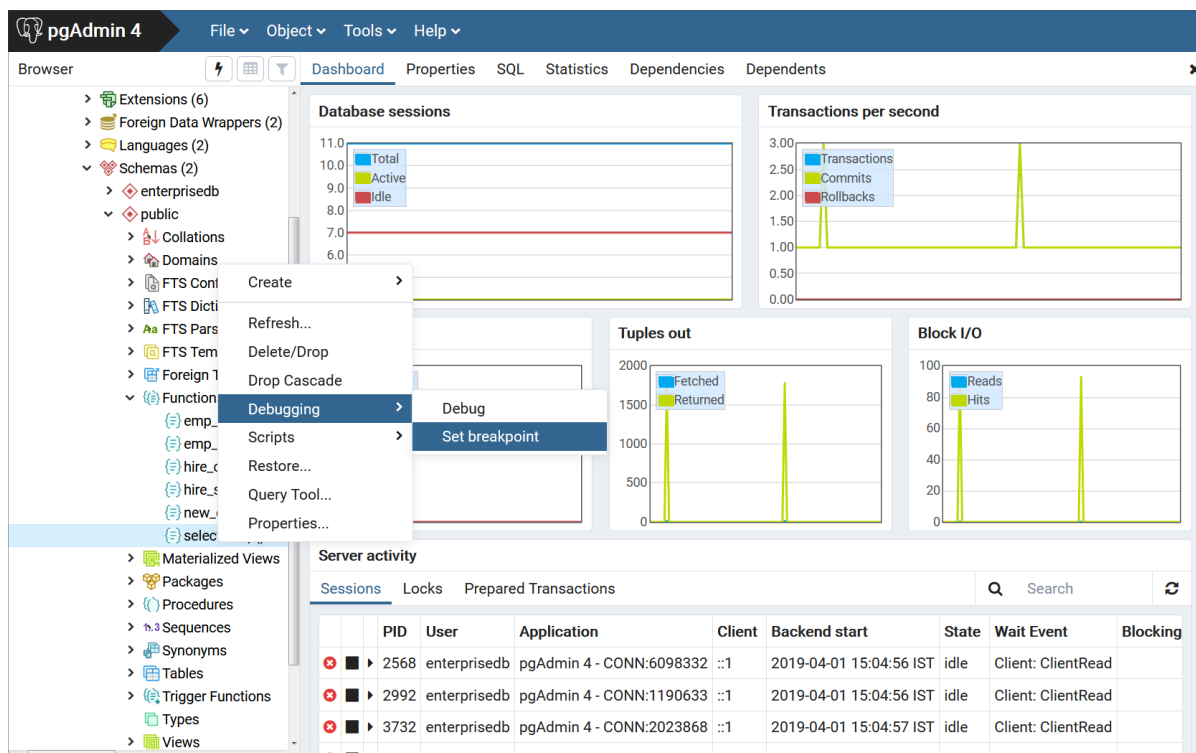


図 14: Set a breakpoint by clicking in left-hand margin

図 14：左マージンをクリックしてブレークポイントを設定

必要な数のローカルブレークポイントを設定できます。ローカルブレークポイントは、削除されるまでデバッグセッションの間有効です。

ローカルブレークポイントの削除

ローカルブレークポイントを削除するには、**Program Body** パネルのグレーの影付きマージンにあるブレークポイントドット上でマウスを左クリックします。ドットが消え、ブレー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ークポイントを削除できます。



図 15: Clear all breakpoints icon

図 15：すべてのブレークポイントアイコンをクリア

!!! Note * 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のブレー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のブレークポイント、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のブレークポイントは削除されません。

6.5.3 インコンテキストデバッグ用のグローバルブレークポイントの設定

</div>

インコンテキストデバッグ用のグローバルブレークポイントを設定するには、**Browser** パネルでブレー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] メニューからグローバルブレークポイントを設定

または、次のように、グローバルブレー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て、**Debugging** を選択し、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グローバルブレークポイントを設定

トリガーにグローバルブレー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関数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ー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および関数の場合と同じ指示に従います。

Set Breakpoint を選択すると、デバッガウィンドウが開き、アプリケーションがデバッグ対象のプログラムを呼び出すのを待ちます。

図 18：デバッグされるプログラムの呼び出しの待機

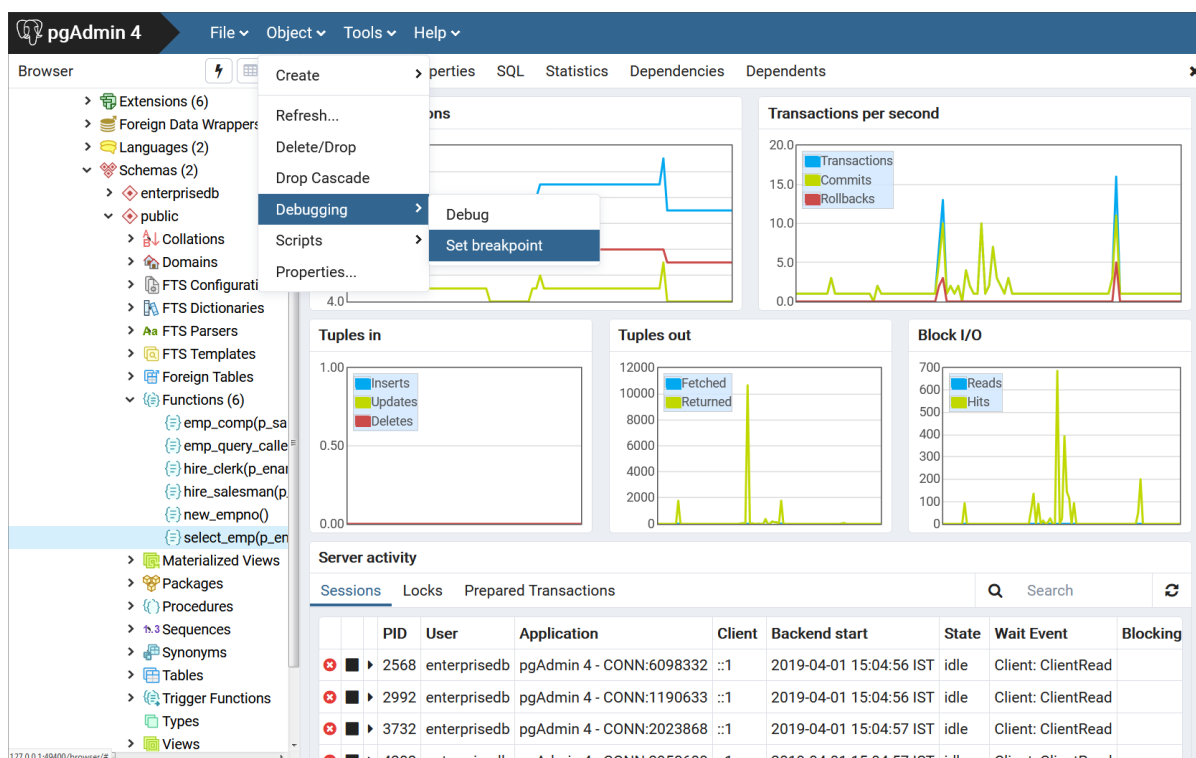


図 16: Setting a global breakpoint from the Object menu

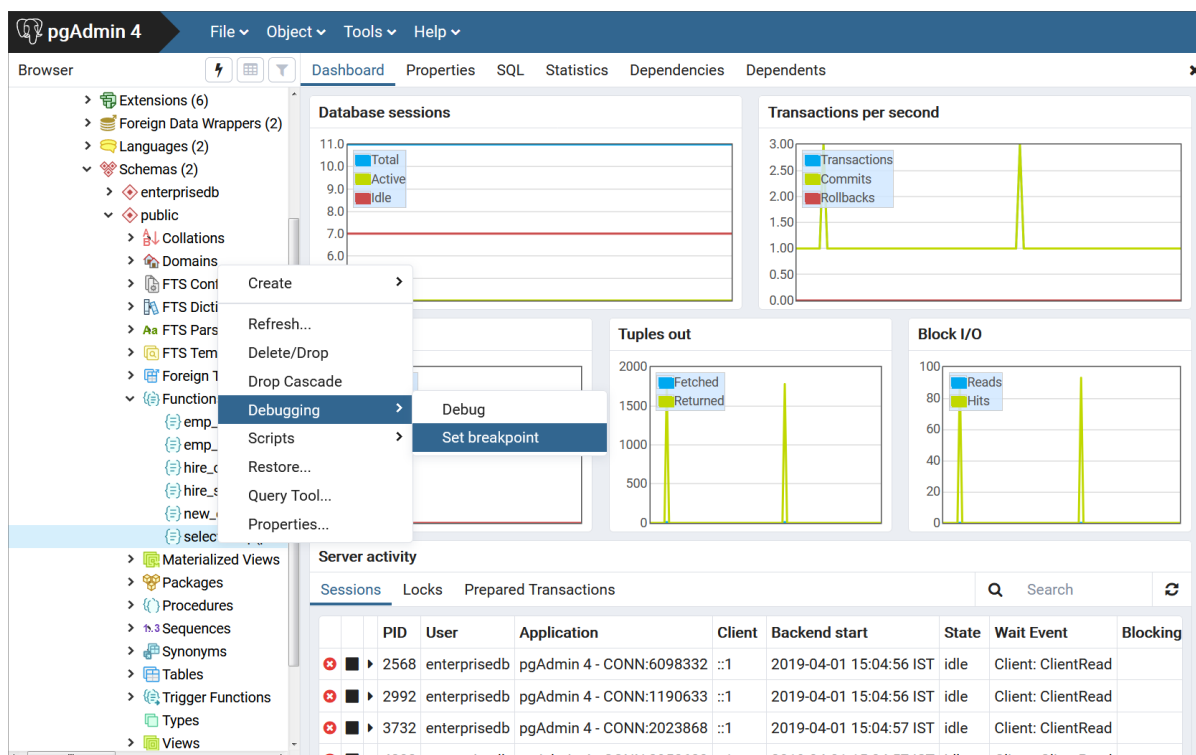


図 17: Setting a global breakpoint from the object's context menu

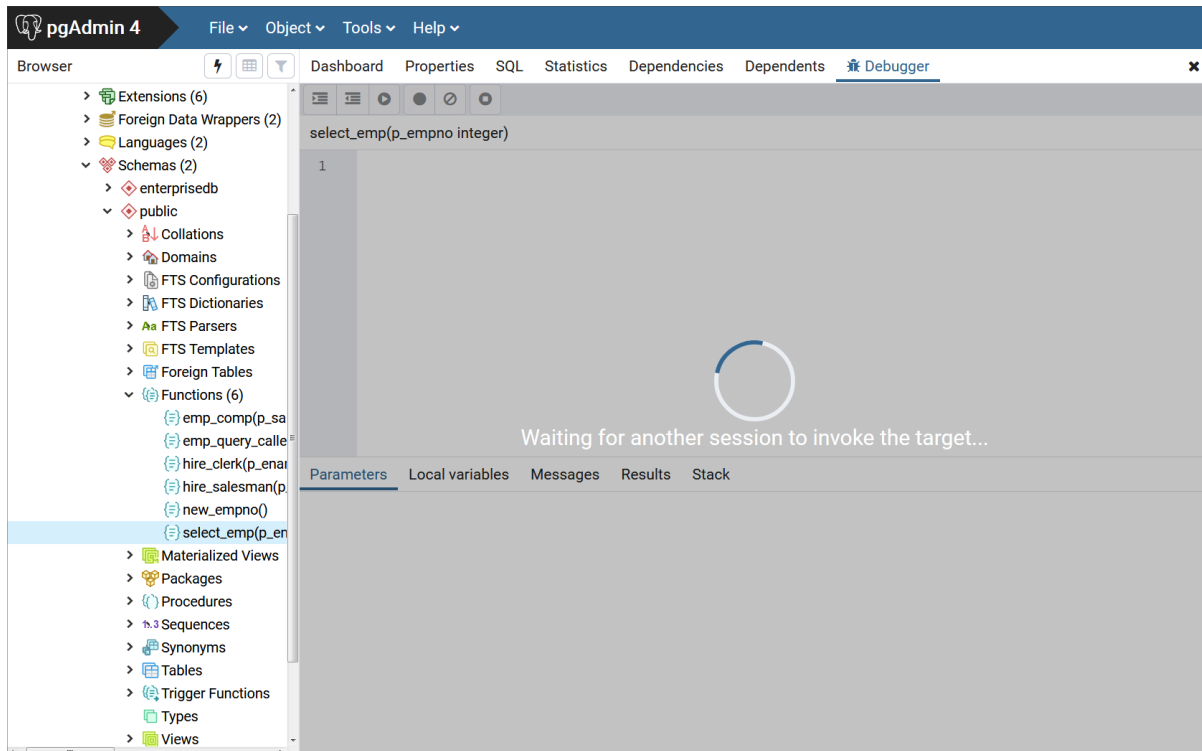


図 18: Waiting for invocation of program to be debugged

PSQL クライアントは、`select_emp` ファンクション（グローバルブレークポイントが設定されている）を呼び出します。

```
$ 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

これで、ステップイン、ステップオーバー、続行、ローカルブレークポイントの設定など、前述の操作のいずれか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実行すると、呼び出し側アプリケーション（PSQL）が制御を取り戻し、`select_emp` ファンクションの実行が完了し、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);
INFO: Number      : 7900
INFO: Name        : JAMES
```

(次のページに続く)

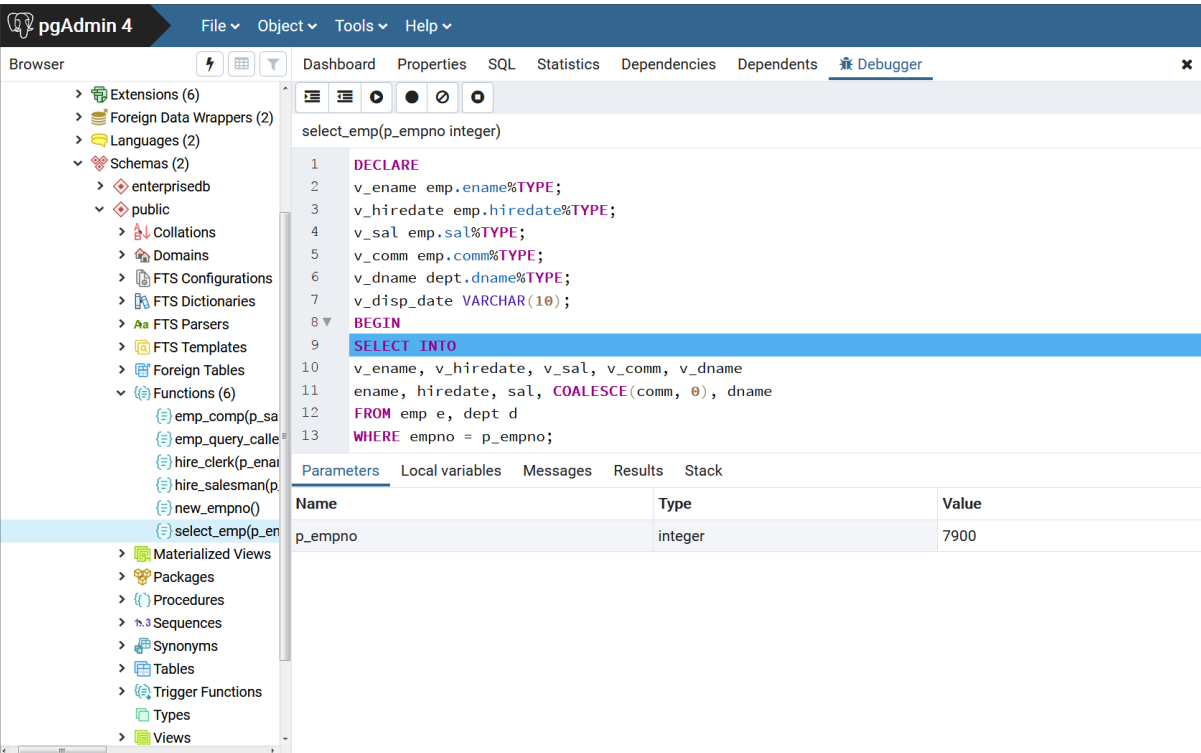


図 19: Program on which a global breakpoint has been set

(前のページからの続き)

```

INFO: Hire Date : 12/03/1981
INFO: Salary    : 950.00
INFO: Commission: 0.00
INFO: Department: SALES
      select_emp
-----
(1 row)

```

この時点で、デバッガセッションを終了できます。デバッガ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はグローバルブレイ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。

6.5.4 デバッガーの終了

デバッガ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

</div>

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

Advanced Server は、データベースサーバの動的チューニングをサポートしており、インストールされているホストマシンで利用可能なシステムリソースを最適に使用します。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`

</div>

7.1.1 edb_dynatune

`edb_dynatune` は、ホストマシンの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がホストシステムのリソースをどれだけ使用するかを決定します。

Advanced Server が最初にインストールされると、`edb_dynatune` パラメータは、インストールされたホストマシン（開発マシン、混在マシン、または専用サーバ）の選択された使用方法に従って設定されます。ほとんどの場合、パフォーマンスを改善オーダーに、データベース管理者が `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。

Advanced Server の初期インストール後に `postgresql.conf` ファイルを編集することにより、`edb_dynatune` パラメータの値を変更できます。新しい設定を有効にオーダーには、ポストマスターを再起動する必要があります。

`edb_dynatune` パラメータは、0 と 100 の間の任意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して、データベースサーバのリソース使用を `postgresql.conf` ファイル内の他の構成パラメーターの制御下に完全に残します。

ゼロ以外の低い値（1-33 など）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専用にします。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で使用されます。

34～66 のレンジの値は、適度な量のリソースをデータベースサーバ専用にします。この設定は、Advanced Server と同じマシンで実行されている他のアプリケーションの固定数を持つ専用アプリケーションサーバに使用される場合があります。

最も高い値（67～100 など）は、サーバーのリソースのほとんどをデータベースサーバ専用に使います。この設定は、Advanced Server の実行専用のホストマシンに使用されます。

`edb_dynatune` の値が選択されると、`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のパフォーマンスをさらに微調整できます。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。パラメータの値を変更するには、設定パラメータのコメントを外し、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。

</div>

7.1.2 edb_dynatune_profile

`edb_dynatune_profile` パラメータは、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するために使用されます。このパラメータは、データベースサーバのスタートアップに有効になります。

`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推奨されます。
<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で提供するサーバに推奨されます。
	NA

7.2 EDB 待機状態

- EDB 待機状態* 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。

**** 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。

セッションごとに、接続先のデータベース、セッションのログインユーザ、そのセッションで実行中のクエリー、待機セッションの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内の一連のファイルに保存されます。指定されたパスは、相対パスではなく完全な絶対パスでなければなりません。

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 待機状態は、`xx RPM` が Advanced Server バージョン番号である `edb-asxx-server-edb-modules RPM` パッケージとともにインストールされます。

ステップ 2: ワーカーを起動するには、`shared_preload_libraries` パラメータを使用して `postgresql.conf` ファイルに登録する必要があります。例を示します。

```
shared_preload_libraries = '$libdir/edb_wait_states'
```

ステップ 3: データベースサーバを再起動します。リスタートが成功すると、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、次の拡張子を作成します。

```
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` パラメータから `$libdir/edb_wait_states` を削除し、データベースサーバをリスタートします。

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、StackBuilder Plus ユーティリティを呼び出して、EDB Modules インストーラで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。

ステップ 2: ワーカーを登録するには、`postgresql.conf` ファイルを変更して、待機状態ライブラリを `shared_preload_libraries` 設定パラメータに含めます。パラメータ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 の待機状態では、インストールにより `edb_wait_states.dll` ライブラリファイルが次のパスに配置されます。

```
C:\Program Files\edb\as14\lib\
```

ステップ 3: データベースサーバを再起動して、変更を有効にします。リスタートに成功すると、バックグラウンドワーカーが開始され、データの収集が開始されます。

ステップ 4: データをビューするには、次の拡張子を作成します。

```
CREATE EXTENSION edb_wait_states;
```

インストーラは、`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
```

待機状態ワーカーの終了

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` 拡張機能をドロップします。次に `postgresql.conf` ファイルを変更して、`shared_preload_libraries` パラメータから `$libdir/edb_wait_states.dll` を削除します。`postgresql.conf` ファイルを変更して変更を適用した後、データベースサーバを再起動します。

待機状態インターフェイス

このインタフェースには、以下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。これらの各機能には、共通の入力および出力パラメータがあります。これらのパラメータは次のとおりです。

- **** start_ts および end_ts (IN)。** ****** これら是一緒に、時間インターバルと読み取り対象のデータを指定します。start_ts のみが指定されている場合、start_ts から始まるデータが出力されます。end_ts

のみが提供される場合、end_ts までのデータが出力されます。これらのいずれも提供されない場合、すべてのデータが出力されます。すべてのファンクションは異なるデータを出力します。以下に、各ファンクションの出力について説明します。

- ** query_id (OUT)。**正規化されたクエリーを識別します。これは、クエリー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、待機イベントサンプルレコード、クエリーレコード、またはセッションレコードです。

各ファンクションの構文については、次のセクションで説明します。

!!! Note * 次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同時に接続された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';
```

7.2.1 edb_wait_states_data

このファンクションは、EWSBW によって収集され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,
  * OUT <wait_event> text
)
```

このファンクションを使用して、以下を見つけることができます。

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) で実行されているクエリ、および待機している待機イベント (存在する場合)。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、クエリはセッション（session_id = 100000）で実行され、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();
```

パラメータ

前に説明した共通パラメーターに加えて、出力の行は次の情報を提供します。

dbname

セッションのデータベース

username

セッションのログインユーザ

query

セッションで実行されているクエリー

query_start_time

クエリーが開始された時間

sample_time

待機イベントデータが収集された時間

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

例

以下は、edb_wait_states_data() ファンクションのサンプル出力です。

```
edb=# SELECT * FROM edb_wait_states_data();
- [ RECORD 1 ]-----+-----
session_id      | 4398
```

(次のページに続く)

(前のページからの続き)

```

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
*               .
*               .
*               .

```

7.2.2 edb_wait_states_queries

このファンクションは、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,

```

(次のページに続く)

(前のページからの続き)

```

* 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マルチプルのクエリーファイルが生成される場合があります。

このファンクションは、指定された時間インターバルと重複するクエリーファイル内のすべてのクエリを返します。以下に示すクエリーは、`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ーファイル内のすべてのクエリを提供します。

つまり、ファンクションは、指定されたインターバル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);
```

パラメータ

前に説明した共通パラメーターに加えて、出力の行は次の情報を提供します。

`query`

正規化されたクエリテキスト

例

以下は、`edb_wait_states_queries()` ファンクションのサンプル出力です。

```

edb=# SELECT * FROM edb_wait_states_queries();
- [ 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 ]+-----
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

```

7.2.3 edb_wait_states_sessions

このファンクションは、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 timestamptz default '-infinity'::timestamptz,
  * IN end_ts timestamptz default 'infinity'::timestamptz,
  * OUT session_id int4,
  * OUT <dbname> text,
  * OUT <username> text,
  * OUT ref_start_ts timestamptz
  * OUT ref_end_ts timestamptz
)
```

このファンクションを使用して、接続されたデータベースや、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、指定されたインターバル内でサンプリングされたセッションだけでなく、指定されたインターバル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。edb_wait_states_data() を使用する必要があることを識別するには。

パラメータ

前に説明した共通パラメーターに加えて、出力の行は次の情報を提供します。

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

例

以下は、edb_wait_states_sessions() ファンクションのサンプル出力です。

```
edb=# SELECT * FROM edb_wait_states_sessions();
- [ 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
```

(次のページに続く)

(前のページからの続き)

```

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

```

7.2.4 edb_wait_states_samples

このファンクションは、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、ユーザはこのファンクションを直接呼び出す必要はありません。

パラメータ

前に説明した共通パラメーターに加えて、出力の行は次の情報を提供します。

query_start_time

このセッションでクエリーが開始された時刻

sample_time

待機イベントデータが収集された時間

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

例

以下は、edb_wait_states_samples() ファンクションのサンプル出力です。

```
edb=# SELECT * FROM edb_wait_states_samples();
- [ 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
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 11:56:02.700001 -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
*                  .
*                  .
*                  .
```

7.2.5 edb_wait_states_purge

このファンクションは、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されたデータファイル（クエリ、セッション、および待機イベントのサンプル）を削除します。

```
edb_wait_states_purge(
    * IN start_ts timestamptz default '-infinity'::timestamptz,
    * IN end_ts timestamptz default 'infinity'::timestamptz
)
```

通常、ユーザはこのファンクション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応じてそれらを削除する必要がありますが、何らかの理由でそれが発生しない場合は、このファンクションを使用できます。

サンプルが保持されている期間を知るには、そのファンクションの前の例で説明したように edb_wait_states_data() を使用します。

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() の実行後の \$PGDATA/edb_wait_states ディレクトリ：

```
edb=# SELECT * FROM edb_wait_states_purge();
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

</ div>

- EDB Clone Schema *は Advanced Server の拡張モジュールであり、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ます。

ソースデータベースとターゲットデータベースは、同じ物理的データベースでも、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、異なるデータベースサーバホスト上の異なるデータベースクラスターの下で実行されている個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関数を使用します。

- **** localcopyschema. ****このファンクションは、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、オリジナルのスキーマ名とは異なるスキーマ名前でスキーマとそのデータベースオブジェクトのコピーを作成します。このファンクションは、オリジナルのソーススキーマと結果のコピーが同じデータベース内にある場合に使用します。localcopyschema ファンクションの詳細については、[localcopyschema](#) を参照してください。
- **** localcopyschema_nb. ****このファンクションは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実行されるため、ファンクションが開始されたターミナルを解放します。これは* non-blocking *ファンクションと呼ばれます。localcopyschema_nb ファンクションの詳細については、[localcopyschema_nb](#) を参照してください。
- **** remotecopyschema. ****このファンクションは、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。オリジナルのソーススキーマと結果のコピーが2つの別個のデータベースに存在する場合、このファンクションを使用します。個別のデータベースは、同じまたは異なる Advanced Server データベースクラスターに配置できます。remotecopyschema ファンクションの詳細については、[remotecopyschema](#) を参照してください。
- **** remotecopyschema_nb. ****このファンクションは、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実行されるため、ファンクションが開始されたターミナルを解放します。これは* non-blocking *ファンクションと呼ばれます。remotecopyschema_nb ファンクションの詳細については、[remotecopyschema_nb](#) を参照してください。
- **** process_status_from_log. ****このファンクションは、クローン作成関数のステータスを表示します。この情報は、クローン作成ファンクションを呼び出すときに指定する必要があるログファイルから取得されます。process_status_from_log ファンクションの詳細については、[process_status_from_log](#) を参照してください。
- **** remove_log_file_and_job. ****このファンクションは、クローニングファンクションによって作成されたログファイルを削除します。このファンクションは、非ブロッキングフォームのファンクションによって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 ファンクションの詳細については、[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
- 外部テーブルを除く、テーブルパーティションを含むテーブル
- インデックス
- Constraints
- シーケンス
- 定義の表示
- マテリアライズドビュー
- プライベートシノニム
- 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
- Functions
- Procedures
- 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に対するコメント
- 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複製については、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
- 演算子クラス

さらに、次の制限が適用されます。

EDB Clone Schema は、インストール中に **Compatible with Oracle** の方言が **Advanced Server Dialect** ダイアログで指定されている場合、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の初期化中に **--redwood-like** キーワードが含まれている場合にのみ **Advanced Server** でサポートされます。 - 関数、プロシージャ、トリガー、パッケージソフトなどのソースコードは、ターゲットスキーマにコピーされた後

は変更されません。そ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化された参照が含まれる場合、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なくなった場合、ターゲットスキーマでの呼び出し時にプログラムが失敗することがあります。- クロススキーマオブジェクトの依存関係は解決されません。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ン作成機能によって解決されません。- 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、リモートデータベースのパブリックスキーマにこの拡張機能を作成する必要があります。- 最大 16 個のコピージョブがスキーマを複製するために並行して実行できますが、各ジョブはテーブルデータを並行してコピーするために最大 16 個のワーカープロセスを持つことができます。- 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 設定プロセス

PL / Perl 言語に加えて、EDB クローンスキーマファンクションによってソースデータベースまたはターゲットデータベースとして使用するデータベースにいくつかの拡張機能をインストールする必要があります。

また、データベースサーバーの `postgresql.conf` ファイル内の一部の構成パラメーターは、一部の変更から恩恵を受ける場合があります。

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

以下に、必要な拡張機能と PL / Perl 言語をインストールする手順を示します。

これらの手順は、EDB クローンスキーマファンクションによってソースデータベ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てください。Linux では、`edb-asxx-pgagent` RPM を使用できます。`xx` は Advanced Server のバージョン番号で、`pgAgent` をインストールします。Windows、StackBuilder Plus を使用して `pgAgent` をダウンロードおよびインストールします。

前述の拡張機能が存在しない場合は、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 コア文書を参照してください。

< [https://www.postgresql.org/docs/current/ 静的/sql-createextension.html](https://www.postgresql.org/docs/current/静的/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 設定パラメータに\$libdir/parallel_clone を追加して、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。xx は 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 言語パックの詳細については、次から入手できる*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 コア文書を参照してください。

< [https://www.postgresql.org/docs/current/ 静的/sql-createlanguage.html](https://www.postgresql.org/docs/current/静的/sql-createlanguage.html) >

</ div>

8.1.2 設定パラメータの設定

以下のセクションでは、postgresql.conf ファイルで変更する必要がある可能性のある構成パラメーターについて説明します。

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パートとしてラージスキーマをコピーするために、システムを調整する必要がある場合があります。

構成パラメータのチューニングは、クローン作成ファンクションで被参照されるソースデータベースサーバー用です。

postgresql.conf ファイル内の調整が必要な構成パラメータには、次のものが含まれます。

- **work_mem**。一時ディスクファイルに書き込む前に、内部ソート操作とハッシュテーブルで使われるメモリ量を指定します。
- **maintenance_work_mem**。VACUUM、CREATE INDEX、ALTER TABLE ADD FOREIGN KEYなどのメンテナンス操作で使われ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一部として、チェックポイント完了のターゲットを指定します。
- **チェックポイント**。チェックポイントの実行中に **checkpoint_flush_after** * bytes * 以上が書き込まれた場合は常に、OS がこれらの書き込みを基本ストレージに強制的に発行するように試みてください。
- **max_wal_size**。WAL が自動 WAL チェックポイント間で成長できる最大サイズ。
- **max_locks_per_transaction**。このパラメータは、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 コア文書を参照してください。

< <https://www.postgresql.org/docs/current/ 静的/runtime-config.html> >

ステータスロギング

クローン作成機能によるステータスロギングは、クローン作成ファクションの呼び出し時に接続しているデータベースサーバの `postgresql.conf` ファイルの `log_directory` パラメータ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、次のように 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、クローン作成ファクション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、クローン作成ファクションを呼び出すときにパラメータリストで指定した内容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` ファクションを使用します。

ログファイルを削除するには、`remove_log_file_and_job` ファクションを使用するか、単にログディレクトリに移動して手動で削除します。

8.1.3 EDB クローンスキーマのインストール

EDB クローンスキーマのインストール手順は次のとおりです。

これらの手順は、EDB クローンスキーマファンクションによってソースデータベ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 `edb_cloneschema` 拡張機能の古いバージョンをインストールしていた場合は、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` 拡張子も削除します。

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、`parallel_clone` 拡張機能を作成してください。

8.1.4 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の 1 つである `localcopyschema` または `localcopyschema_nb` を使用する場合、必要なパラメーターの 1 つには、クローンサーバーのソースおよびレシーバであるデータベースと共にデータベースサーバを識別する単一の外部サーバが含まれスキーマ。

リモートクローニング関数の 1 つである `remotecopyschema` または `remotecopyschema_nb` を使用する場合、2 つの必須パラメーターには 2 つの外部サーバーが含まれます。最初のパラメータとして指定された外部サーバは、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共にソースデータベースサーバを識別します。2 番目のパラメータとして指定された外部サーバは、クローンスキーマのレシーバであるデータベースとともにターゲットデータベースサーバを識別します。

外部サーバごとに、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がクローン作成ファンクションを呼び出す場合、その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スーパーユーザは、クローン作成ファンクションのパラメータとして指定された外部サーバにアクセスできるデータベースユーザ名前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情報については、次の PostgreSQL コア文書を参照してください。

< <https://www.postgresql.org/docs/current/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の定義方法について説明します。

ローカルクローニング Functions のための外部サーバーおよびユーザーマッピング

localcopyschema および localcopyschema_nb 関数の場合、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は 1 つだけです。この外部サーバは、**ローカルサーバ**とも呼ばれます。

このサーバは 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、このサーバはローカルサーバと呼ばれます。

ユーザマッピングは、外部サーバの接続および認証情報を定義します。

この外部サーバとユーザマッピングは、クローンを作成するローカルサーバのデータベース内に作成する必要があります。

ユーザマッピングが定義されるデータベースユーザはスーパーユーザであり、EDB クロンスキーマファンクションを呼び出すときにローカルサーバに接続しているスーパーユーザでなければなりません。

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を作成し、複製したスキーマも受け取ります。

```
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 コア文書を参照してください。

< <https://www.postgresql.org/docs/current/sql-createserver.html> >

このサーバのユーザ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 コア文書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、外部サーバとユーザマッピングを表示します。

```
edb=# \des+
List of foreign servers
- [ RECORD 1 ]-----+-----
Name                | local_server
Owner                | enterprisedb
```

(次のページに続く)

```

Foreign-data wrapper | postgres_fdw
Access privileges    |
Type                 |
Version              |
FDW options           | (host 'localhost', port '5444', dbname 'edb')
Description          |

edb=# \deu+
*
          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 **enterprisedb** がクローン作成ファンクションを呼び出すと、データベースユーザ **enterprisedb** とそのパスワードを使用して、**localhost** の **local_server** に接続し、ポート 5444 からデータベース **edb** に接続します。

この場合、ローカルの **edb** データベースに接続するために使用される、マップされたデータベースユーザ **enterprisedb** とデータベースユーザ **enterprisedb** は、偶然同じ同一のデータベースユーザですが、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とユーザマッピングの例の特定の使用方法については、**localcopyschema** にある例を参照してください。

リモートクローニング Functions のための外部サーバーとユーザーマッピング

remotecopyschema および **remotecopyschema_nb** 関数の場合、ソーススキーマとターゲットスキーマは、同じデータベースサーバー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、これらの機能に対して 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。

複製元のデータベースサーバと、複製されるソーススキーマを含むデータベースを定義する外部サーバは、**ソースサーバー**または**リモートサーバ**と呼ばれます。

クローンを作成するスキーマを受け取るデータベースサーバとそのデータベースを定義する外部サーバは、**ターゲットサーバー**または**ローカルサーバ**と呼ばれます。

ターゲットサーバーは、ローカルサーバとも呼ばれます。これは、このサーバーが **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サーバーだからです。

ユーザマッピングは、外部サーバーの接続および認証情報を定義します。

これらの外部サーバーとユーザマッピングはすべて、ターゲット/ローカルサーバ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。

ユーザマッピングが定義されるデータベースユーザはスーパーユーザであり、EDB クローンスキーマファンクションを呼び出すときにローカルサーバに接続しているスーパーユーザでなければなりません。

次の例では、クローン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、クロンスキーマのソースとなるリモートソースデータベースの外部サーバを作成します。

```
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マッピングを表示します。

```
tgtdb=# \des+
List of foreign servers
- [ RECORD 1 ]-----+-----
Name                | src_server
Owner                | tgtuser
Foreign-data wrapper | postgres_fdw
Access privileges    |
Type                 |
Version              |
FDW options           | (host '192.168.2.28', port '5444', dbname 'srcdb')
Description          |
- [ RECORD 2 ]-----+-----
```

(次のページに続く)

(前のページからの続き)

```

Name          | tgt_server
Owner         | tgtuser
Foreign-data wrapper | postgres_fdw
Access privileges |
Type          |
Version       |
FDW options   | (host 'localhost', port '5444', dbname 'tgtdb')
Description   |

```

```
tgtdb=# \deu+
```

```

*
          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 **enterprisedb** がクローン作成ファンクションを呼び出す場合、パスワード **tgtpassword** のデータベースユーザ **tgtuser** を使用して、**localhost** の **tgt_server** に接続し、ポート **5444** からデータベース **tgtdb** に接続します。

さらに、パスワード **srcpassword** のデータベースユーザ **srcuser** は、ポート **5444** を使用してホスト **192.168.2.28** の **src_server** に接続し、データベース **srcdb** に接続します。

!!! Note * 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の **pg_hba.conf** ファイルに、外部のユーザマッピングに含まれているデータベースユーザ **srcuser** に接続するターゲットサーバの場所 (次の例ではアドレス **192.168.2.27**) からの接続を許可する適切なエントリーがあることを確認してくださいソースサーバとデータベースを定義する外部サーバ **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とユーザマッピングの例の特定の使用方法については、**remotecopyschema** の例を参照してください。

8.2 EDB クローンスキーマ Functions

`parallel_clone` および `edb_cloneschema` 拡張機能がインストールされると、EDB クローンスキーマ関数が `edb_util` スキーマに作成されます。

EDB クローンスキーマファンクションを使用する前に、次の条件を確認してください。

- ターゲットデータベースまたはローカルデータベースの外部サーバの `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として、ターゲットデータベースまたはローカルデータベースに接続している。
- `edb_util` スキーマが検索パスにあるか、または `edb_util` プレフィックスを使用してクローン作成ファンクションが呼び出されます。
- ターゲットスキーマがターゲットデータベースに存在しません。
- リモートコピー機能を使用する場合、`on_tblspace` パラメータを `true` に設定する場合、ターゲットデータベースクラスタには、ソーススキーマ内のオブジェクトによって被参照されるすべてのテーブルスペースが含まれます。そうしないと、こ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が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` パラメータを `true` に設定すると、ソーススキーマ内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に存在します。そうしないと、これらのロールへの特権の付与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。
- 非ブロッキングフォームのファンクションを使用する場合、`pgAgent` はターゲットデータベースに対して実行されます。

`pgAgent` の詳細については、次の場所にある `pgAdmin` の文書を参照してください。

<https://www.pgadmin.org/docs/pgadmin4/dev/pgagent.html>

`pgAgent` は 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 localcopyschema

</ div>

`localcopyschema` ファンクションは、`source_fdw` 外部サーバ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
  * [, <on_tblspace> BOOLEAN
  * [, <verbose_on> BOOLEAN
```

(次のページに続く)

```

* [, <copy_acls> BOOLEAN
* [, <worker_count> INTEGER ]]]]
)

```

関数によって **BOOLEAN** 値が返されファンクション。ファンクションが成功すると、**true** が返されます。ファンクションが失敗すると、**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。

パラメータ

source_fdw

データベースオブジェクトのクローン元となる **postgres_fdw** 外部データラップによって管理される外部サーバの名前。

source_schema

データベースオブジェクトの複製元のスキーマの名前。

target_schema

データベースオブジェクトがソーススキーマから複製されるスキーマの名前。

log_filename

ファンクション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** 設定パラメータ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。データベースオブジェクトをテーブルスペース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るときに、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** パラメータが省略された場合、デフォルト値は **false** です。

verbose_on

BOOLEAN 値。ターゲットスキーマにオブジェクトを作成するときに **log_filename** に **DDL** を出力するかどうかを指定します。**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、デフォルト値は **false** です。

copy_acls

BOOLEAN 値。ターゲットスキーマに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** 権限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、アクセス制御リストはターゲットスキーマに含まれます。**copy_acls** パラメータが省略された場合、デフォルト値は **false** です。

worker_count

クローンを並行して実行するバックグラウンドワーカーの数。省略した場合、デフォルト値は 1 です。

例

次の例は、`local_server` で定義されたデータベース `edb` 内の両方のターゲットスキーマ `edbcopy` へのデータベースオブジェクトのセットを含むスキーマ `edb` のクローン作成を示しています。

この例は、次の環境用です。

- データベースサーバが実行されているホスト：`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`
- 外部サーバー (`local_server`) および先行する簡条書きの情報を使用したユーザマッピング
- ソーススキーマ：`edb`
- ターゲットスキーマ：`edbcopy`
- `localcopyschema`: `enterprisedb` を呼び出すデータベーススーパーユーザ

ファンクションを呼び出す前に、データベースユーザ `enterprisedb` がデータベース `edb` に接続します。

```
edb=# SET search_path TO "$user",public,edb_util;
SET
edb=# SHOW search_path;
      *
      search_path
-----
"$user", public, edb_util
(1 row)

edb=# SELECT localcopyschema ('local_server','edb','edbcopy','clone_edb_edbcopy');
 localcopyschema
-----
t
(1 row)
```

以下は、`process_status_from_log` ファンクション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');
      *
      process_status_from_log
-----
→-----
(FINISH,"2017-06-29 11:07:36.830783-04",3855,INFO,"STAGE: FINAL","successfully
→cloned schema")
(1 row)
```

クローンの完了後、次の図は、`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示しています。

```
edb=# SET search_path TO edbcopy;
```

```
SET
```

```
edb=# \dt+
```

```

*
          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
```

```

*
          List of relations
 Schema | Name      | Type  | Owner      |
-----+-----+-----+-----+
 edbcopy | salesemp  | view  | enterprisedb |
(1 row)
```

```
edb=# \di
```

```

*
          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
```

```

*
          List of relations
 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;
```

```

 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
```

(次のページに続く)

(前のページからの続き)

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)

8.2.2 localcopyschema_nb

</ div>

localcopyschema_nb ファンクションは、ジョブ外部サーバ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_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ファンクションによって返されます。ファンクションが失敗すると、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。

pgAgent ジョブの完了後、remove_log_file_and_job ファンクションを使用してジョブを削除します。

パラメータ

source_fdw

データベースオブジェクトのクローン元となる `postgres_fdw` 外部データラップによって管理される外部サーバの名前。

source

データベースオブジェクトの複製元のスキーマの名前。

target

データベースオブジェクトがソーススキーマから複製されるスキーマの名前。

log_filename

ファンクション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` 設定パラメータ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。データベースオブジェクトをテーブルスペース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` パラメータが省略された場合、デフォルト値は `false` です。

verbose_on

BOOLEAN 値。ターゲットスキーマにオブジェクトを作成するときに `log_filename` に `DDL` を出力するかどうかを指定します。`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、デフォルト値は `false` です。

copy_acls

BOOLEAN 値。ターゲットスキーマに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` 権限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、アクセス制御リストはターゲットスキーマに含まれます。`copy_acls` パラメータが省略された場合、デフォルト値は `false` です。

worker_count

クローンを並行して実行するバックグラウンドワーカーの数。省略した場合、デフォルト値は 1 です。

例

`localcopyschema` の例と同じクローンオペレーションが実行されますが、非ブロッキングファンクション `localcopyschema_nb` が使用されます。

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 が実行されていない場合は、次のように起動できます。pgagent プログラムファイルは、Advanced Server インストールディレクトリ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
```

!!! Note * pgagent -l 2 オプションは DEBUG モードで pgAgent を起動し、-s オプションで指定されたログファイルに継続的なデバッグ情報を記録します。-l オプションに低い値を使用するか、完全に省略してレコードする情報を減らします。

localcopyschema_nb ファンクションは、例で 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');
 localcopyschema_nb
-----
*                  4
(1 row)
```

以下にジョブのステータスを表示します。

```
edb=# SELECT edb_util.process_status_from_log('clone_edb_edbcopy');
*                  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);
 remove_log_file_and_job
-----
t
(1 row)
```

8.2.3 remotecopyschema

</ div>

`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外部サーバ。

```
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 ]]]  
)
```

関数によって BOOLEAN 値が返されファンクション。ファンクションが成功すると、`true` が返されます。ファンクションが失敗すると、`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。

パラメータ

`source_fdw`

データベースオブジェクトのクローン元となる `postgres_fdw` 外部データラップによって管理される外部サーバの名前。

`target_fdw`

データベースオブジェクトのクローン先となる `postgres_fdw` 外部データラップによって管理される外部サーバの名前。

`source_schema`

データベースオブジェクトの複製元のスキーマの名前。

`target_schema`

データベースオブジェクトがソーススキーマから複製されるスキーマの名前。

`log_filename`

ファンクション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` 設定パラメータ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。データベースオブジェクトをテーブルスペース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 パラメータが省略された場合、デフォルト値は false です。

!!! Note * 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、そのテーブルスペースがターゲットデータベースクラスタに存在しない場合、クローニングファンクションは失敗します。

verbose_on

BOOLEAN 値。ターゲットスキーマにオブジェクトを作成するときに log_filename に DDL を出力するかどうかを指定します。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、デフォルト値は false です。

copy_acls

BOOLEAN 値。ターゲットスキーマに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 権限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、アクセス制御リストはターゲットスキーマに含まれます。copy_acls パラメータが省略された場合、デフォルト値は false です。

!!! Note * 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に存在しない場合、クローン作成ファンクションは失敗します。

worker_count

クローンを並行して実行するバックグラウンドワーカーの数。省略した場合、デフォルト値は 1 です。

例

次の例は、src_server によって定義されたデータベース srcdb 内のスキーマ srcschema の、tgt_server によって定義されたデータベース tgtddb 内のスキーマ tgtschema へのクローン作成を示しています。

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) および先行する簡条書きの情報を使用したユーザマッピング
- ソーススキーマ：srcschema

ターゲットサーバー環境：

- ターゲットデータベースサーバーが実行されているホスト：localhost
- ターゲット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ターゲット：tgtddb
- 外部サーバー (tgt_server) および先行する簡条書きの情報を使用したユーザマッピング
- ターゲットスキーマ：tgtschema

- **remotecopyschema:** **enterprisedb** を呼び出すデータベーススーパーユーザ

ファンクションを呼び出す前に、データベースユーザ **enterprisedb** がデータベース **tgtdb** に接続します。このファンクションには、4 の **worker_count** が指定されています。

```
tgtdb=# SELECT edb_util.remotecopyschema
('src_server','tgt_server','srcschema','tgtschema','clone_rmt_src_tgt',worker_count =>
4);
remotecopyschema
-----
t
(1 row)
```

以下は、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からのステータスを表示します。

```
tgtdb=# SELECT edb_util.process_status_from_log('clone_rmt_src_tgt');
*
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');
*
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');
*
process_status_from_log
-----
->-----
(RUNNING,"28-JUN-17 13:18:10.550393 -04:00",4039,INFO,"STAGE: POST-DATA","CREATE
->PRIMARY KEY
```

(次のページに続く)

(前のページからの続き)

```

CONSTRAINT pgbench_tellers_pkey successful"
)
(1 row)

tgtddb=# SELECT edb_util.process_status_from_log('clone_rmt_src_tgt');
      *
      process_status_from_log
-----
(1 row)

(FINISH,"28-JUN-17 13:18:12.019627 -04:00",4039,INFO,"STAGE: FINAL","successfully
clone
schema into tgtschema")
(1 row)

```

以下に、クローン表を示します。

```

tgtddb=# \dt+
      *
      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ると、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コピーオペレーション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つの角括弧番号が含まれていることに注意してください (例、[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。4つのバックグラウンドワーカーのワーカーインデックス値のレンジは、0から3です。

2つのクローンスキーマ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番目にはジョブインデックスとして1があります。

8.2.4 remotecopyschema_nb

</div>

remotecopyschema_nb 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内のソーススキーマから、target_fdw 外部サーバー外部サーバーで指定されたローカルターゲットデータベース内の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。

```

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)

(前のページからの続き)

```
* [, <worker_count> INTEGER ]]]
)
```

INTEGER 値のジョブ ID は、pgAgent に送信されたジョブのファンクションによって返されます。ファンクションが失敗すると、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。

pgAgent ジョブの完了後、remove_log_file_and_job ファンクションを使用してジョブを削除します。

パラメータ

source_fdw

データベースオブジェクトのクローン元となる postgres_fdw 外部データラップによって管理される外部サーバの名前。

target_fdw

データベースオブジェクトのクローン先となる postgres_fdw 外部データラップによって管理される外部サーバの名前。

source

データベースオブジェクトの複製元のスキーマの名前。

target

データベースオブジェクトがソーススキーマから複製されるスキーマの名前。

log_filename

ファンクション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 設定パラメータ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。データベースオブジェクトをテーブルスペース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 パラメータが省略された場合、デフォルト値は false です。

!!! Note * 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、そのテーブルスペースがターゲットデータベースクラスタに存在しない場合、クローニングファンクションは失敗します。

verbose_on

BOOLEAN 値。ターゲットスキーマにオブジェクトを作成するときに log_filename に DDL を出力するかどうかを指定します。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、デフォルト値は false です。

copy_acls

BOOLEAN 値。ターゲットスキーマに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 権限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、アクセス制御リストはターゲットスキーマに含まれます。copy_acls パラメータを省略すると、デフォルト値は false になります。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に存在しない場合、クローニングファンクションは失敗します。

worker_count

クローンを並行して実行するバックグラウンドワーカーの数。省略した場合、デフォルト値は 1 です。

例

remotecopyschema の例と同じクローンオペレーションが実行されますが、非ブロッキングファンクション remotecopyschema_nb が使用されます。

次のコマンドは、ターゲットデータベース tgtadb で pgAgent を起動します。pgagent プログラムファイルは、Advanced Server インストールディレクトリサブディレクトリにあります。

```
[root@localhost bin]# ./pgagent -l 1 -s /tmp/pgagent_tgtadb_log hostaddr=127.0.0.1
→port=5444
user=enterprisedb dbname=tgtadb password=password
```

remotecopyschema_nb ファンクションは、例で 2 として示されているジョブ ID を返します。

```
tgtadb=# SELECT edb_util.remotecopyschema_nb
('src_server','tgt_server','srcschema','tgtschema','clone_rmt_src_tgt',worker_count =>
4);
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。

```
tgtadb=# SELECT edb_util.process_status_from_log('clone_rmt_src_tgt');
*                process_status_from_log
-----
→-----
(FINISH,"29-JUN-17 current:16:00.100284 -04:00",3849,INFO,"STAGE: FINAL",
→"successfully clone schema into tgtschema")
(1 row)
```

以下は、ログファイルと pgAgent ジョブを削除します。

```
tgtadb=# SELECT edb_util.remove_log_file_and_job ('clone_rmt_src_tgt',2);
remove_log_file_and_job
-----
t
(1 row)
```

8.2.5 process_status_from_log

</div>

`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
)
```

このファンクションは、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 `FAILED` 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とき。タイムスタンプフォーマットで表示されます。
<i>pid</i>	クロー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 `SUCCESSFUL` 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 `FINAL` のいずれかを表示します。
<i>message</i>	各コマンドまたは失敗に対応する情報。
	NA

パラメータ

`log_file`

クローン作成ファンクションが呼び出されたときに指定されたスキーマのクローン作成を記録するログファイルの名前。

例

以下は、`process_status_from_log` ファンクションの使用法を示しています。

```
edb=# SELECT edb_util.process_status_from_log('clone_edb_edbcopy');
      *
      process_status_from_log
-----
->-----
(FINISH,"26-JUN-17 11:57:03.214458 -04:00",3691,INFO,"STAGE: FINAL","successfully_
->cloned schema")
(1 row)
```

8.2.6 remove_log_file_and_job

</ div>

`remove_log_file_and_job` ファンクションは、スキーマのクローン作成関数によって作成されたログファイルと、非ブロッキング関数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、ファンクション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、ファンクションはジョブのみを削除します。
- 両方が指定されている場合、ファンクションはログファイルとジョブを削除します。

パラメータ

`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。

例

次の例では、ログファイル名を指定して、ログ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');
remove_log_file_and_job
-----
t
(1 row)
```

次の例では、ジョブ ID を指定してジョブのみを削除します。

```
edb=# SELECT edb_util.remove_log_file_and_job (3);
remove_log_file_and_job
-----
t
(1 row)
```

次の例では、両方の値が指定された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);
remove_log_file_and_job
```

```
-----
```

```
t
```

```
(1 row)
```


第9章 Enhanced SQL and Other Miscellaneous Features

</div>

Advanced Server には、拡張された SQL 機能と、柔軟性と利便性を高めるさまざまなその他の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

Advanced Server は、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、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> |
```

(次のページに続く)

(前のページからの続き)

```

* PROCEDURE <proc_name> [([[<argmode>] [<argname>] <argtype> [, ...]])]
* 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> [ , ... ]

```

パラメータ

object_name

コメントするオブジェクトの名前。

AGGREGATE aggregate_name (aggregate_signature)

AGGREGATE 句を含めて、集約に関するコメントを作成します。aggregate_name は集約の名前を指定し、aggregate_signature は次のいずれかの形式で関連するシグネチャを指定します。

```

* |
[ <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 になります。省略した場合、デフォルトは IN です。

argname は集約引数の名前です。

argtype は、集約引数のデータタイプです。

CAST (source_type AS target_type)

CAST 句を含めて、キャストに関するコメントを作成します。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タイプを指定し、target_type はキャストのターゲットデータタイプを指定します。

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、コメントが適用され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

制約に関するコメントを追加するには、CONSTRAINT 句を含めます。制約に関するコメントを作成するとき、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 [, ...]])

ファンクション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、ファンクションの名前を指定します。argmode は、ファンクション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タイプを指定します。

large_object_oid

large_object_oid は、ユーザーが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 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された）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された）データタイプです。

OPERATOR CLASS object_name USING index_method

OPERATOR CLASS 句を含めて、演算子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された）名前を指定します。index_method は、演算子クラスに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、演算子族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族の（オプションでスキーマ修飾された）名前を指定します。index_method は、演算子族に関連するインデックスメソッドを指定します。

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ているテーブルを指定します。

PROCEDURE proc_name ([[argmode] [argname] argtype [, ...]])

PROCEDURE 句を含めて、プロシージャに関するコメントを追加します。`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タイプを指定します。

`RULE rule_name ON table_name`

ルールに `COMMENT` を指定するには、`RULE` 句を含めます。`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` は変換のデータタイプの名前を指定し、`lang_name` は変換の言語の名前を指定します。

`TRIGGER trigger_name ON table_name`

トリガーに `COMMENT` を指定するには、`TRIGGER` 句を含めます。`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`

コメントを削除する文字列リテラルまたは `NULL` として記述されたコメント。

注：

テーブル、集計、照合、変換、ドメイン、外部テーブル、関数、インデックス、演算子、演算子クラス、演算子ファミリー、パッケージソフト、プロシージャ、シーケンス、テキストサーチオブジェクト、タイプ、およびビューの名前は、スキーマで修飾できます。

例：

次の例では、`new_emp` 名前付け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 コア文書を参照してください。

< <https://www.postgresql.org/docs/current/ 静的/sql-comment.html> >

9.2 ファンクションバージョンの出力 ()

`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、バージョン、および製品が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。

Advanced Server の場合、`version()` 出力は、最初のテキストワードが Advanced Server バージョン 10 以前のように * 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フォーマット。

バージョン () 出力の一般的なフォーマットは次のとおりです。

```
PostgreSQL $PG_VERSION_EXT (EnterpriseDB Advanced Server $PG_VERSION) on $host
```

したがって、現在の Advanced Server の場合、バージョン文字列は次のように表示されます。

```
edb@45032=#select version();
version
-----
->-----
-----
PostgreSQL 14.0 (EnterpriseDB 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、Advanced Server 10 の場合、バージョン文字列は次のとおりでした。

```
edb=# select version();
*                               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 スタンバイでの論理デコード

スタンバイサーバでの論理デコードにより、`get`、`peek`、`advance` などの API 操作に応答できるスタンバイサーバにロジカルレプリケーションスロットを作成できます。

LOGICAL DECODING の詳細については、次の場所にある PostgreSQL コア文書を参照してください。

<https://www.postgresql.org/docs/current/logicaldecoding-explanation.html>

スタンバイサーバ上のロジカルスロットを機能させるには、スタンバイで `hot_standby_feedback` パラメータを `ON` に設定する必要があります。`hot_standby_feedback` パラメータは、`VACUUM` がスタンバイサーバ上の既存のロジカルレプリケーションスロットに必要な最近停止した行を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すると、スロットはドロップされます。

スタンバイでのロジカルデコーディングを機能させるには、プライマリサーバとスタンバイサーバの両方で `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ロジカル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ーサーバに書き込まれると、アクティビティによってスタンバイサーバにロジカルスロットが作成されます。プライマリーサーバがアイドル状態の場合、スタンバイサーバにロジカルスロットを作成するのに時間がかかることがあります。

レプリケーションとロジカルデコーディングの例をサポートする関数の詳細については、次の URL で入手可能な PostgreSQL 文書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

</ div>

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サブジェクト性があります。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、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から分離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別名ごとに 1 行が含まれます。ディレクトリは、ホストファイルシステムへのユーザ限定アクセスを許可するパス名の別名です。

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をフェンスできます。例、UTL_FILE パッケージは、ユーザ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の読み取りと書き込みを許可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経路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>	別名の名前。
<i>diowner</i>	<i>oid</i>	<i>not null</i>	別名を所有するユーザの OID。
<i>dirpath</i>	<i>text</i>		アクセスが許可されるディレクトリ名前。
<i>diracl</i>	<i>aclitem[]</i>		別名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。
			NA

</ div>

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 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が表示されます。

Column	Type	Modifiers	Description
<i>group_name</i>	<i>*name*</i>		リソースグループの名前。
<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`は制限がないことを意味します。
<i>per_process_cpu_rate_limit</i>	<i>float8</i>		リソースグループで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`は制限がないことを意味します。
<i>per_process_dirty_rate_limit</i>	<i>float8</i>		リソースグループで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。
			NA
			NA
<i>## edb_policy</i>			NA
			NA

edb_policy テーブルには、ポリシーごとに 1 つの行が含まれます。

Column	Type	Modifiers	Description
<i>polycyname</i>	<i>name</i>	<i>not null</i>	ポリシー名前。
<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>	このポリシーで保護されるオブジェクトの種類: テーブルの場合は「r」 ビューの場合は「v」 =同義語の場合 現在、常に「r」。
<i>policyproc</i>	<i>oid</i>	<i>not null</i>	ポリシー関ションの 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強制される場合は True。
<i>policyselect</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強制される場合は 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強制される場合は True。
<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強制される場合は True。
<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 True。
<i>policyupdate-check</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 True。
<i>polycystatic</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。

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>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。
			NA

10.3 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格納されます。`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>prffailedlogi-nattempts</i>	<i>integer</i>		プロファイルで許可されたログイン試行の失敗回数。 -1 は、デフォルトプロファイルの値を使用する必要があることを示します。 -2 は、失敗したログイン試行に制限がないことを示します。
<i>prfpassword-locktime</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（秒単位）。 -1 は、デフォルトプロファイルの値を使用する必要があることを示します。 -2 は、アカウントを永続的にロックする必要があることを示します。
<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（秒単位）。 -1 は、デフォルトプロファイルの値を使用する必要があることを示します。 -2 は、パスワードの有効期限が切れないことを示します。
<i>prfpassword-gracetime</i>	<i>integer</i>		プロファイルに関連付けられたパスワード猶予時間（秒単位）。 -1 は、デフォルトプロファイルの値を使用する必要があることを示します。 -2 は、パスワードの有効期限が切れないことを示します。
<i>prfpassword-reusestime</i>	<i>integer</i>		ユーザ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、デフォルトプロファイルの値を使用する必要があることを示します。 -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>prfpassword-reusemax</i>	<i>integer</i>		パスワードを再利用する前に発生する必要があるパスワード変更の数。 -1 は、デフォルトプロファイルの値を使用する必要があることを示します。 -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>prfpassword-verifyfuncdb</i>	<i>oid</i>	<i>pg_database.oid</i>	パスワード検証ファンクションが存在するデータベースの OID。

次のページに続く

表 4 – 前のページからの続き

Column	Type	References	Description
			NA
<i>prfpassword-verifyfunc</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関クションの OID。
			NA
			NA

</ div>

10.4 edb_redaction_column

カタログ `edb_redaction_column` は、テーブルの列に添付されたデータ改訂ポリシーの情報を格納します。

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）。
			NA
<i>rdpolicyid</i>	<i>oid</i>	<i>edb_redaction_policy.oid</i>	データ編集ポリシーは、説明されている列に適用されます。
			NA
<i>rdrelid</i>	<i>oid</i>	<i>pg_class.oid</i>	記述された列が属するテーブル。
			NA
<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>		墨消しスコープ：1 = クエリー、2 = top_tlist、4 = top_tlist_or_error
<i>rdexception</i>	<i>int2</i>		墨消しの例外：8 = なし、16 = 等しい、32 = 漏れ防止
<i>rdfuncexpr</i>	<i>pg_node_tree</i>		データ編集ファンクション式
			NA

!!! Note * 表のリダクションポリシー `edb_redaction_column.rdpolicyid` が有効で、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、説明されている列はリダクションされます。

10.5 edb_redaction_policy

`edb_redaction_policy` カタログには、テーブルの墨消し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>		データ編集ポリシーの名前
<i>rdrelid</i>	<i>oid</i>	<i>pg_class.oid</i>	データ編集ポリシーが適用されるテーブル。
<i>rdenable</i>	<i>boolean</i>		データ編集ポリシーは有効になっていますか？
<i>rdexpr</i>	<i>pg_node_tree</i>		データ編集ポリシー式。
			NA
!!! Note			NA
The data redaction policy applies for the table if it is enabled and the expression ever evaluated true.			NA
			NA

10.6 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 行が含まれます。

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>rgrpcpurate-limit</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` は制限がないことを意味します。
<i>rgrpdirtyrate-limit</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` は制限がないことを意味します。
			NA

10.7 edb_variable

`edb_variable` テーブルには、パッケージレベル変数（パッケージ内で宣言された各変数）ごとに 1 行が含まれます。

Column	Type	Modifiers	Description
<i>varname</i>	<i>*name*</i>	<i>not null</i>	変数の名前。
			NA
<i>varpackage</i>	<i>oid</i>	<i>not null</i>	パッケージを格納する `pg_namespace` 行の OID。
			NA

次のページに続く

表 8 – 前のページからの続き

Column	Type	Modifiers	Description
<i>vartype</i>	<i>oid</i>	<i>not null</i>	変数のタイプを定義する `pg_type` 行の OID。
			NA
<i>varaccess</i>	<i>*char*</i>	<i>not null</i>	変数がパッケージの外部に可視される場合は `+`。 変数がパッケージ内にのみ可視される場合は `.`。 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、プライベート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る変数。
<i>varseq</i>	<i>smallint</i>	<i>not null</i>	パッケージで変数が宣言されたオーダー。
			NA
			NA

10.8 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シノニムごとに 1 行が含まれます。

Column	Type	Modifiers	Description
<i>synname</i>	<i>*name*</i>	<i>not null</i>	同義語の名前。
<i>synnamespace</i>	<i>oid</i>	<i>not null</i>	`synowner` を置き換えます。同義語が格納されている `pg_namespace` 行の OID が含まれます
<i>synowner</i>	<i>oid</i>	<i>not null</i>	同義語を所有するユーザの OID。
<i>synobjschema</i>	<i>*name*</i>	<i>not null</i>	被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>	被参照オブジェクトの名前。
<i>synlink</i>	<i>text</i>		被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（オプションの）名前。
			NA

10.9 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クエリーして、アプリケーションで使用される機能がこの展開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。

次のページに続く

表 10 – 前のページからの続き

Column	Type	Description
<i>version</i>	<i>character varying</i> (74)	製品のバージョン番号。
<i>status</i>	<i>character varying</i> (74)	リリースのステータス。

第11章 Advanced Server Exceptions

</ div>

次の表に、事前定義の例外、SQLstate 値、関連するレッドウッドエラーコード、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	例外は、文字の並びから数字への変換が失敗した場合に発生します。
<i>in-valid_number</i>	22000	-6502	文字の並び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	例外は、有効レンジ外の日付フォーマットでフィールドを書き込んでいるときに発生します。
<i>divide_by_zero</i>	22012	-1476	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	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	例外は、プログラムが一意インデックスによって制約されている列に重複する値を格納しようとしたときに発生します。
<i>invalid_cursor</i>	34000	-1001	例外は、プログラムが、開いていないカーソルを閉じるなど、無効なカーソルに対してカーソルオペレーションを試みた場合に発生しカーソル。
<i>cursor_already_open</i>	42P03	-6511	例外は、プログラムがすでに開いているカーソルを開こうとしたときに発生します。
<i>collection_is_null</i>	P1403	-6531	例外は、プログラムが初期化されていないネストされたテーブルまたは VARRAY の要素に値を割り当てようとしたときに発生します。
<i>sub-script_beyond_count</i>	P1404	-6533	例外は、プログラムがコレクション内の要素数よりも大きいインデックス番号を使用してネストされたテーブルまたは VARRAY をリファレンスしようとした場合に発生します。
<i>sub-script_outside_limit</i>	P1405	-6532	例外は、プログラムがレンジ外のインデックス番号を使用してネストされたテーブルまたは VARRAY 要素をリファレンスしようとした場合に発生します。
			NA

** Redwood 組み込みパッケージの例外**

DBMS_CRYPTO Package	
<i>ciphersuiteinvalid</i>	00009
<i>ciphersuitenull</i>	00009
<i>keybadsize</i>	00009
<i>keynull</i>	00009
UTL_FILE Package	
<i>invalid_filehandle</i>	P0001 00009
<i>invalid_maxlinesize</i>	P0001 00009
<i>invalid_mode</i>	P0001 00009
<i>invalid_operation</i>	P0001 00009
<i>invalid_path</i>	P0001 00009
<i>read_error</i>	P0001 00009
<i>write_error</i>	P0001 00009
UTL_HTTP Package	
<i>end_of_body</i>	P0001 00009
UTL_URL Package	
<i>bad_fixed_width_charset</i>	00009
<i>bad_url</i>	P0001 00009

</ div>

11.1 Advanced Server キーワード

キーワードは、Advanced Server パーサによって特別な意味または関連があると認識されるワードです。pg_get_keywords() ファンクションを使用して、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();			
*	word	catcode	catdesc
-----+-----+-----			
abort		U	unreserved
absolute		U	unreserved

(次のページに続く)

(前のページからの続き)

access	U	unreserved
...		

`pg_get_keywords` は、Advanced Server が認識す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` 列には、カテゴリコードが表示されます。
- `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。

名前がダブルクォートで囲まれている場合、識別子に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クエリーして、特定のカテゴリに属する 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-ワードは予約。予約済みのキーワードを識別子として使用することはできません。サーバーで使用するために予約されています。

U-ワードは予約されていません。予約されていない単語は、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-ワードは内部的に使用されますが、ファンクションまたは型の名前として使用できます。

C-ワードは内部的に使用され、ファンクションまたは型の名前として使用することはできません。

Advanced Server の識別子とキーワードの詳細については、次の PostgreSQL コア文書を参照してください。

< <https://www.postgresql.org/docs/current/sql-syntax-lexical.html> >