



EDB Postgres™ Advanced Server

リリース **13**

Advanced Server Guide

2020 年 11 月 29 日

目次

第 1 章	Introduction	1
1.1	新機能	3
1.2	制限事項	6
1.3	このガイドで使用される規則	7
1.4	このガイドで使用されている例について	8
1.4.1	サンプルデータベースの説明	8
第 2 章	Enhanced Compatibility Features	21
2.1	互換性機能を有効にする	21
2.2	ストアドプロシージャ言語	21
2.3	オプティマイザーのヒント	22
2.4	データディクショナリビュー	22
2.5	dblink_oracle	23
2.6	プロファイル管理	23
2.7	組み込みパッケージ	24
2.8	クライアントライブラリを開く	26
2.9	公益事業	27
2.10	ECPGPlus	28
2.11	テーブルのパーティション分割	29
第 3 章	Database Administration	31
3.1	Configuration Parameters	31
3.1.1	構成パラメーターの設定	31
3.1.1.1	postgresql.conf ファイルの変更	33
3.1.1.2	pg_hba.conf ファイルの変更	33
3.1.2	構成パラメーターの要約	34
3.1.3	機能別の構成パラメーター	40
3.1.3.1	トップパフォーマンス関連のパラメーター	42
3.1.3.1.1	shared_buffers	42
3.1.3.1.2	work_mem	42
3.1.3.1.3	maintenance_work_mem	43
3.1.3.1.4	wal_buffers	44
3.1.3.1.5	checkpoint_segments	44

3.1.3.1.6	checkpoint_completion_target	44
3.1.3.1.7	checkpoint_timeout	45
3.1.3.1.8	max_wal_size	45
3.1.3.1.9	min_wal_size	46
3.1.3.1.10	bgwriter_delay	46
3.1.3.1.11	seq_page_cost	47
3.1.3.1.12	random_page_cost	47
3.1.3.1.13	effective_cache_size	48
3.1.3.1.14	同期コミット	48
3.1.3.1.15	edb_max_spins_per_delay	49
3.1.3.1.16	pg_prewarm.autoprewarm	50
3.1.3.1.17	pg_prewarm.autoprewarm_interval	51
3.1.3.2	リソース使用量/メモリ	51
3.1.3.2.1	edb_dynatune	51
3.1.3.2.2	edb_dynatune_profile	52
3.1.3.3	リソース使用量/EDB リソースマネージャー	52
3.1.3.3.1	edb_max_resource_groups	53
3.1.3.3.2	edb_resource_group	53
3.1.3.4	クエリチューニング	53
3.1.3.4.1	enable_hints	53
3.1.3.5	クエリチューニング/プランナーメソッドの構成	54
3.1.3.5.1	edb_enable_pruning	54
3.1.3.6	レポートとログ/ログの対象	55
3.1.3.6.1	trace_hints	55
3.1.3.6.2	edb_log_every_bulk_value	56
3.1.3.7	監査設定	56
3.1.3.7.1	edb_audit	56
3.1.3.7.2	edb_audit_directory	56
3.1.3.7.3	edb_audit_filename	57
3.1.3.7.4	edb_audit_rotation_day	57
3.1.3.7.5	edb_audit_rotation_size	58
3.1.3.7.6	edb_audit_rotation_seconds	58
3.1.3.7.7	edb_audit_connect	58
3.1.3.7.8	edb_audit_disconnect	59
3.1.3.7.9	edb_audit_statement	59
3.1.3.7.10	edb_audit_tag	60
3.1.3.7.11	edb_audit_destination	60
3.1.3.7.12	edb_log_every_bulk_value	60
3.1.3.8	クライアント接続のデフォルト/ロケールとフォーマット	61
3.1.3.8.1	icu_short_form	61
3.1.3.9	クライアント接続のデフォルト/ステートメントの動作	61

3.1.3.9.1	default_heap_fillfactor	61
3.1.3.9.2	edb_data_redaction	62
3.1.3.10	クライアント接続のデフォルト/その他のデフォルト	62
3.1.3.10.1	oracle_home	63
3.1.3.10.2	odbc_lib_path	63
3.1.3.11	互換性オプション	64
3.1.3.11.1	edb_redwood_date	64
3.1.3.11.2	edb_redwood_greatest_least	64
3.1.3.11.3	edb_redwood_raw_names	65
3.1.3.11.4	edb_redwood_strings	67
3.1.3.11.5	edb_stmt_level_tx	69
3.1.3.11.6	db_dialect	70
3.1.3.11.7	default_with_rowids	71
3.1.3.11.8	optimizer_mode	72
3.1.3.12	カスタマイズされたオプション	72
3.1.3.12.1	custom_variable_classes	72
3.1.3.12.2	dbms_alert.max_alerts	73
3.1.3.12.3	dbms_pipe.total_message_buffer	73
3.1.3.12.4	index_advisor.enabled	73
3.1.3.12.5	edb_sql_protect.enabled	74
3.1.3.12.6	edb_sql_protect.level	75
3.1.3.12.7	edb_sql_protect.max_protected_relations	75
3.1.3.12.8	edb_sql_protect.max_protected_roles	76
3.1.3.12.9	edb_sql_protect.max_queries_to_save	77
3.1.3.12.10	edb_wait_states.directory	78
3.1.3.12.11	edb_wait_states.retention_period	79
3.1.3.12.12	edb_wait_states.sampling_interval	79
3.1.3.12.13	edblldr.empty_csv_field	79
3.1.3.12.14	utl_encode.uudecode_redwood	80
3.1.3.12.15	utl_file.umask	80
3.1.3.13	グループ化されていない	81
3.1.3.13.1	nls_length_semantics	81
3.1.3.13.2	query_rewrite_enabled	82
3.1.3.13.3	query_rewrite_integrity	82
3.1.3.13.4	timed_statistics	83
3.2	インデックスアドバイザー	84
3.2.1	インデックスアドバイザーのコンポーネント	86
3.2.2	インデックスアドバイザーの構成	88
3.2.3	インデックスアドバイザーの使用	90
3.2.3.1	pg_advise_index ユーティリティの使用	90
3.2.3.2	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	91

3.2.4	インデックスアドバイザーの推奨事項の確認	94
3.2.4.1	show_index_recommendations () 関数を使用する	94
3.2.4.2	index_advisor_log テーブルのクエリ	95
3.2.4.3	index_recommendations ビューのクエリ	97
3.2.5	インデックスアドバイザーの制限	99
3.3	SQL プロファイラー	100
3.4	pgsnmpd	103
3.4.1	pgsnmpd の構成	103
3.4.2	リスナーアドレスの設定	104
3.4.3	pgsnmpd の呼び出し	104
3.4.4	pgsnmpd ヘルプの表示	105
3.4.5	pgsnmpd からの情報の要求	105
3.5	EDB 監査ログ	106
3.5.1	監査ログ構成パラメーター	106
3.5.2	監査する SQL ステートメントの選択	108
3.5.2.1	データ定義言語とデータ制御言語のステートメント	110
3.5.2.2	データ操作言語ステートメント	118
3.5.3	監査ログを有効にする	120
3.5.4	監査ログファイル	127
3.5.5	エラーコードを使用して監査ログをフィルタリングする	135
3.5.6	コマンドタグを使用して監査ログをフィルタリングする	136
3.5.7	監査ログからのパスワードの編集	137
3.6	Unicode 照合アルゴリズム	139
3.6.1	基本的な Unicode 照合アルゴリズムの概念	139
3.6.2	Unicode の国際コンポーネント	140
3.6.2.1	ローケール照合	140
3.6.2.2	照合属性	141
3.6.3	照合の使用	143
第 4 章	EDB リソースマネージャー	149
4.1	リソースグループの作成と管理	151
4.1.1	リソースグループの作成	151
4.1.2	リソースグループの変更	152
4.1.3	ドロップグループ	153
4.1.4	リソースグループへのプロセスの割り当て	154
4.1.5	リソースグループからのプロセスの削除	155
4.1.6	リソースグループのプロセスの監視	155
4.2	CPU 使用率の調整	157
4.2.1	リソースグループの CPU レート制限の設定	157
4.2.2	例-単一グループ内の単一プロセス	158
4.2.3	例-単一グループ内の複数のプロセス	159

4.2.4	例-複数のグループの複数のプロセス	161
4.3	ダーティバッファスロットリング	164
4.3.1	リソースグループのダーティレート制限の設定	164
4.3.2	例-単一グループ内の単一プロセス	165
4.3.3	例-単一グループ内の複数のプロセス	168
4.3.4	例-複数のグループの複数のプロセス	170
4.4	システムカタログ	174
4.4.1	edb_all_resource_groups	174
4.4.2	edb_resource_group	174
第 5 章	libpqC ライブラリ	175
5.1	EDBSPL での libpq の使用	175
5.2	REFCURSOR のサポート	176
5.3	配列バインディング	183
5.3.1	PQBulkStart	183
5.3.2	PQexecBulk	183
5.3.3	PQBulkFinish	184
5.3.4	PQexecBulkPrepared	184
5.3.5	サンプルコード (PQBulkStart、PQexecBulk、PQBulkFinish を使用)	184
5.3.6	サンプルコード (PQexecBulkPrepared を使用)	186
第 6 章	デバッガー	187
6.1	デバッガーの構成	188
6.2	デバッガーの起動	189
6.3	デバッガーウィンドウ	190
6.4	メインデバッガーウィンドウ	192
6.4.1	プログラム本体パネル	193
6.4.2	タブパネル	194
6.4.3	スタックタブ	196
6.5	プログラムのデバッグ	199
6.5.1	コードのステップ実行	199
6.5.2	ブレークポイントを使用する	199
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	201
6.5.4	デバッガーの終了	205
第 7 章	パフォーマンス分析とチューニング	207
7.1	ダイナチューン	207
7.1.1	edb_dynatune	207
7.1.2	edb_dynatune_profile	208
7.2	EDB 待ち状態	209
7.2.1	edb_wait_states_data	212
7.2.2	edb_wait_states_queries	214

7.2.3	edb_wait_states_sessions	215
7.2.4	edb_wait_states_samples	216
7.2.5	edb_wait_states_purge	218
第 8 章	EDB クローンスキーマ	221
8.1	設定プロセス	223
8.1.1	拡張機能と PL/Perl のインストール	223
8.1.2	構成パラメーターの設定	226
8.1.2.1	パフォーマンス構成パラメーター	226
8.1.2.2	ステータスロギング	226
8.1.3	EDB クローンスキーマのインストール	227
8.1.4	外部サーバーとユーザーマッピングの作成	228
8.1.4.1	ローカルクローニング機能の外部サーバーおよびユーザーマッピング	228
8.1.4.2	リモートクローニング機能の外部サーバーおよびユーザーマッピング	230
8.2	EDB クローンスキーマ関数	233
8.2.1	localcopyschema	233
8.2.2	localcopyschema_nb	237
8.2.3	remotecopyschema	240
8.2.4	remotecopyschema_nb	244
8.2.5	process_status_from_log	247
8.2.6	remove_log_file_and_job	249
第 9 章	拡張された SQL およびその他のその他の機能	251
9.1	コメント	251
9.2	Functionversion () の出力	255
9.3	スタンバイでの論理デコード	255
第 10 章	システムカタログ表	257
10.1	edb_dir	257
10.2	edb_all_resource_groups	258
10.3	edb_policy	258
10.4	edb_profile	259
10.5	edb_redaction_column	261
10.6	edb_redaction_policy	261
10.7	edb_resource_group	262
10.8	edb_variable	262
10.9	pg_synonym	263
10.10	product_component_version	263
第 11 章	高度なサーバーの例外	265
第 12 章	高度なサーバーキーワード	269

第 13 章 Conclusion	271
索引	273

第 1 章

Introduction

AdvancedServer 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、パフォーマンスの監視と分析、およびアプリケーション開発ユーティリティをサポートします。このガイドでは、AdvancedServer 専用の機能について説明しています。

- **Enhanced Compatibility Features.** このセクションでは、AdvancedServer がサポートする互換性機能の概要を説明します。
- **Database Administration.** 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、AdvancedServer の現在の状態に関する階層監視情報を返す SNMP エージェントです。
- **Security.** このセクションには、AdvancedServer がサポートするセキュリティ機能に関する情報が含まれています。
 - SQL/Protect は、SQL インジェクション攻撃に対する保護を提供します。
 - Virtual Private Database は、きめ細かい行レベルのアクセスを提供します。
 - sslutils は SSL 証明書生成機能を提供します。
 - Data redaction は、機密データの漏洩に対する保護を提供します。
- **EDB Resource Manager.** このセクションには、AdvancedServer 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 EDBResourceManager 機能に関する情報が含まれています。
 - Resource Groups は、リソース制限を定義できるグループを作成および維持する方法を示します。

- CPU Usage Throttling は、AdvancedServer プロセスによる CPU 使用率を制御する方法を提供します。
- Dirty Buffer Throttling は、AdvancedServer 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方法を提供します。
- **The libpq C Library.** libpq C library は、AdvancedServer の C アプリケーションプログラミングインターフェイス (API) 言語です。
- **The PL Debugger.** PL Debugger は、PL/pgSQL 用のグラフィカル指向のデバッグツールです。
- **Performance Analysis and Tuning.** 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ー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 AdvancedServer を構成するための迅速かつ簡単な手段を提供します。
 - 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 Clone Schema.** このセクションには、単一のデータベース内またはあるデータベースから別のデータベースにスキーマとそのデータベースオブジェクトをコピーする機能を提供する EDB クローンスキーマ機能に関する情報が含まれています。
- **Enhanced SQL and Other Miscellaneous Features.** このセクションには、拡張された SQL 機能および追加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。
- ****System Catalog Tables.**** このセクションには、AdvancedServer 固有のデータベースオブジェクト用に追加された追加の*システムカタログテーブル* が含まれています。
- **Advanced Server Keywords.** このセクションには、AdvancedServer 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。

AdvancedServer と PostgreSQL で共有される機能の詳細については、次の場所にある PostgreSQL コアドキュメントを参照してください。

<https://www.postgresql.org/docs/current/index.html>

1.1 新機能

AdvancedServer13 を作成するために、EDBPostgresAdvancedServer12 では次の機能が変更されました。

- AdvancedServer は PARALLEL をサポートするようになりました。インデックスまたはテーブルの並列処理を有効または無効にする CREATE TABLE, ALTER TABLE, CREATE INDEX および ALTER INDEX コマンドの NOPARALLEL 節。詳細については、*OracleDeveloper'sSQLGuide* のデータベース互換性を参照してください。
- AdvancedServer は、テーブルのインデックスを明示的に作成するために、CREATE TABLE および ALTER TABLE コマンドで USING INDEX ... create_index_statement 句をサポートするようになりました。詳細については、*OracleDeveloper'sSQLGuide* のデータベース互換性を参照してください。
- AdvancedServer には AUTOMATIC LIST PARTITIONING が導入されており、これにより、パーティションキーの値が既存のパーティションに対応していない場合に、データベースが新しいパーティションを自動的に作成できます。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。
- AdvancedServer は、最も頻繁な入力値を返す値のセットを引数とする STATS_MODE 関数を追加しました。詳細については、『Oracle 開発者リファレンスガイドのデータベース互換性』を参照してください。
- AdvancedServer は、タイムゾーンなしのタイムスタンプを返す SYSDATE 機能を追加しました。詳細については、『Oracle 開発者リファレンスガイドのデータベース互換性』を参照してください。
- AdvancedServer は、MEDIAN 機能に smallint, init, bigint および numeric バリエーションを追加しました。詳細については、『Oracle 開発者リファレンスガイドのデータベース互換性』を参照してください。
- AdvancedServer は、TO_NUMBER 機能に FM 形式のサポートを追加しました。詳細については、『Oracle 開発者リファレンスガイドのデータベース互換性』を参照してください。
- AdvancedServer は、ステートメントのログ記録をサポートし、一括実行中に処理された行数を記録するようになりました。詳細については、レポートとログ/何をログに記録するか<reporting_and_logging_what_to_log>の ‘edb_log_every_bulk_value’ を参照してください。
- AdvancedServer は、DBMS_OUTPUT パッケージで dbms_output.serveroutput を FALSE に設定することにより、Oracle スタイルの表示出力をサポートする新しい GUC を追加しました。詳細については、*OracleDeveloper'sBuilt-inPackageGuide* のデータベース互換性を参照してください。
- EDB*Loader は、重複レコードを BAD ファイルに移動するための HANDLE_CONFLICTS パラメーターをサポートするようになりました。詳細については、*OracleDeveloper'sToolsandUtilitiesGuide* のデータベース互換性を参照してください。
- AdvancedServer は、ディレクトリの所有者を変更するための ALTER DIRECTORY ...OWNER TO コマンドを追加しました。詳細については、*OracleDeveloper'sSQLGuide* のデータベース互換性を参照してください。

- AdvancedServer は、トリガーの暗黙オブジェクトの所有者を変更する `ALTER TRIGGER ...ON AUTHORIZATION` コマンドを追加しました。詳細については、*OracleDeveloper'sSQLGuide* のデータベース互換性を参照してください。
- EDB*Loader は `-c CONNECTION STRING` または `connstr=CONNECTION_STRING` パラメーターをサポートするようになりました。これにより、`libpq` でサポートされる SSL パラメーターと接続パラメーターを指定できます。詳細については、*OracleDeveloper'sToolsandUtilitiesGuide* のデータベース互換性を参照してください。
- AdvancedServer には、タイムスタンプ形式の文字列を `TIMESTAMP_TZ` データ型に変換する `TO_TIMESTAMP_TZ` 関数が追加されています。詳細については、『Oracle 開発者リファレンスガイドのデータベース互換性』を参照してください。
- AdvancedServer は、CSV と XML の両方の監査ログでログの一貫性を保つための追加フィールドを追加しました。詳細については、*EDBAuditLogging* を参照してください。
- AdvancedServer は、リモートステートメントのスキャンタイプを制御する新しい GUC `edb_dblink_oci.rescan` を追加しました。詳細については、*OracleDeveloper'sSQLGuide* のデータベース互換性を参照してください。
- AdvancedServer は、応答本文にデータが存在しない場合に `read_line`, `read_text` および `read_raw` パッケージプロシージャからのエラーを処理するために、`UTL_HTTP` パッケージに `utl_http.end_of_body` 例外のサポートを追加しました。詳細については、*OracleDeveloper'sBuilt-inPackageGuide* のデータベース互換性を参照してください。
- AdvancedServer は、パッケージボディ内の関数またはプロシージャの `forward declaration` をサポートするようになりました。詳細については、*OracleDeveloper'sSQLGuide* のデータベース互換性を参照してください。
- AdvancedServer は、サブパーティションレベルで自動ハッシュパーティションを作成するための `PARTITIONS`, `SUBPARTITIONS` 番号と、自動生成されたパーティションまたはサブパーティションを格納するテーブルスペースを指定するための `STORE IN` 句を導入します。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。
- AdvancedServer は、データ型の暗号化と復号化のために `DBMS_CRYPT` パッケージに `AES192` と `AES256` のサポートを追加しました。詳細については、*OracleDeveloper'sBuilt-inPackageGuide* のデータベース互換性を参照してください。
- AdvancedServer は、サーバーとの認証に成功すると `pg_hba.conf` エントリをサポートするようになりました。詳細については、*pg_hba.conf ファイルの変更* を参照してください。
- AdvancedServer は、指定された条件が発生したときにトリガーを実行するために、`COMPOUND TRIGGER` 内に `WHEN` 条件を実装しています。詳細については、『OracleDeveloperSQL ガイドのデータベース互換性*』または『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。
- AdvancedServer は、`LONG` 列の一部を定義して返すために、`DBMS_SQL` パッケージに

column_value_long、define_column_long、および last_error_position のサポートを追加しました。詳細については、*OracleDeveloper'sBuilt-inPackageGuide* のデータベース互換性を参照してください。

1.2 制限事項

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容量などの実質的な制限が適用される場合があります。

アイテム	上限	コメント
データベースサイズ	無制限	
データベースの数	4,294,950,911	
データベースごとの関係	1,431,650,303	
関係サイズ	32TB	デフォルトの BLCKSZ は 8192 バイトです
テーブルごとの行	4,294,967,295 ページに収まる タブルの数によって制限されま す	
テーブルごとの列	1600	単一ページ上のタブルサイズのフィッティング によりさらに制限されます。以下の注を参照
フィールドサイズ	1GB	
識別子の長さ	63 バイト	PostgreSQL を再コンパイルすることで増やすこ とができます
テーブルごとのインデッ クス	無制限	データベースごとの最大リレーションによる制 約
インデックスごとの列	32	PostgreSQL を再コンパイルすることで増やすこ とができます
パーティションキー	32	PostgreSQL を再コンパイルすることで増やすこ とができます

注釈: 格納されるタブルは単一の 8192 バイトのヒープページに収まる必要があるため、テーブルの列の最大数はさらに削減されます。たとえば、タブルヘッダーを除外すると、1600 の int 列で構成されるタブルは 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 列のタブルは 12800 バイトを消費するため、内部に収まりませんヒープページ。text、varchar、char などのタイプの可変長フィールドは、値が必要な大きさの場合、テーブルの TOAST テーブルに値を行外に格納できます。テーブルのヒープ内のタブル内に残るのは 18 バイトのポインターのみです。短い可変長フィールドの場合、4 バイトまたは 1 バイトのフィールドヘッダーが使用され、値はヒープタブル内に格納されます。

注釈: テーブルから削除された列も、最大列制限に寄与します。さらに、新しく作成されたタブルのドロップされた列の値は、タブルの null ビットマップで null として内部的にマークされますが、null ビットマップはスペースも占有します。

1.3 このガイドで使用される規則

以下は、このドキュメント全体で使用される規則のリストです。

- 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。ディレクトリパスは、スラッシュ付きの Linux 形式で表示されます。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開始し、スラッシュをバックスラッシュに置き換えます。
- このドキュメントの情報の一部は、PostgreSQL および EDBPostgresAdvancedServer 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。*AdvancedServer**という用語は、*EDBPostgresAdvancedServer* を指すために使用されます。**Postgres* という用語は、PostgreSQL と AdvancedServer 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する必要がある場合は、特定の名前、PostgreSQL または AdvancedServer が使用されます。
- PostgreSQL または AdvancedServer 製品のインストールディレクトリパスは、*POSTGRES_INSTALL_HOME* と呼ばれます。
 - PostgreSQLLinux インストールの場合、これはバージョン 10 以前のデフォルトは `/opt/PostgreSQL/x.x` です。それ以降のバージョンでは、PostgreSQL コミュニティパッケージを使用してください。
 - バージョン 10 以前の対話型インストーラーを使用して行われる AdvancedServerLinux インストールの場合、デフォルトは `/opt/edb/asx.x` です。
 - RPM パッケージを使用して AdvancedServerLinux をインストールする場合、デフォルトは `/usr/edb/asxx` です。
 - AdvancedServerWindows インストールの場合、デフォルトは `C:/Program Files/edb/as/xx` 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。

1.4 このガイドで使用されている例について

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、白い背景に固定幅の黒いフォントで表示されます。

例では、AdvancedServer 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 AdvancedServer のバージョン番号です。

さらに、Oracle データベースと互換性のある構文を使用して作成されたデータベースオブジェクト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。
- テーブルのすべての権限を PUBLIC グループに付与します。

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

1.4.1 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、および管理者がいます。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在籍しており、さまざまな役職に就いたり、昇給を受け取ったり、部門を切り替えたりしました。開始日、新しい役

職、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 テーブルに保持されます。

以下は pg-sample.sql スクリプトです。

```
SET datestyle TO 'iso, dmy';

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

(次のページに続く)

(前のページからの続き)

```

    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');
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

```

(次のページに続く)

(前のページからの続き)

```

(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');

```

(次のページに続く)

(前のページからの続き)

```

INSERT INTO jobhist VALUES
(7902, '03-DEC-81', NULL, 'ANALYST', 3000, NULL, 20, 'New Hire');
INSERT INTO jobhist VALUES
(7934, '23-JAN-82', NULL, 'CLERK', 1300, NULL, 10, 'New Hire');
--
-- Populate statistics table and view (pg_statistic/pg_stats)
--
ANALYZE dept;
ANALYZE emp;
ANALYZE jobhist;
--
-- Function that lists all employees' numbers and names
-- from the 'emp' table using a cursor.
--
CREATE OR REPLACE FUNCTION list_emp() RETURNS VOID
AS $$
DECLARE
    v_empno NUMERIC(4);
    v_ename VARCHAR(10);
    emp_cur CURSOR FOR
        SELECT empno, ename FROM emp ORDER BY empno;
BEGIN
    OPEN emp_cur;
    RAISE INFO 'EMPNO ENAME';
    RAISE INFO '-----';
    LOOP
        FETCH emp_cur INTO v_empno, v_ename;
        EXIT WHEN NOT FOUND;
        RAISE INFO '% %', v_empno, v_ename;
    END LOOP;
    CLOSE emp_cur;
    RETURN;
END;
$$ 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;

```

(次のページに続く)

(前のページからの続き)

```

    v_disp_date VARCHAR(10);
BEGIN
    SELECT INTO
        v_ename, v_hiredate, v_sal, v_comm, v_dname
        ename, hiredate, sal, COALESCE(comm, 0), dname
    FROM emp e, dept d
    WHERE empno = p_empno
        AND e.deptno = d.deptno;
    IF NOT FOUND THEN
        RAISE INFO 'Employee % not found', p_empno;
        RETURN;
    END IF;
    v_disp_date := TO_CHAR(v_hiredate, 'MM/DD/YYYY');
    RAISE INFO 'Number : %', p_empno;
    RAISE INFO 'Name : %', v_ename;
    RAISE INFO 'Hire Date : %', v_disp_date;
    RAISE INFO 'Salary : %', v_sal;
    RAISE INFO 'Commission: %', v_comm;
    RAISE INFO 'Department: %', v_dname;
    RETURN;
EXCEPTION
    WHEN OTHERS THEN
        RAISE INFO 'The following is SQLERRM : %', SQLERRM;
        RAISE INFO 'The following is SQLSTATE: %', SQLSTATE;
        RETURN;
END;
$$ LANGUAGE 'plpgsql';
--
-- A RECORD type used to format the return value of
-- function, 'emp_query'.
--
CREATE TYPE emp_query_type AS (
    empno NUMERIC,
    ename VARCHAR(10),
    job VARCHAR(9),
    hiredate DATE,
    sal NUMERIC
);
--
-- Function that queries the 'emp' table based on
-- department number and employee number or name. Returns
-- employee number and name as INOUT parameters and job,
-- hire date, and salary as OUT parameters. These are
-- returned in the form of a record defined by
-- RECORD type, 'emp_query_type'.
--
CREATE OR REPLACE FUNCTION emp_query (
    IN p_deptno NUMERIC,

```

(次のページに続く)

(前のページからの続き)

```

    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
    WHEN OTHERS THEN
        RAISE INFO 'The following is SQLERRM : %', SQLERRM;
        RAISE INFO 'The following is SQLSTATE: %', SQLSTATE;
        RETURN;
END;

```

(次のページに続く)

(前のページからの続き)

```

$$ LANGUAGE 'plpgsql';
--
-- Function to compute yearly compensation based on semimonthly
-- salary.
--
CREATE OR REPLACE FUNCTION emp_comp (
    p_sal NUMERIC,
    p_comm NUMERIC
) RETURNS NUMERIC
AS $$
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
    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);

```

(次のページに続く)

(前のページからの続き)

```

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);
    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

```

(次のページに続く)

(前のページからの続き)

```

        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,
                  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

```

(次のページに続く)

(前のページからの続き)

```
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
        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;
```

(次のページに続く)

(前のページからの続き)

```
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

AdvancedServer には、Oracle アプリケーションでサポートされる構文の互換性を提供する拡張機能が含まれています。AdvancedServer でサポートされている互換性機能の詳細については、Oracle 開発者向けデータベース互換性ガイドに記載されています。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/edb-docs><<https://www.enterprisedb.com/resources/product-documentation>>‘__

次のセクションでは、AdvancedServer 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

AdvancedServer 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。

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

AdvancedServer 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、EDBWeb サイトから入手できる*EDBPostgresAdvancedServer インストールガイド*を参照してください。

<https://www.enterprisedb.com/edb-docs><<https://www.enterprisedb.com/resources/product-documentation>>‘__

2.2 ストアドプロシージャ言語

AdvancedServer は、カスタムプロシージャ、関数、トリガー、パッケージを作成できる生産性の高い手続き言語をサポートしています。手続き言語:

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

ストアドプロシージャ言語の使用方法については、次の URL にある「ストアドプロシージャ言語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/edb-docs<https://www.enterprisedb.com/resources/product-documentation>’](https://www.enterprisedb.com/edb-docs&https://www.enterprisedb.com/resources/product-documentation)__

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

- データ処理操作の推定実行コスト。
- postgresql.conf ファイルの「クエリ調整」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。

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

オプティマイザーヒントは、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まれたディレクティブ（または複数のディレクティブ）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。オプティマイザーのヒントの使用については、次の場所にある「OracleDevelopersGuide 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/edb-docs<https://www.enterprisedb.com/resources/product-documentation>’](https://www.enterprisedb.com/edb-docs&https://www.enterprisedb.com/resources/product-documentation)__

2.4 データディクショナリビュー

AdvancedServer には、Oracle データディクショナリビューと互換性のある方法でデータベースオブジェクトに関する情報を提供するビューのセットが含まれています。AdvancedServer で使用可能なビューの詳細については、次の場所にある『OracleDevelopersCatalogViews 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/edb-docs><https://www.enterprisedb.com/resources/product-documentation>>‘__

2.5 dblink_oracle

dblink_oracle は、AdvancedServer 内から Oracle システムに保存された SELECT, INSERT, UPDATE または DELETE の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 の使用方法の詳細、およびサポートされている関数と手順については、次の場所にある「OracleDevelopersGuide 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/edb-docs><https://www.enterprisedb.com/resources/product-documentation>>‘__

2.6 プロファイル管理

AdvancedServer 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、データベースのスーパーユーザーが名前付き*profiles*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
- ログイン試行が過度に失敗したためにアカウントをロックする
- 期限切れのパスワードをマークする
- パスワードの有効期限後の猶予期間を定義する
- パスワードの複雑さのルールを定義する
- パスワードの再利用を制限するルールを定義する

プロファイルは、同等の認証要件を共有する役割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きセットです。パスワード要件が変更された場合、プロファイルを変更して、そのプロファイルに関連付けられている各ユーザーに新しい要件を適用できます。

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

プロファイル管理コマンドの使用については、次の場所にある「OracleDevelopersGuide 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/edb-docs><<https://www.enterprisedb.com/resources/product-documentation>>‘__

2.7 組み込みパッケージ

AdvancedServer は、Oracle の手順および機能との互換性を提供する多数の組み込みパッケージをサポートしています。

パッケージ名	説明
DBMS_ALERT	DBMS_ALERT パッケージは、アラートを登録、送信、および受信する機能を提供します。
DBMS_AQ	DBMS_AQ パッケージは、AdvancedServer のメッセージキューと処理を提供します。
DBMS_AQADM	DBMS_AQADM パッケージは、アドバンスドキュー機能のサポート手順を提供します。
DBMS_CRYPT	DBMS_CRYPT パッケージは、RAW、BLOB または CLOB データの暗号化または復号化を可能にする関数およびプロシージャを提供します。DBMS_CRYPT 関数を使用して、暗号的に強力なランダム値を生成することもできます。
DBMS_JOB	DBMS_JOB パッケージは、ジョブの作成、スケジューリング、および管理を提供します。
DBMS_LOB	DBMS_LOB パッケージは、大きなオブジェクトを操作する機能を提供します。
DBMS_LOCK	AdvancedServer 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更新します。
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、ランダムな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 パッケージを使用すると、クエリによって返されるデータの編集またはマスキングが可能になります。
DBMS_RLS	DBMS_RLS パッケージを使用すると、特定の AdvancedServer 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	AdvancedServer 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 の動的 SQL 機能へのアプリケーションインターフェイスを提供します。
DBMS_UTILITY	DBMS_UTILITY パッケージは、さまざまなユーティリティプログラムを提供します。
UTL_ENCODE	UTL_ENCODE パッケージは、データをエンコードおよびデコードする方法を提供します。
UTL_FILE	UTL_FILE パッケージは、オペレーティングシステムのファイルシステム上のファイルを読み書きする機能を提供します。
UTL_HTTP	UTL_HTTP パッケージは、HTTP または HTTPS プロトコルを使用して、URL で見つかった情報を取得する方法を提供します。
UTL_MAIL	UTL_MAIL パッケージは、電子メールを管理する機能を提供します。
UTL_RAW	UTL_RAW パッケージを使用すると、生データ型の長さを操作または取得できます。
2.7. 組み込みパッケージ UTL_SMTP	UTL_SMTP パッケージは、SMTP (SimpleMailTransferProtocol) を介して電子メールを送信する機能を提供します。
UTL_URL	UTL_URL パッケージは、URL 内の不正な予約文字をエスケープする方法を提供します。

各パッケージ内で使用可能な手順と機能の詳細については、次の場所にある『OracleDevelopers 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/edb-docs><https://www.enterprisedb.com/resources/product-documentation>>‘__

2.8 クライアントライブラリを開く

OpenClientLibrary は、OracleCallInterface 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、AdvancedServer または Oracle データベースと連携できます。OpenClientLibrary の EDB 実装は C で記述されています。

次の図は、OpenClientLibrary と OracleCallInterface アプリケーションスタックを比較しています。

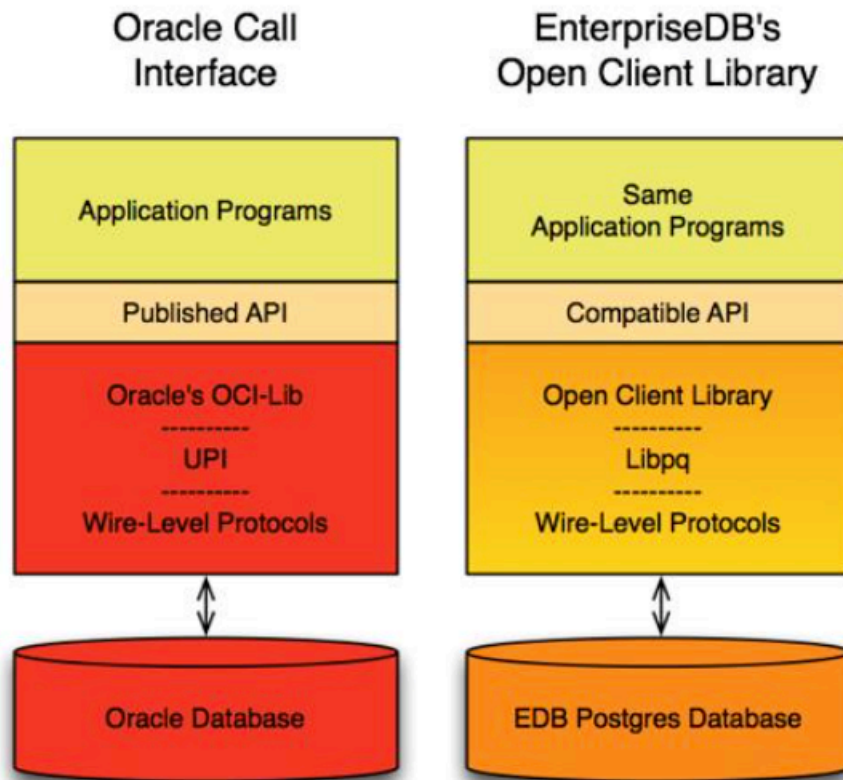


図 1 OpenClientLibrary

OpenClientLibrary でサポートされている機能の詳細については、次の Web サイトで入手可能な*EDBPostgresAdvancedServerOCLConnectorGuide* を参照してください。

<https://www.enterprisedb.com/edb-docs><https://www.enterprisedb.com/resources/product-documentation>>‘__

2.9 公益事業

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『Oracle Developer's 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/edb-docs><<https://www.enterprisedb.com/resources/product-documentation>>‘ __

EDB*Plus

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

EDB*Plus でできること:

- 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を実行する
- SQL コマンドからの出力のフォーマット
- バッチスクリプトを実行する
- OS コマンドを実行する
- レコード出力

EDB*Plus の詳細については、次の Web サイトで入手可能な*EDB*Plus ユーザーガイド*を参照してください。

<https://www.enterprisedb.com/edb-docs/p/edbplus>

EDB*Loader

EDB*Loader は、AdvancedServer 用の Oracle データベースと互換性のあるインターフェイスを提供する高性能バルクデータローダーです。EDB*Loader コマンドライン・ユーティリティは、OracleSQL* 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 つ以上の表にデータをロードします。

EDB*Loader の機能は次のとおりです。

- OracleSQL*Loader データのロード方法のサポート-従来のパス・ロード、ダイレクト・パス・ロードおよびパラレル・ダイレクト・パス・ロード
- 制御ファイルのディレクティブ用の OracleSQL*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
- 複数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します
- 標準入力からのデータ読み込みとリモート読み込み

EDB*Wrap

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）を不正な検査から保護します。EDB*Wrap プログラムは、SPL または PL/pgSQL ソースコード（ブレンテキスト）を含むファイルを、読み取りがほとんど不可能な形式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形式を取得したら、そのコードを AdvancedServer に送信すると、それらのプログラムが難読化された形式で保存されます。EDB* Wrap は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形式で保存されます。パッケージ全体をラップすると、パッケージ本体のソース、パッケージヘッダーに含まれるプロトタイプ、パッケージ本体に含まれる関数とプロシージャが難読化された形式で保存されます。

Dynamic Runtime Instrumentation Tools Architecture (DRITA)

DynamicRuntimeInstrumentationToolsArchitecture (DRITA) を使用すると、DBA はカタログビューを照会して、個々のセッションまたはシステム全体のパフォーマンスに影響する*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。スナップショットは、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。各スナップショットは一意の ID 番号で識別されます。DRITA レポート機能でスナップショット ID 番号を使用して、システムパフォーマンス統計を返すことができます。

2.10 ECPGPlus

EDB は ECPG (PostgreSQL プリコンパイラ) を強化して 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、AdvancedServer 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:

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

ECPGPlus の使用については、次の EDBWeb サイトから入手できる*EDBPostgresAdvancedServerECPG コネクタガイド*を参照してください。

<https://www.enterprisedb.com/edb-docs><<https://www.enterprisedb.com/resources/product-documentation>>‘__

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、論理的に大きな 1 つのテーブルが小さな物理的な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、インデックスの先頭からパーティション列を省略することができ、インデックスサイズが縮小され、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。
- クエリまたは更新が単一パーティションの大部分にアクセスする場合、テーブル全体に散在するインデックスおよび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順次スキャンを実行するため、パフォーマンスが向上する場合があります。
- パーティションの設計にその要件を計画している場合は、パーティションを追加または削除することで、バルクロード（またはアンロード）を実装できます。ALTER TABLE は一括操作よりもはるかに高速です。また、バルク DELETE による VACUUM オーバーヘッドを完全に回避します。
- めったに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ン化は、テーブルが非常に大きい場合にのみ価値があります。テーブルがパーティション化の恩恵を受ける正確なポイントは、アプリケーションによって異なります。目安として、テーブルのサイズはデータベースサーバーの物理メモリを超える必要があります。

AdvancedServer 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、次の場所にある「*TablePartitioningGuide* 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/edb-docs><<https://www.enterprisedb.com/resources/product-documentation>>‘__

第 3 章

Database Administration

この章では、AdvancedServer 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

このセクションでは、AdvancedServer のデータベースサーバー構成パラメーターについて説明します。これらのパラメーターは、データファイルとログファイルの場所、接続、認証、セキュリティ設定、リソースの割り当てと消費、アーカイブとレプリケーションの設定、エラーログと統計の収集、最適化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ーの動作と環境のさまざまな側面を制御します。ロケールとフォーマット設定など。

AdvancedServer にのみ適用される構成パラメーターは、[構成パラメーターの概要](#) に記載されています。

構成パラメーターに関する追加情報は、次の場所にある [PostgreSQLCoreDocumentation](#) にあります。

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

3.1.1 構成パラメーターの設定

このセクションでは、構成パラメーターの指定および設定方法の概要を説明します。

各構成パラメーターは、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と小文字を区別しません。パラメーター名は、通常、オプションの等号 (=) によって値と区切られています。

以下は、`postgresql.conf` ファイルのいくつかの構成パラメーター設定の例です。

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

パラメーター値は、5 つのタイプのいずれかとして指定されます。

- **Boolean.** 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。
- **Integer.** 小数部のない数字。
- **Floating Point.** 小数点で区切られたオプションの小数部を持つ数値。
- **String.** テキスト値。値が単純な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）、単一引用符で囲みます。
- **Enum.** 特定の文字列値のセット。許可される値は、システムビュー `pg_settings.enumvals` で確認できます。列挙値は大文字と小文字を区別しません。

一部の設定では、メモリまたは時間の値を指定します。これらにはそれぞれ、キロバイト、ブロック（通常は 8 キロバイト）、ミリ秒、秒、または分という暗黙の単位があります。デフォルトの単位は、システムビュー `pg_settings.unit` を参照することで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。

構成パラメーターの設定は、さまざまな方法で確立できます。

- **AdvancedServer** データベース製品のビルド時に確立されるパラメーター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、データベースに対して後で変更することはできません。
- データベースクラスター全体のほ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ーの起動または再起動時に有効になります。これらの初期パラメーター設定の一部は、次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限り有効な組み込み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。
- パラメータ設定は、データベースサーバーの実行中に構成ファイルで変更できます。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）、特定のパラメータータイプに対して、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については、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ー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 コアドキュ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、特定のパラメーター設定を変更できます。変更されたパラメーター設定は、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す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、a) これらのコマンドを異なるパラメーター値で再発行するか、b) SET parameter TO DEFAULT のいずれかを使用してこれらのコマンドを発行することによってのみ変更できます節または「10」節。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ーを変更します。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>

- 環境変数 PGOPTIONS 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の 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ー設定を変更できます。この方法で行われたパラメーター設定は、セッションの期間中のみ、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。

3.1.1.1 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 AdvancedServer 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。パラメーターを設定または変更した後、新しいパラメーター値を有効にするには、サーバーを reload または restart 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。サーバーの再起動が必要なパラメーターのリストを表示するには、psql コマンドラインで次のクエリを実行します。

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

3.1.1.2 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、接続クライアントを認証するときにサーバーが使用する認証方法を指定します。サーバーに接続する前に、pg_hba.conf ファイルで指定された認証プロパティの変更が必要になる場合があります。

initdb ユーティリティを起動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、選択したエディター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、サーバーを再起動して変更を適用します。

認証と pg_hba.conf ファイルの変更の詳細については、次の PostgreSQLCore ドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/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 ファイルのエントリを記録してサーバーへの接続を認証するには、postgresql.conf ファイルの log_connections パラメーターを ON に設定します。

レコードは、接続タイプ、データベース名、ユーザー名、クライアント IP アドレス、および pg_hba.conf ファイルのこれらのパラメーターと一致したときに接続を許可する認証方法を指定します。サーバーへの接続が承認されると、一致した行番号と 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=ul 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 構成パラメーターの要約

このセクションには、すべての AdvancedServer 構成パラメーターとパラメーターのいくつかの主要な属性をリストした要約表が含まれています。

これらの属性は、要約表の以下の列で説明されています。

- **Parameter.** 設定パラメータ名。
- **Scope of Effect.** 構成パラメーター設定の効果の範囲。
 - Cluster –設定は、データベースクラスター全体（つまり、データベースサーバーインスタンスによって管理されるすべてのデータベース）に影響します。
 - Database –設定はデータベースによって異なり、データベースの作成時に設定されます。ロケール設定に関連する少数のパラメーターに適用されます。
 - Session –設定は、個々のセッションの粒度まで変更できます。

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。このリストの後者の設定は、以前の設定を上書きします。a) データベースクラスター全体、b) データベースクラスター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。

- **When Takes Effect.** 変更されたパラメーター設定が有効になるとき。
 - **Preset** –AdvancedServer 製品が構築されたとき、または特定のデータベースが作成されたときに確立されます。これは読み取り専用パラメーターであり、変更できません。
 - **Restart** –データベースサーバ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（またはデータベースサーバーを再起動できます）。
 - **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** –EDBPostgresAdvancedServer サービスアカウント（Oracle データベースと互換性のあるインストールの場合は enterprisedb、PostgreSQL 互換モードのインストールの場合は postgres）。
 - **Superuser** –スーパーユーザー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 コマンドで変更され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** –ユーザーはパラメーター設定を変更できません。
- **Description.** 構成パラメーターの簡単な説明。
- **EPAS Only.** 「X」 –構成パラメーターは EDBPostgresAdvancedServer のみに適用されます。この列のエントリーは、構成パラメーターが PostgreSQL にも適用されることを示しています。

注釈: 変更してはなら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「**Note:For internal use only**」と指定されています。

次の表に、構成パラメーターの要約を示します。

パラメータ	効果の 範囲	有効になる 時期	承認済み ユーザー	説明	EPAS の み
-------	-----------	-------------	--------------	----	-------------

次のページに続く

表 1 – 前のページからの続き

custom_variable_classes	クラス ター	リロード	EPAS サービス アカウント	AdvancedServer9.2 で廃 止されました。	X
db_dialect	セッ ション	すぐに	ユーザー	組み込みの名前空間の優 先順位を設定します。	X
dbms_alert.max_alerts	クラス ター	再起動	EPAS サービス アカウント	アラートの最大数を設定 します。	X
dbms_pipe.total_message_buffer	クラス ター	再起動	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。	X
default_heap_fillfactor	セッ ション	すぐに	ユーザー	デフォルトでは、この ヒープ fillfactor で新しい テーブルを作成します。	X
default_with_rowids	セッ ション	すぐに	ユーザー	デフォルトでは、ROWID サポート（インデックス 付き ROWID）を使用して 新しいテーブルを作成し ます。	X
edb_audit	クラス ター	リロード	EPAS サービス アカウント	EDB 監査を有効にして、 XML または CSV 形式で 監査レポートを作成しま す。	X
edb_audit_connect	クラス ター	リロード	EPAS サービス アカウント	成功した各接続を監査し ます。	X
edb_audit_destination	クラス ター	リロード	EPAS サービス アカウント	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
edb_audit_directory	クラス ター	リロード	EPAS サービス アカウント	監査ファイルの宛先ディ レクトリを設定します。	X

次のページに続く

表 1 – 前のページからの続き

edb_audit_disconnect	クラス ター	リロード	EPAS サービス アカウント	セッションの終了を監査 します。	X
edb_audit_filename	クラス ター	リロード	EPAS サービス アカウント	監査ファイルのファイル 名パターンを設定します。	X
edb_audit_rotation_day	クラス ター	リロード	EPAS サービス アカウント	曜日に基づいたログファ イルの自動ローテーショ ン。	X
edb_audit_rotation_seconds	クラス ター	リロード	EPAS サービス アカウント	N 秒後にログファイルの 自動ローテーションが行 われます。	X
edb_audit_rotation_size	クラス ター	リロード	EPAS サービス アカウント	N メガバイト後にログフ ァイルの自動ローテーシ ョンが発生します。	X
edb_audit_statement	クラス ター	リロード	EPAS サービス アカウント	監査するステートメント のタイプを設定します。	X
edb_audit_tag	セッ ション	すぐに	ユーザー	監査ログに含めるタグを 指定します。	X
edb_connectby_order	セッ ション	すぐに	ユーザー	ORDERBY を使用しない CONNECTBY クエリの 結果を深さ優先順に並べ 替えます。注:内部使用の み。	X
edb_data_redaction	セッ ション	すぐに	ユーザー	データ編集を有効にしま す。	X
edb_dynatune	クラス ター	再起動	EPAS サービス アカウント	edb 使用率を設定します。	X
edb_dynatune_profile	クラス ター	再起動	EPAS サービス アカウント	dynatune のワークロード プロファイルを設定しま す。	X
edb_enable_pruning	セッ ション	すぐに	ユーザー	プランナーがパーティシ ョンテーブルを早期にプ ルーニングできるように します。	X

次のページに続く

表 1 – 前のページからの続き

edb_log_every_bulk_value	セッション	すぐに	スーパーユーザー	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
edb_max_resource_groups	クラスター	再起動	EPAS サービスアカウント	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
edb_max_spins_per_delay	クラスター	再起動	EPAS サービスアカウント	ロックを待機している間にセッションがスピンする回数を指定します。	X
edb_redwood_date	セッション	すぐに	ユーザー	DATE が TIMESTAMP のように動作するかどうかを決定します。	X
edb_redwood_greatest_least	セッション	すぐに	ユーザー	GREATEST および LEAST 関数が NULL パラメーターを処理する方法を決定します。	X
edb_redwood_raw_names	セッション	すぐに	ユーザー	Redwood インターフェースから PostgreSQL システムカタログに保存されている未変更の名前を返します。	X
edb_redwood_strings	セッション	すぐに	ユーザー	NULL をテキスト値と連結する場合、空の文字列として扱います。	X
edb_resource_group	セッション	すぐに	ユーザー	現在のプロセスで使用されるリソースグループを指定します。	X
edb_sql_protect.enabled	クラスター	リロード	EPAS サービスアカウント	SQL/Protect がクエリを追跡するかどうかを定義します。	X
edb_sql_protect.level	クラスター	リロード	EPAS サービスアカウント	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
edb_sql_protect.max_protected_relations	クラスター	再起動	EPAS サービスアカウント	ロールごとに SQL/Protect によって保護される関係の最大数を設定します。	X

次のページに続く

表 1 – 前のページからの続き

edb_sql_protect.max_protected_roles	クラスター	再起動	EPAS サービスアカウント	SQL/Protect によって保護されるロールの最大数を設定します。	X
edb_sql_protect.max_queries_to_save	クラスター	再起動	EPAS サービスアカウント	SQL/Protect で保存する問題のあるクエリの最大数を設定します。	X
edb_stmt_level_tx	セッション	すぐに	ユーザー	トランザクションの中止を要求する代わりに、エラーで続行できます。	X
edb_wait_states.directory	クラスター	再起動	EPAS サービスアカウント	EDB 待機状態ログはこのディレクトリに保存されます。	X
edb_wait_states.retention_period	クラスター	リロード	EPAS サービスアカウント	EDB 待機状態ログファイルは、保持期間後に自動的に削除されます。	X
edb_wait_states.sampling_interval	クラスター	リロード	EPAS サービスアカウント	2 つの EDB 待機状態サンプリングサイクルの間隔。	X
edbldr.empty_csv_field	セッション	すぐに	スーパーユーザー	EDB*Loader が空の文字列を処理する方法を指定します。	X
enable_hints	セッション	すぐに	ユーザー	SQL ステートメントでオプティマイザーヒントを有効にします。	X
icu_short_form	データベース	プリセット	該当なし	ICU 照合順序の構成を示します。	X
index_advisor.enabled	セッション	すぐに	ユーザー	IndexAdvisor プラグインを有効にします。	X
nls_length_semantics	セッション	すぐに	スーパーユーザー	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
odbc_lib_path	クラスター	再起動	EPAS サービスアカウント	ODBC ライブラリのパスを設定します。	X
optimizer_mode	セッション	すぐに	ユーザー	デフォルトのオプティマイザーモード。	X

次のページに続く

表 1 – 前のページからの続き

oracle_home	クラスター	再起動	EPAS サービスアカウント	Oracle ホームディレクトリのパスを設定します。	X
pg_prewarm.autoprewarm	クラスター	再起動	EPAS サービスアカウント	自動事前ウォームバックグラウンドワーカーを有効にします。	X
pg_prewarm.autoprewarm_interval	クラスター	リロード	EPAS サービスアカウント	autoprewarm がダンプするまでの最小秒数を設定します 共有バッファ。	X
qreplace_function	セッション	すぐに	スーパーユーザー	クエリ置換機能で使用される関数。注:内部使用のみ。	X
query_rewrite_enabled	セッション	すぐに	ユーザー	子テーブルスキャンは、制約がクエリに一致する行がないことを保証している場合はスキップされます。	X
query_rewrite_integrity	セッション	すぐに	スーパーユーザー	クエリの書き換えを強制する度合いを設定します。	X
timed_statistics	セッション	すぐに	ユーザー	時限統計の記録を有効にします。	X
trace_hints	セッション	すぐに	ユーザー	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
utl_encode.uudecode_redwood	セッション	すぐに	ユーザー	Oracle が作成した uencoded データのデコードを許可します。	X
utl_file.umask	セッション	すぐに	ユーザー	UTL_FILE パッケージで作成されたファイルに使用される Umask。	X

3.1.3 機能別の構成パラメーター

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。

各パラメーター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。パラメーターの種類の値について

は、[設定パラメーター](#) セクションを参照してください。

- `Default Value`. 値が構成ファイルで明示的に設定されていない場合のデフォルト設定。
- `Range`. 値の許容範囲。
- `Minimum Scope of Effect`. 個別の設定が可能な最小のスコープ。一般に、個別の設定の最小スコープは、`cluster` 全体（設定はクラスター内のすべてのデータベースとそのセッションと同じです）または `per session`（設定はロール、データベース、または個人によって異なる場合があります）セッション）。（この属性は、構成パラメーターの概要<summary_of_configuration_parameters>の表の “Scope of Effect” 列と同じ意味を持ちます）。
- `When Value Changes Take Effect`. パラメータの値の変更を有効にするために必要な最小限の侵襲的アクション。構成ファイルで行われたすべてのパラメーター設定の変更は、データベースサーバーを再起動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ー `restart` が必要です。一部のパラメーター設定の変更は、データベースサーバーを停止せずに、構成ファイルの `reload` で有効にすることができます。最後に、他のパラメーター設定の変更は、結果が `immediate` であるクライアント側のアクションで有効にすることができます。（この属性は、`SummaryofConfigurationParameters`<summary_of_configuration_parameters>の表の “When Takes Effect” 列と同じ意味を持ちます）。
- `Required Authorization to Activate`. パラメータの設定の変更を有効にするユーザー認証のタイプ。データベースサーバーの再起動または構成ファイルの再読み込みが必要な場合、ユーザーは EPAS サービスアカウントである必要があります（AdvancedServer 互換インストールモードに応じて `enterprisedb` または `postgres`）。（この属性は、構成パラメーターの概要<summary_of_configuration_parameters>の表の “Authorized User” 列と同じ意味を持ちます）。

3.1.3.1 トップパフォーマンス関連のパラメーター

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

3.1.3.1.1 shared_buffers

Parameter Type: 整数

Default Value: 32MB

Range: 128kB からシステムに依存

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: 再起動

Required Authorization to Activate: 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（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 の大きな設定でも効果的なワークロードがありますが、AdvancedServer はオペレーティングシステムのキャッシュにも依存しているため、shared_buffers への RAM の 40 % 以上の割り当てが小さいものよりもうまく機能する可能性は低いです量。

RAM が 1GB 未満のシステムでは、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 では、shared_buffers の値を大きくしても効果的ではありません。設定を比較的低く保ち、代わりにオペレーティングシステムのキャッシュをより多く使用すると、より良い結果が得られる場合があります。Windows システムでの shared_buffers の有効範囲は、一般に 64MB から 512MB です。

このパラメーターを大きくすると、AdvancedServer 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 SystemV 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、*PostgreSQLCoreDocumentation* のセクション Shared Memory and Semaphores を参照してください。

3.1.3.1.2 work_mem

Parameter Type: 整数

Default Value: 1MB

Range: 64kB から 2097151kB

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

一時ディスクファイルに書き込む前に、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。複雑なクエリの場合、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。各操作は、一時ファイルへのデータの書き込みを開始する前に、この値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`、および結合の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。

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

3.1.3.1.3 maintenance_work_mem

Parameter Type: 整数

Default Value: 16MB

Range: 1024kB から 2097151kB

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

`VACUUM`、`CREATE INDEX`、`ALTER TABLE ADD FOREIGN KEY` など、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、インストールでは通常、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。

`autovacuum` を実行すると、このメモリが最大 `autovacuum_max_workers` 回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。

目安として、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。

3.1.3.1.4 wal_buffers

Parameter Type: 整数

Default Value: 64kB

Range: 32kB からシステムに依存

Minimum Scope of Effect: クラスタ

When Value Changes Take Effect: 再起動

Required Authorization to Activate: 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のコミットごと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大きさである必要があります。

このパラメーターを大きくすると、AdvancedServer 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 SystemV 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、*PostgreSQLCoreDocumentation* 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わけではありませんが、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、このパラメーターは AdvancedServer の初期共有メモリ割り当てを増やします。そのため、このパラメーターを増やすと AdvancedServer の起動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。

3.1.3.1.5 checkpoint_segments

廃止されました。このパラメーターは、サーバーバージョン 9.5 以降ではサポートされていません。

3.1.3.1.6 checkpoint_completion_target

Parameter Type: 浮動小数点

Default Value: 0.5

Range: 0 から 1

Minimum Scope of Effect: クラスタ

When Value Changes Take Effect: リロード

Required Authorization to Activate: 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 %が準備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、チェックポイントの書き込みを長時間にわたって調整し、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。

3.1.3.1.7 checkpoint_timeout

Parameter Type: 整数

Default Value: 5 分

Range: 30 から 3600

Minimum Scope of Effect: クラスタ

When Value Changes Take Effect: リロード

Required Authorization to Activate: 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。このパラメーターを増やすと、クラッシュ回復に必要な時間が長くなる可能性があります。

checkpoint_timeout を 15 分などの大きな値に増やすと、特に shared_buffers に大きな値を使用する場合に、システムの I/O 負荷を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、システムが適度な量の追加ディスクスペースを使用し、クラッシュが発生した場合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。

3.1.3.1.8 max_wal_size

Parameter Type: 整数

Default Value: 1GB

Range: 2 から 2147483647

Minimum Scope of Effect: クラスタ

When Value Changes Take Effect: リロード

Required Authorization to Activate: 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（高負荷、archive_command の失敗、または高い wal_keep_segments

設定の場合) で `max_wal_size` を超えることがあります。

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

3.1.3.1.9 min_wal_size

Parameter Type: 整数

Default Value: 80MB

Range: 2 から 2147483647

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: リロード

Required Authorization to Activate: EPAS サービスアカウント

WAL ディスクの使用量が `min_wal_size` で指定された値を下回った場合、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の将来の使用のためにリサイクルされます。これにより、WAL 使用量の急増に対処するために十分な WAL スペースが確保されます (大規模なバッチジョブの実行時など)。このパラメーターは、`postgresql.conf` ファイルまたはサーバーのコマンドラインでのみ設定できます。

3.1.3.1.10 bgwriter_delay

Parameter Type: 整数

Default Value: 200ms

Range: 10ms から 10000ms

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: リロード

Required Authorization to Activate: EPAS サービスアカウント

バックグラウンドライター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、ライター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 の倍数ではない値に設定すると、次に高い 10 の倍数に設定した場合と同じ結果になる場合があります。

通常、`bgwriter_delay` を調整する場合は、デフォルト値から減らす必要があります。このパラメーターは、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。

3.1.3.1.11 seq_page_cost

Parameter Type: 浮動小数点

Default Value: 1

Range: 0 から 1.79769e+308

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

一連の順次フェッチの一部であるディスクページフェッチのコスト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。この値は、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ます。(PostgreSQLCoreDocumentation 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非常に少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 未満（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。

3.1.3.1.12 random_page_cost

Parameter Type: 浮動小数点

Default Value: 4

Range: 0 から 1.79769e+308

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

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。この値は、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ます。(PostgreSQLCoreDocumentation の ALTER TABLESPACE コマンドを参照してください。)

seq_page_cost と比較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。値を大きくすると、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 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。これは、cpu_tuple_cost および cpu_index_tuple_cost パラメータで説明されています。

デフォルト値はキャッシングが非常に少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メモリよりも大幅に大きい場合でも、このパラメーターを（デフォルトの 4 ではなく）2 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。

システムで許可されますが、`random_page_cost` を `seq_page_cost` よりも小さく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場合、それらを等しく（または非常に近い）に設定することは理にかなっています。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 パラメーターに比べて両方の値を低くする必要があります。

3.1.3.1.13 `effective_cache_size`

Parameter Type: 整数

Default Value: 128MB

Range: 8kB から 17179869176kB

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

単一のクエリ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、インデックスの使用コストの見積もりに含まれます。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、値が小さいほど順次スキャンが使用される可能性が高くなります。このパラメーターを設定するときは、AdvancedServer の共有バッファと、AdvancedServer データファイルに使用されるカーネルのディスクキャッシュの両方を考慮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、異なるテーブルでの同時クエリの予想数を考慮してください。このパラメータは、AdvancedServer によって割り当てられた共有メモリのサイズには影響しません。また、カーネルディスクキャッシュを予約しません。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。

このパラメーターの設定が低すぎると、プランナーは、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、物理メモリの約 75 % です。

3.1.3.1.14 同期コミット

Parameter Type: ブール

Default Value: true

Range: {true,false}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

コマンドがクライアントに `success` の指示を返す前に、トランザクションのコミットが WAL レコードがディスクに書き込まれるのを待つ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ークラッシュに対して安全であることが実際に保証されるまでに遅延が発生する可能性があります。(最大遅延は `wal_writer_delay` の 3 倍です。)

`fsync` とは異なり、このパラメーター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のクラッシュにより、コミットされたとされる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とまったく同じになりますきれいに中止されました。

そのため、トランザクションの耐久性に関する正確な確実性よりもパフォーマンスが重要な場合、`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。

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

3.1.3.1.15 `edb_max_spins_per_delay`

Parameter Type: 整数

Default Value: 1000

Range: 10 から 1000

Minimum Scope of Effect: クラスターごと

When Value Changes Take Effect: 再起動

Required Authorization to Activate: 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` に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。

3.1.3.1.16 pg_prewarm.autoprewarm

Parameter Type: ブール

Default Value: true

Range: {true,false}

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: 再起動

Required Authorization to Activate: EPAS サービスアカウント

このパラメーターは、データベースサーバーが 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ープロセスです。次に、サーバーの次回起動時に共有バッファを prewarms、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ドすることにより、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。このパラメーター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 構成パラメーター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ーを再起動します。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ーが起動します。

autoprewarm プロセスは、バッファプールに空きバッファスペースがなくなるまで、\$PGDATA/autoprewarm ブロックに以前に記録されたブロックのロードを開始します。この方法では、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ドを完了すると、pg_prewarm.autoprewarm_interval パラメータで指定された間隔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ーの詳細については、pg_prewarm.autoprewarm_interval<pg_prewarm.autoprewarm_interval>を参照してください。次のサーバーの再起動時に、“autoprewarm” 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。

3.1.3.1.17 pg_prewarm.autoprewarm_interval

Parameter Type: 整数

Default Value: 300 秒

Range: 0 から 2147483s

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: リロード

Required Authorization to Activate: 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ーが共有バッファをディスクにダンプするまでの最小秒数です。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ーについては、[pg_prewarm.autoprewarm](#) を参照してください。

3.1.3.2 リソース使用量/メモリ

このセクションの構成パラメーターは、メモリーに関するリソース使用量を制御します。

3.1.3.2.1 edb_dynatune

Parameter Type: 整数

Default Value: 0

Range: 0 から 100

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: 再起動

Required Authorization to Activate: 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ー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。

AdvancedServer が最初にインストールされる時、edb_dynatune パラメーターは、インストールされたホストマシン（開発マシン、混合マシン、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は、パフォーマンスを向上させるために postgresql.conf ファイルのさまざまな構成パラメーターを調整する必要はありません。

edb_dynatune パラメーターは、0 から 100 までの任意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ーのリソース使用量が postgresql.conf ファイルの他の構成

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ーに割り当てます。この設定は、他の多くのアプリケーションが使用されている開発マシンで使用されます。

34 から 66 の範囲の値は、適度な量のリソースをデータベースサーバー専用にします。この設定は、AdvancedServer と同じマシン上で実行され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ー専用にします。この設定は、AdvancedServer の実行専用のホストマシンに使用されます。

edb_dynatune の値を選択すると、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ー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 で選択された対応する値を上書きします。パラメーターの値を変更するには、構成パラメーターのコメントを外し、目的の値を指定して、データベースサーバーを再起動します。

3.1.3.2.2 edb_dynatune_profile

Parameter Type: 列挙

Default Value: oltp

Range: {oltp|レポート|混合}

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: 再起動

Required Authorization to Activate: EPAS サービスアカウント

このパラメーターは、データベースサーバーで予想されるワークロードプロファイルに基づいてチューニング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ー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.** 大量のデータの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。

3.1.3.3 リソース使用量/EDB リソースマネージャー

このセクションの構成パラメーターは、EDBResourceManager を介してリソースの使用を制御します。

3.1.3.3.1 edb_max_resource_groups

Parameter Type: 整数

Default Value: 16

Range: 0 から 65536

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: 再起動

Required Authorization to Activate: EPAS サービスアカウント

このパラメーターは、EDBResourceManager 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。

3.1.3.3.2 edb_resource_group

Parameter Type: スtring

Default Value: なし

Range: 該当なし

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

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、EDBResourceManager 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。

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

3.1.3.4 クエリチューニング

このセクションでは、オブティマイザー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。

3.1.3.4.1 enable_hints

Parameter Type: ブール

Default Value: true

Range: {true,false}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

SQL コマンドに埋め込まれたオプティマイザーヒントは、`enable_hints` がオンのときに使用されます。このパラメーターがオフの場合、オプティマイザーのヒントは無視されます。

3.1.3.5 クエリチューニング/プランナーメソッドの構成

このセクションでは、プランナー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。

3.1.3.5.1 `edb_enable_pruning`

Parameter Type: ブール

Default Value: true

Range: {true,false}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

TRUE に設定すると、`edb_enable_pruning` により、クエリプランナーはパーティションテーブルを早期にブルーニングできます。Early-pruning は、クエリプランナが、クエリ before でクエリプランを生成するクエリ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、クエリプランナがパーティション after をブルーニングし、各パーティションのクエリプランを生成することを意味します。（`constraint_exclusion` 構成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 句のクエリの性質に依存します。アーリーブルーニングは、WHERE column = literal タイプの制約を持つ単純なクエリでのみ使用できます（例:WHERE deptno = 10）。

アーリーブルーニングは、WHERE column = expression などのより複雑なクエリには使用されません（WHERE deptno = 10 + 5）など）。

3.1.3.6 レポートとログ/ログの対象

このセクションの構成パラメーターは、レポートとロギングを制御します。

3.1.3.6.1 trace_hints

Parameter Type: ブール

Default Value: false

Range: {true,false}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

ヒントがプランナー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、オプティマイザー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)
```

3.1.3.6.2 edb_log_every_bulk_value

Parameter Type: ブール

Default Value: false

Range: {true,false}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: スーパーユーザー

一括処理は、結果のステートメントを AdvancedServer ログファイルと EDBAudit ログファイルの両方に記録します。ただし、一括処理ですべてのステートメントをログに記録するにはコストがかかります。これは、edb_log_every_bulk_value 構成パラメーター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。

3.1.3.7 監査設定

このセクションでは、AdvancedServer 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。

3.1.3.7.1 edb_audit

Parameter Type: 列挙

Default Value: なし

Range: {なし|csv|xml}

Minimum Scope of Effect: クラスタ

When Value Changes Take Effect: リロード

Required Authorization to Activate: 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、これもデフォルトです。

3.1.3.7.2 edb_audit_directory

Parameter Type: スtring

Default Value: edb_audit

Range: 該当なし

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: リロード

Required Authorization to Activate: 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、AdvancedServer の data ディレクトリからの絶対パスまたは相対パスです。

3.1.3.7.3 edb_audit_filename

Parameter Type: スtring

Default Value: 監査-% Y % m % **d**_% H % M % S

Range: 該当なし

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: リロード

Required Authorization to Activate: 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を指定します。デフォルトのファイル名は audit-%Y%m%d_%H%M%S です。エスケープシーケンス %Y, %m などは、システムの日付と時刻に従って適切な現在の値に置き換えられます。

3.1.3.7.4 edb_audit_rotation_day

Parameter Type: スtring

Default Value: ごと

Range: {なし|毎|太陽|月|火|水|木|金 sat}...

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: リロード

Required Authorization to Activate: EPAS サービスアカウント

監査ファイルをローテーションする曜日を指定します。有効な値は sun、mon、tue、wed、thu、fri、sat、「11」、および none です。回転を無効にするには、値を none に設定します。ファイルを毎日ローテーションするには、「10」の値を「11」に設定します。特定の曜日にファイルをローテーションするには、値を目的の曜日に設定します。

3.1.3.7.5 edb_audit_rotation_size

Parameter Type: 整数

Default Value: 0MB

Range: 0MB から 5000MB

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: リロード

Required Authorization to Activate: 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。パラメーター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。

3.1.3.7.6 edb_audit_rotation_seconds

Parameter Type: 整数

Default Value: 0s

Range: 0 から 2147483647s

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: リロード

Required Authorization to Activate: 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、このパラメーターを 0 に設定します。

3.1.3.7.7 edb_audit_connect

Parameter Type: 列挙

Default Value: が失敗しました

Range: { なし|失敗しました|すべて }

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: リロード

Required Authorization to Activate: EPAS サービスアカウント

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

3.1.3.7.8 `edb_audit_disconnect`

Parameter Type: 列挙

Default Value: なし

Range: {なし|すべて}

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: リロード

Required Authorization to Activate: 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。

3.1.3.7.9 `edb_audit_statement`

Parameter Type: ストリング

Default Value: `ddl`、エラー

Range: {なし|`ddl`|`dml`|挿入|更新|削除|切り捨てる|選択|エラー|作成|ドロップ|変更する|助成|取り消す|ロールバック|すべて}...

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: リロード

Required Authorization to Activate: EPAS サービスアカウント

この構成パラメーターは、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の監査を指定するために使用されます。エラーを記録するには、パラメーター値を `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、パラメーター値を `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、パラメーター値にはそれらの特定の 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、オブジェクトタイプは、`create table`、`create view`、`drop role` などのコマンドに従って指定できます。「10」、「11」、「12」、「13」などのすべての変更ステートメントは、「14」を「15」に設定して監査します。特定のタイプの DML ステートメントを監査するために、パラメーター値に特定の SQL コマンド「16」、「17」、「18」、または「19」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

ター値「20」、「21」、「22」、または「23」を含めます。値を「24」に設定するとすべてのステートメントが監査され、「25」ではこの機能が無効になります。

3.1.3.7.10 edb_audit_tag

Parameter Type: スtring

Default Value: なし

Minimum Scope of Effect: セッション

When Value Changes Take Effect: 即時

Required Authorization to Activate: ユーザー

edb_audit_tag を使用して、edb_audit パラメーターが csv または xml に設定されている場合に監査ログに含まれる String 値を指定します。

3.1.3.7.11 edb_audit_destination

Parameter Type: 列挙

Default Value: ファイル

Range: {ファイル|syslog}

Minimum Scope of Effect: クラスタ

When Value Changes Take Effect: リロード

Required Authorization to Activate: 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 (デフォルト設定) 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 AdvancedServer 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 が 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。

3.1.3.7.12 edb_log_every_bulk_value

edb_log_every_bulk_value の詳細については、*edb_log_every_bulk_value* を参照してください。

3.1.3.8 クライアント接続のデフォルト/ロケールとフォーマット

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。

3.1.3.8.1 icu_short_form

Parameter Type: スtring

Default Value: なし

Range: 該当なし

Minimum Scope of Effect: データベース

When Value Changes Take Effect: 該当なし

Required Authorization to Activate: 該当なし

構成パラメーター `icu_short_form` は、データベースまたは `AdvancedServer` インスタンスに割り当てられたデフォルトの ICU 短縮名を含むパラメーターです。ICU の短い形式と Unicode 照合アルゴリズムに関する一般情報については、[Unicode 照合アルゴリズム](#) を参照してください。

この構成パラメーターは、`CREATE DATABASE` コマンドを `ICU_SHORT_FORM` パラメーターとともに使用すると設定されます。この場合、指定された短縮形名が設定され、このデータベースに接続するときに `icu_short_form` 構成パラメーターに表示されます。インスタンスは `--icu_short_form` オプションで使用する `initdb` コマンドで作成されます。この場合、指定された短い形式名が設定され、その `AdvancedServer` インスタンスのデータベースに接続されたときに `icu_short_form` 構成パラメーターに表示され、データベースは別の短い形式の独自の `ICU_SHORT_FORM` パラメータでオーバーライドしないでください。

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

3.1.3.9 クライアント接続のデフォルト/ステートメントの動作

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。

3.1.3.9.1 default_heap_fillfactor

Parameter Type: 整数

Default Value: 100

Range: 10 から 100

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

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、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）がデフォルトです。より小さな fillfactor が指定されている場合、INSERT 操作は指定された割合だけテーブルページをバックします。各ページの残りのスペースは、そのページの行を更新するために予約されています。これにより、UPDATE は更新された行のコピーを元のページと同じページに配置する機会を与えます。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。

3.1.3.9.2 edb_data_redaction

Parameter Type: ブール

Default Value: true

Range: {true,false}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

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、データ編集はスーパーユーザーとテーブル所有者を除くすべてのユーザーに適用されます。

- スーパーユーザーとテーブル所有者は、データ編集をバイパスします。
- 他のすべてのユーザーは、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、次のようになります。

- 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にはエラーが発生します。

データのリダクションについては、*EDBPostgresAdvancedServerSecurityFeaturesGuide* を参照してください。

3.1.3.10 クライアント接続のデフォルト/その他のデフォルト

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。

3.1.3.10.1 oracle_home

Parameter Type: スtring

Default Value: なし

Range: 該当なし

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: 再起動

Required Authorization to Activate: EPAS サービスアカウント

Oracle サーバーへの OracleCallInterface (OCI) データベースリンクを作成する前に、AdvancedServer を正しい Oracle ホームディレクトリに指定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 Oracle クライアントのインストールディレクトリの lib ディレクトリに設定します。

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

AdvancedServer を起動するたびに、Linux の LD_LIBRARY_PATH 環境変数 (Windows の「PATH」環境変数または oracle_home 構成パラメーター) を適切に設定する必要があります。

For Windows only: postgresql.conf ファイルに oracle_home 設定パラメータを設定するには、ファイルを編集して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 構成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再起動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。

3.1.3.10.2 odbc_lib_path

Parameter Type: スtring

Default Value: なし

Range: 該当なし

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: 再起動

Required Authorization to Activate: EPAS サービスアカウント

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

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

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

3.1.3.11 互換性オプション

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機能を制御します。

3.1.3.11.1 `edb_redwood_date`

Parameter Type: ブール

Default Value: false

Range: {true|false}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

コマンドの列のデータ型として `DATE` が表示される場合、構成パラメーター `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保存されるときに `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型はネイティブ PostgreSQL の `DATE` データ型のままになり、データベースにそのまま格納されます。PostgreSQL の `DATE` データ型は、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、`DATE` が SPL 宣言セクションの変数のデータ型、SPL プロシージャまたは SPL の仮パラメータのデータ型など、他のコンテキストでデータ型として表示される場合関数、または SPL 関数の戻り値型は、常に内部的に `TIMESTAMP` に変換されるため、存在する場合は時間コンポーネントを処理できます。

3.1.3.11.2 `edb_redwood_greatest_least`

Parameter Type: ブール

Default Value: true

Range: {true|false}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

GREATEST 関数は、パラメーターのリストから最大値を持つパラメーターを返します。LEAST 関数は、パラメーターのリストから最小の値を持つパラメーターを返します。

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

```
SET edb_redwood_greatest_least TO on;

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

greatest
-----

(1 row)
```

edb_redwood_greatest_least が FALSE に設定されている場合、すべてのパラメーターが null の場合を除き、null パラメーターは無視されます。その場合、関数によって 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)
```

3.1.3.11.3 edb_redwood_raw_names

Parameter Type: ブール

Default Value: false

Range: {true,false}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

edb_redwood_raw_names がデフォルト値の FALSE に設定されている場合、Redwood カタログから表示すると、テーブル名、列名、トリガー名、プログラム名、ユーザー名などのデータベースオブジェクト名が大文字で表示されます（つまり、ALL_、DBA_、または USER_ のプレフィックスが付いたシステムカタログ）。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ます。

edb_redwood_raw_names を TRUE に設定すると、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、引用符で囲まら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。引用符で囲まれて作成された名前は、作成されたとおりに表示されますが、引用符は表示されません。

たとえば、次のユーザー名が作成され、そのユーザー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);
```

Redwood カタログ USER_TABLES から表示すると、edb_redwood_raw_names がデフォルト値 FALSE に設定されているため、Mixed_Case 名を除き、名前は小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、作成時に表示される Mixed_Case 名を除き、名前は小文字で表示されますが、引用符は含まれません。

```
edb=> SET edb_redwood_raw_names TO true;
SET
edb=> SELECT * FROM USER_TABLES;
 schema_name | table_name | tablespace_name | status | temporary
```

(次のページに続く)

(前のページからの続き)

```

-----+-----+-----+-----+-----
reduser      | all_lower  |          | VALID | N
reduser      | all_upper  |          | VALID | N
reduser      | Mixed_Case |          | VALID | N
(3 rows)

```

これらの名前は、PostgreSQL の pg_tables カタログから見ると大文字と小文字が一致するようになりました。

```

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

```

3.1.3.11.4 edb_redwood_strings

Parameter Type: ブール

Default Value: false

Range: {true,false}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

edb_redwood_strings パラメーターが TRUE に設定されている場合、ストリングがヌル変数またはヌル列と連結されると、結果は元のストリングになります。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、文字列とヌル変数またはヌル列の連結によりヌル結果が得られることです。

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ヌルである comm という名前の列があります。次のクエリは、edb_redwood_string を FALSE に設定して実行されます。NULL 列と空でない文字列を連結すると、最終結果として NULL が生成されるため、クエリ結果には手数料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ヌルです。

```

SET edb_redwood_strings TO off;

SELECT RPAD(ename,10) || ' ' || TO_CHAR(sal,'99,999.99') || ' ' ||

```

(次のページに続く)

(前のページからの続き)

```
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連結すると、空でない文字列が生成されます。

```
SET edb_redwood_strings TO on;
```

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

```
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)
```

3.1.3.11.5 edb_stmt_level_tx

Parameter Type: ブール

Default Value: false

Range: {true,false}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

statement level transaction isolation という用語は、SQL コマンドで実行時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。たとえば、単一の UPDATE コマンドが 5 行を正常に更新したが、6 行目を更新しようとする例外が発生した場合、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。

AdvancedServer では、SQL コマンドの実行中に例外が発生すると、トランザクションの開始以降のデータベース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の更新をロールバックします。

注釈: パフォーマンスに悪影響を与える可能性があるため、絶対に必要な場合にのみ edb_stmt_level_tx を TRUE に設定してください。

PSQL で実行される次の例は、edb_stmt_level_tx が FALSE の場合、2 番目の INSERT コマンドの中止によって最初の 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ることに注意してください。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"
```

(次のページに続く)

(前のページからの続き)

```
DETAIL:  Key (deptno)=(0) is not present in table "dept".
```

```
COMMIT;
```

```
SELECT empno, ename, deptno FROM emp WHERE empno > 9000;
```

```
empno  | ename | deptno
-----+-----+-----
(0 rows)
```

次の例では、`edb_stmt_level_tx` が TRUE に設定されているため、最初の INSERT コマンドは、2 番目の INSERT コマンドのエラー後に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。

3.1.3.11.6 db_dialect

Parameter Type: 列挙

Default Value: postgres

Range: {postgres|レッドウッド}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

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

`postgres` に設定すると、ネームスペースの優先順位は `pg_catalog`、`sys` となり、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`、`pg_catalog` となり、展開されたカタログビューの優先順位が最も高くなります。

3.1.3.11.7 default_with_rowids

Parameter Type: ブール

Default Value: false

Range: {true,false}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

`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 コマンドで参照できます。AdvancedServer の以前のバージョンでは `ROWIDs` は `OIDs` にマッピングされていましたが、AdvancedServer バージョン 12 以降では、`ROWID` は 1 で始まり、テーブルの各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 `ROWIDs` オプションで作成。デフォルトでは、`ROWID` 列に一意的な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。

AdvancedServer11 以前のバージョンから `ROWIDs` で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、AdvancedServer 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. アップグレードが正常に完了したら、ダンプを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復元します。

3.1.3.11.8 optimizer_mode

Parameter Type: 列挙

Default Value: を選択

Range: { 選択|ALL_ROWS|FIRST_ROWS|FIRST_ROWS_10|FIRST_ROWS_100|FIRST_ROWS_1000 }

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

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

次の表は、可能な値を示しています。

ヒント	説明
ALL_ROWS	結果セットのすべての行の取得用に最適化します。
選ぶ	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を行いません。これがデフォルトです。
FIRST_ROWS	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。
FIRST_ROWS_10	結果セットの最初の 10 行の取得のために最適化します。
FIRST_ROWS_100	結果セットの最初の 100 行を取得するために最適化します。
FIRST_ROWS_1000	結果セットの最初の 1000 行の取得のために最適化します。

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

3.1.3.12 カスタマイズされたオプション

AdvancedServer 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。

3.1.3.12.1 custom_variable_classes

AdvancedServer9.2 では、custom_variable_classes パラメーターは非推奨です。以前にこのパラメーター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。

3.1.3.12.2 dbms_alert.max_alerts

Parameter Type: 整数

Default Value: 100

Range: 0 から 500

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: 再起動

Required Authorization to Activate: 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。

3.1.3.12.3 dbms_pipe.total_message_buffer

パラメータタイプ: 整数

Default Value: 30Kb

Range: 30Kb から 256Kb

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: 再起動

Required Authorization to Activate: 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。

3.1.3.12.4 index_advisor.enabled

Parameter Type: ブール

Default Value: true

Range: {true,false}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

EDB-PSQL または PSQL セッションでインデックスアドバイザーを一時的に中断する機能を提供します。
index_advisor.enabled 構成パラメーターを使用するには、EDB-PSQL または PSQL セッションに Index-Advisor プラグイン index_advisor をロードする必要があります。

IndexAdvisor 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 (13.0.0)
Type "help" for help.

edb=# LOAD 'index_advisor';
LOAD
```

SET コマンドを使用してパラメーター設定を変更し、次の例に示すように、インデックスアドバイザー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
-----
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)
```

3.1.3.12.5 edb_sql_protect.enabled

Parameter Type: ブール

Default Value: false

Range: {true,false}

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: リロード

Required Authorization to Activate: 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監視するかどうかを制御します。こ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 で監視を開始する準備ができれば、このパラメーターをオンに設定します。

3.1.3.12.6 edb_sql_protect.level

Parameter Type: 列挙

Default Value: パッシブ

Range: {学ぶ|パッシブ|アクティブ}

Minimum Scope of Effect: クラスタ

When Value Changes Take Effect: リロード

Required Authorization to Activate: EPAS サービスアカウント

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

`edb_sql_protect.level` 構成パラメータは、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するために、次の値のいずれかに設定できます。

- **learn.** 保護されたロールのアクティビティを追跡し、ロールによって使用される関係を記録します。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本番環境で実行できます。
- **active.** 保護されたロールの無効なステートメントをすべて停止します。これは、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するため、攻撃者がシステムに侵入する別の方法を見つける前に管理者に警告することができます。

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。

3.1.3.12.7 edb_sql_protect.max_protected_relations

Parameter Type: 整数

Default Value: 1024

Range: 1 から 2147483647

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: 再起動

Required Authorization to Activate: EPAS サービスアカウント

役割ごとに保護できる関係の最大数を設定します。サーバーの保護された関係の総数は、保護された関係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注意してください。保護された関係はすべて、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関係のスペースは、データベースサーバーの起動時に予約されます。

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

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

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

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

共有メモリに必要なスペースを確保できる限り、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ーの起動は次のようなエラーメッセージで失敗します。

```
2014-07-18 15:22:17 EDT FATAL: could not map anonymous shared memory:
Cannot allocate memory
2014-07-18 15:22:17 EDT HINT: This error usually means that 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ーを正常に起動できるようになるまでパラメーター値を減らします。

3.1.3.12.8 `edb_sql_protect.max_protected_roles`

Parameter Type: 整数

Default Value: 64

Range: 1 から 2147483647

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: 再起動

Required Authorization to Activate: 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関係の数（`edb_sql_protect.max_protected_relations`）を掛けたスペースを予約することに注意してください。可能な限り最大の保護されたロールのスペースは、データベースサーバーの起動時に確保されます。

`edb_sql_protect.max_protected_roles` が有効な範囲外の値（たとえば、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ーが起動すると、警告メッセージがデータベースサーバー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_roles": "2147483648"
2014-07-18 16:04:12 EDT HINT: Value exceeds integer range.
```

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

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

共有メモリに必要なスペースを確保できる限り、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ーの起動は次のようなエラーメッセージで失敗します。

```
2014-07-18 15:22:17 EDT FATAL: could not map anonymous shared memory:
Cannot allocate memory
2014-07-18 15:22:17 EDT HINT: This error usually means that 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ーを正常に起動できるようになるまでパラメーター値を減らします。

3.1.3.12.9 edb_sql_protect.max_queries_to_save

Parameter Type: 整数

Default Value: 5000

Range: 100 から 2147483647

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: 再起動

Required Authorization to Activate: 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は、共有メモリの領域を消費します。保存可能なクエリの最大可能容量は、データベースサーバーの起動時に確保されます。

`edb_sql_protect.max_queries_to_save` が有効範囲外の値（たとえば、値 10）に設定されているときにデータベースサーバーが起動すると、警告メッセージがデータベースサーバーのログファイルに記録されます。

```
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ーは正常に起動しますが、`edb_sql_protect.max_queries_to_save` がデフォルト値の 5000 に設定されています。

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

共有メモリに必要なスペースを確保できる限り、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ーの起動は次のようなエラーメッセージで失敗します。

```
2014-07-18 15:22:17 EDT FATAL: could not map anonymous shared memory:
Cannot allocate memory
2014-07-18 15:22:17 EDT HINT: This error usually means that 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ーを正常に起動できるようになるまでパラメーター値を減らします。

3.1.3.12.10 `edb_wait_states.directory`

Parameter Type: スtring

Default Value: `edb_wait_states`

Range: 該当なし

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: 再起動

Required Authorization to Activate: 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、パフォーマンス分析とチューニング <performance_analysis_and_tuning> の ‘EDB Wait States’ を参照してください。

3.1.3.12.11 `edb_wait_states.retention_period`

Parameter Type: 整数

Default Value: 604800s

Range: 86400s から 2147483647s

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: リロード

Required Authorization to Activate: 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、パフォーマンス分析とチューニング <performance_analysis_and_tuning> の ‘EDB Wait States’ を参照してください。

3.1.3.12.12 `edb_wait_states.sampling_interval`

Parameter Type: 整数

Default Value: 1 秒

Range: 1 秒から 2147483647 秒

Minimum Scope of Effect: クラスター

When Value Changes Take Effect: リロード

Required Authorization to Activate: 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間隔を設定します。デフォルト設定は 1 秒です。EDB の待機状態については、パフォーマンス分析とチューニング <performance_analysis_and_tuning> の ‘EDB Wait States’ を参照してください。

3.1.3.12.13 `edbldr.empty_csv_field`

Parameter Type: 列挙

Default Value: NULL

Range: {NULL|empty_string|pgsql}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

edbldr.empty_csv_field パラメーターを使用して、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。

パラメータ設定	EDB*Loader の動作
NULL	空のフィールドは NULL として扱われます。
empty_string	空のフィールドは、長さがゼロの文字列として扱われます。
pgsql	空のフィールドは、引用符が含まれていない場合は NULL として扱われ、引用符が含まれている場合は空の文字列として扱われます。

EDB*Loader の edbldr.empty_csv_field パラメータの詳細については、EDBWeb 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/edb-docs&https://www.enterprisedb.com/resources/product-documentation’](https://www.enterprisedb.com/edb-docs&https://www.enterprisedb.com/resources/product-documentation) **__**

3.1.3.12.14 utl_encode.uudecode_redwood

Parameter Type: ブール

Default Value: false

Range: {true|false}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

TRUE に設定すると、AdvancedServer の UTL_ENCODE.UUDECODE 機能は、UTL_ENCODE.UUENCODE 機能の Oracle 実装によって作成された uuencode されたデータをデコードできます。

デフォルト設定の FALSE に設定すると、AdvancedServer の UTL_ENCODE.UUDECODE 機能は、AdvancedServer の UTL_ENCODE.UUENCODE 機能によって作成された uuencode されたデータをデコードできます。

3.1.3.12.15 utl_file.umask

Parameter Type: スtring

Default Value: 0077**Range:** umask 設定の 8 進数**Minimum Scope of Effect:** セッションごと**When Value Changes Take Effect:** 即時**Required Authorization to Activate:** セッションユーザー

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*mask* を設定します。これは、AdvancedServer UTL_FILE パッケージ内でのみ使用されます。

注釈: utl_file.umask パラメーターは、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 関数および手順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源を参照してください。)

次に、デフォルトの utl_file.umask 設定の 0077 の結果を示します。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権限が拒否されることに注意してください。ユーザー enterprisedb のみがファイルの読み取りおよび書き込み権限を持っています。

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

3.1.3.13 グループ化されていない

このセクションの構成パラメーターは AdvancedServer 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。

3.1.3.13.1 nls_length_semantics

Parameter Type: 列挙**Default Value:** バイト**Range:** {バイト|char}**Minimum Scope of Effect:** セッションごと**When Value Changes Take Effect:** 即時**Required Authorization to Activate:** スーパーユーザー

このパラメーターは、AdvancedServer では効果がありません。

たとえば、ALTER SESSION コマンドの形式は、構文エラーをスローすることなく AdvancedServer で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 nls_length_semantics = char;
```

注釈: このパラメーター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。

3.1.3.13.2 query_rewrite_enabled

Parameter Type: 列挙

Default Value: false

Range: {true|偽|力}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

このパラメーターは、AdvancedServer では効果がありません。

たとえば、次の形式の ALTER SESSION コマンドは、構文エラーをスローすることなく AdvancedServer で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;
```

注釈: このパラメーター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。

3.1.3.13.3 query_rewrite_integrity

Parameter Type: 列挙

Default Value: 施行

Range: {強制信頼できる|stale_tolerated}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: スーパーユーザー

このパラメーターは、AdvancedServer では効果がありません。

たとえば、次の形式の ALTER SESSION コマンドは、構文エラーをスローすることなく AdvancedServer で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;
```

注釈: このパラメーター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。

3.1.3.13.4 timed_statistics

Parameter Type: ブール

Default Value: true

Range: {true,false}

Minimum Scope of Effect: セッションごと

When Value Changes Take Effect: 即時

Required Authorization to Activate: セッションユーザー

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

注釈: AdvancedServer をインストールすると、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が含まれます。このエントリ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、時限統計収集がオンになります。

3.2 インデックスアドバイザー

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 ツリー（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。IndexAdvisor は EDBPostgresAdvancedServer と共にインストールされます。

IndexAdvisor は AdvancedServer のクエリプランナーと連携し、クエリプランナーがそのようなインデックスが利用可能であるかのように実行コストを計算するために使用する*仮説**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。

IndexAdvisor 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。

- IndexAdvisor 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で分析するクエリを EDB-PSQL コマンドラインで提供します。
- PostgresEnterpriseManager クライアントから IndexAdvisor 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て、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 EDBWeb 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/edb-docs><<https://www.enterprisedb.com/resources/product-documentation>>‘ __

インデックスアドバイザーは、INSERT、UPDATE、DELETE、SELECT ステートメントでインデックスの推奨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場合、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）の形式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、提供されたクエリのクエリプランと推定実行コストを表示しますが、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、クエリ実行コストと仮想インデックスの有無を比較します。仮想インデックスを使用した実行コストがそれを使用しない場合の実行コストよりも小さい場合、両方のプランが EXPLAIN ステートメント出力で報告され、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、作成に必要な CREATE INDEX ステートメントを生成しますインデックス。

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

IndexAdvisor 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。

AdvancedServer 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、スクリプトは関数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期間中のみ利用可能な一時テーブルに推奨事項を記録します。

3.2.1 インデックスアドバイザーのコンポーネント

インデックスアドバイザーの共有ライブラリは、クエリプランナーと対話して、インデックス作成の推奨事項を作成します。Windows では、AdvancedServer インストーラーは、AdvancedServer ホームディレクトリの libdir 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、xx が AdvancedServer バージョン番号である edb-asxx-server-indexadvisor RPM パッケージをインストールします。

Linux の場合:

```
index_advisor.so
```

Windows の場合:

```
index_advisor.dll
```

libdir ディレクトリ内のライブラリは、スーパーユーザーのみがロードできることに注意してください。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「AdvancedServer ホームディレクトリの下」の libdir/plugins ディレクトリ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、実稼働環境では安全ではありません。

インストーラーは、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、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。このファイルを使用して、インデックスアドバイザーが推奨するインデックスを作成できます。pg_advise_index プログラムは、AdvancedServer ホームディレクトリの bin 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機能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。このスクリプトは、AdvancedServer 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() 関数、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、インデックス作成の推奨事項を 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、特定のインデックスアドバイザーセッション中に行われた推奨事項を解釈して表示する PL/pgSQL 関数です（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、クエリ分析中に `index_advisor_log` テーブルに格納された情報に基づいて `index_recommendations` ビューを作成します。ビューは `show_index_recommendations()` 関数と同じ形式で出力を生成しますが、`show_index_recommendations()` 関数によって返される結果セットは指定されたセッションに制限されますが、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 インデックスアドバイザーの構成

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。複数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。`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、関数、およびビューのストレージスキーマを選択する場合、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する）すべてのユーザーがスキーマに対する `USAGE` 権限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してい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` パラメーターの先頭に配置します。たとえば、検索パスが現在次の場合:

```
search_path=public, accounting
```

インデックスアドバイザーオブジェクトを `advisor` という名前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 `IndexAdvisor` ユーザーに付与します。`IndexAdvisor` ユーザーがスーパーユーザーまたはこれらのデータベースオブジェクトの所有者である場合、この手順は不要です。

- `index_advisor_log` テーブルに `SELECT` および `INSERT` 特権を付与して、ユーザーがインデックスアドバイザーを起動できるようにします。
- `index_advisor_log` テーブルに `DELETE` 権限を付与して、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` ビューに `SELECT` 権限を付与します。

次の例は、`ia` という名前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。このスキーマオブジェクトには、ユーザー名 `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 (13.0.0, server 13.0.0)
Type "help" for help.

edb=# CREATE SCHEMA ia;
CREATE SCHEMA
edb=# SET search_path TO ia;
SET
edb=# \i /usr/edb/as13/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 パラメーターに含める必要があります。

3.2.3 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（コマンドラインで指定）、またはクエリのセットを含むファイル（`pg_advise_index()` 関数によって実行される）です。ワークロードの分析後、`IndexAdvisor` は結果セットを一時テーブルまたは永続テーブルのいずれかに保存します。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作成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。

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

次の例では、スーパーユーザー `enterprisedb` がインデックスアドバイザーユーザーであり、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ー `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
1	99998
2	99997
3	99996
.	.
.	.
.	.
99997	2
99998	1
99999	0

3.2.3.1 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場合、`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` キーワードで開始しないでください。

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_advice_index プログラムを実行します。

```
$ pg_advice_index -d edb -h localhost -U enterprisedb -s 100M -o
advisory.sql workload.sql
poolsize = 102400 KB
ad 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、IndexAdvisor が推奨するインデックスの最大サイズを制限するオプションのパラメーターです。インデックスアドバイザーが結果セットを返さない場合、-s の設定が低すぎる可能性があります。

-o

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

pg_advice_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、IndexAdvisor 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
```

3.2.3.2 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、IndexAdvisor プラグインのロードと IndexAdvisor 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、IndexAdvisor 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る各 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、クエリに関する推奨事項を index_advisor_log テーブルに保存します。index_advisor_log テーブルがユーザーの search_path に存在しない場合、一時テーブルが同じ名前で作成されます。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。

IndexAdvisor プラグインをロードした後、IndexAdvisor 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記録します。

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

ステートメントの前に EXPLAIN キーワードを付けない場合、IndexAdvisor 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 index_advisor_log テーブルにインデックスの推奨事項を書き込みます。

次の例では、EXPLAIN ステートメントは、クエリが推奨仮想インデックスを使用している場合、通常のクエリプランを表示し、その後に同じクエリ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)
```

(次のページに続く)

(前のページからの続き)

```
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)
```

IndexAdvisor プラグインをロードした後、`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、IndexAdvisor プラグインをロードする必要があります。

`index_advisor.enabled` パラメーターを使用して、psql セッションを中断せずに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、パラメーターを `on` に設定します。

```
edb=# SET index_advisor.enabled TO on;
SET
```

3.2.4 インデックスアドバイザーの推奨事項の確認

IndexAdvisor 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` 機能を実行します。
- `index_recommendations` ビューを照会します。

3.2.4.1 `show_index_recommendations ()` 関数を使用する

`show_index_recommendations ()` 関数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 を指定して関数を呼び出します。

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

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

次のコードは、結果セットの行の例を示しています。

```
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 IndexAdvisor によって提案されたインデックスを作成するために必要な SQL ステートメントです。結果セットの各行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する計算された利点。
- 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;
```

3.2.4.2 index_advisor_log テーブルのクエリ

インデックスアドバイザーは、インデックスの推奨事項を `index_advisor_log` という名前のテーブルに保存します。`index_advisor_log` テーブルの各行には、クエリの実行コストを削減するために仮想インデックスを推奨できるとインデックスアドバイザーが判断したクエリの結果が含まれます。

コラム	種類	説明
レロイド	oid	インデックスのベーステーブルの OID
relname	お名前	インデックスのベーステーブルの名前
attrs	整数 []	推奨インデックス列 (列番号で識別)
恩恵	本当の	このクエリのインデックスの計算された利点
index_size	整数	ディスクページの推定インデックスサイズ
backend_pid	整数	この推奨事項を生成するプロセスのプロセス ID
タイムスタンプ	タイムスタンプ	推奨が生成された日付/時刻

`psql` コマンドラインで `index_advisor_log` テーブルを照会できます。次の例は、2 つの `IndexAdvisor` セッションの結果の `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ており、異なる `backend_pid` 値によって (下の表で) 識別できます。各セッションで、`IndexAdvisor` 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た次の 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、これらの結果がプロセス ID 3442 のセッションからのものであることを示しています。

最初の行の列 `attrs` の 1 の値は、仮想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 行目の `attrs` 列の 2 の値は、仮想インデックスがテーブルの 2 列目（テーブル `t` の `b` 列）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。

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

「例」に示す `index_advisor_log` テーブルの最後の行の `benefit` 列の値は、次の SQL ステートメントのクエリプランを使用して計算されます。

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

さまざまな実行計画の実行コストが評価され、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。

3.2.4.3 index_recommendations ビューのクエリ

`index_recommendations` ビューには、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おり、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します。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 `IndexAdvisor` 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` ビューには次のように表示されます。

```
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せられて、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` という名前のフィールドに反映されます。

フィールドの式は次のとおりです。

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

たとえば、`backend_pid` が 3506 であるプロセスの次のクエリ結果を使用します。

```
relroid | 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導出されたさまざまな推奨インデックスの相対的な利点を比較する場合に役立ちます。ゲイン値が大きいほど、インデックスから得られる費用対効果は、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 インデックスアドバイザーの制限

IndexAdvisor は IndexOnly 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どのような種類の表現でもありません。フィールドは単純な列名になります。

仮想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しません。

index_advisor_log テーブルまたは index_advisor_log テーブルにインデックス作成の推奨事項が作成され保存されたテーブルを含む pg_dump バックアップファイルを復元すると、index_advisor_log テーブルとオブジェクト識別子 (OID) の変更により、index_advisor_log テーブルの行によって参照される復元されたテーブル。

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 を参照先テーブルの新しい 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 プロファイラー

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、大規模で複雑なシステムでこのコードを見つけて最適化することです。

SQLProfiler 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。

SQLProfiler の特定の機能と利点は次のとおりです。

- **On-Demand Traces.** パラメータを手動で設定して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。
- **Scheduled Traces.** 不便な場合は、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。
- **Save Traces.** トレースを実行し、後で確認できるように保存します。
- **Trace Filters.** データベースおよびユーザーごとに SQL キャプチャを選択的にフィルタリングするか、すべてのデータベースに対してすべてのユーザーから送信されたすべての SQL ステートメントをキャプチャします。
- **Trace Output Analyzer.** グラフィカルテーブルを使用すると、クエリを期間またはステートメントですばやく並べ替えてフィルタリングできます。グラフィカルまたはテキストベースの EXPLAIN プランでは、クエリパスと結合がレイアウトされます。
- **Index Advisor Integration.** 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。

Step 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 の AdvancedServer インストーラーまたは Linux の `edb-asxx-server-sqlprofiler` RPM パッケージからインストールされます。xx は AdvancedServer のバージョン番号です。

Step 2: SQL プロファイラ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`” 構成パラメータに SQL プロファイラライブラリを含めます。

Linux インストールの場合、パラメーター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ている必要があります。

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

Step 3: SQL プロファイラが使用する関数を作成します

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

Linux の場合:

```
/usr/edb/as13/share/contrib/
```

Windows の場合:

```
C:\Program Files\edb\as13\share\contrib\
```

psql コマンドラインインターフェイスを使用して、プロファイルするサーバー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。AdvancedServer を使用している場合、デフォルトのメンテナンスデータベースの名前は edb です。PostgreSQL インスタンスを使用している場合、デフォルトのメンテナンスデータベースの名前は postgres です。

次のコマンドは、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/as13/bin/psql -U user_name database_name <
/usr/edb/as13/share/contrib/sql-profiler.sql
```

Step 4: サーバーを停止して再起動すると、変更が有効になります。

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えるようになります。EDBPostgresEnterpriseManager で SQL プロファイラー機能を利用できます。PostgresEnterpriseManager の詳細については、EDBWeb サイトにアクセスしてください。

<https://www.enterprisedb.com/edb-docs><<https://www.enterprisedb.com/resources/product-documentation>>‘__

Troubleshooting

(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、既存のクエリセットを新しいクエリセットに置き換える必要があります。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

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

AdvancedServer 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/as13/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ドキュメント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は snmpd.conf です。構成ファイルで指定できるディレクティブについては、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。none を入力して、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ない)。

snmpconf はメニュー方式のウィザードです。メニュー項目 1:snmpd.conf を選択して、構成ウィザードを開始します。snmpconf で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、ウィザードは一連の質問を順番に実行し、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 を入力して snmpd.conf という名前の構成ファイルを生成します。ファイルを次の場所にコピーします。

```
/usr/edb/as13/share/
```

3.4.2 リスナーアドレスの設定

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

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

snmpd.conf ファイルに次の行を追加することにより、代替リスナーポートを指定できます。

```
agentaddress $host_address:2000
```

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

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

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

3.4.3 pgsnmpd の呼び出し

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

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

POSTGRES_INSTALL_HOME は AdvancedServer 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 がバックグラウンドで実行されるように指定します。

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

AdvancedServer のインストールの接続情報（libpq 接続文字列の形式）を -C オプションの後に含めます。

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表示する場合は、`--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 サービスを照会できます。例:

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

上記の例では:

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

`-c public` はコミュニティ名を指定します。

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

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

特定のオブジェクトのクエリに必要なエンコーディングは、**MIB** (管理情報ベース) で定義されています。**SNMP** クライアントは、さまざまなサーバーを監視できます。サーバーの種類によって、特定のサーバーによって公開される情報が決まります。各 **SNMP** サーバーは、公開されたデータを **MIB** (管理情報ベース) の形式で記述します。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 監査ログ

AdvancedServer 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 監査ログ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、次のような情報を記録するように構成できます。

- ロールが AdvancedServer データベースへの接続を確立するとき
- AdvancedServer 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
- 失敗した認証試行が発生したとき

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 監査ログ構成パラメーター

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。[構成パラメータの概要](#) を参照して、構成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必要があるか、またはデータベースサーバーを再起動する必要があるかを確認してください。

`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、これもデフォルトです。

`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を指定します。デフォルトのファイル名は `audit-%Y%m%d_%H%M%S` です。エスケープシーケンス `%Y`、`%m` などは、システムの日付と時刻に従って適切な現在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ションする曜日を指定します。有効な値は `sun`、`mon`、`tue`、`wed`、`thu`、`fri`、`sat`、「12」、および `none` です。回転を無効にするには、値を `none` に設定します。ファイルを毎日ローテーションするには、「10」の値を「12」に設定します。特定の曜日にファイルをローテーションするには、値を目的の曜日に設定します。「12」がデフォルト値です。

`edb_audit_rotation_size`

ファイルのローテーション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。パラメーター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、このパラメーターをデフォルトの 0 に設定します。

edb_audit_connect

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

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、値をデフォルトの `none` に設定します。

edb_audit_statement

この構成パラメーターは、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。`none`、`dml`、`insert`、`update`、`delete`、`truncate`、`select`、「17」、`rollback`、「16」、「10」、「11」、「12」、「13」、「14」、および「15」。デフォルトは「16」と「17」です。このパラメーターの設定に関する情報については、[監査する SQL ステートメントの選択](#) を参照してください。

edb_audit_tag

この構成パラメーターを使用して、各エントリの監査ログファイルに追跡タグとして含める文字列値を指定します。

edb_log_every_bulk_value

一括処理は、結果のステートメントを `AdvancedServer` ログファイルと `EDBAudit` ログファイルの両方に記録します。ただし、一括処理ですべてのステートメントをログに記録するにはコストがかかります。これは、`edb_log_every_bulk_value` 構成パラメーター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。デフォルト設定である `edb_audit_directory` で指定されたディレクトリを使用するには、`file` に設定します。

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

次のセクションでは、edb_audit_statement パラメーター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 監査する SQL ステートメントの選択

edb_audit_statement 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含めることを許可します。一般的な形式は次のとおりです。

```
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ー値は次のとおりです。

- **all** –ステートメントに関する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、DROP) および GRANT および REVOKE データ制御言語 (DCL) ステートメントの監査を行います。
- **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) の監査になります。
- **select** –SELECT ステートメントの監査になります。
- **rollback** –ROLLBACK ステートメントの監査になります。
- **error** –発生したすべてのエラーメッセージがログに記録されます。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ー値に関連する SQL ステートメントで発生したエラーに関するエラーメッセージは記録されません。

Data Definition Language and Data Control Language Statements は、監査用に特定の 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ー値を示します。

Data Manipulation Language Statements は、監査用の特定の DML ステートメントを選択するための追加のパラメーター値を示します。

サポートされていない値が `edb_audit_statement` パラメーターに含まれている場合、データベースサーバー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ー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/as13/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。

3.5.2.1 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` パラメーター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` パラメーターに `ddl` または `all` のいずれか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 または DCL ステートメントは監査されません。
- `edb_audit_statement` パラメーター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に選択できます。

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

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

`object_type` は次のいずれかです。

ACCESS METHOD

AGGREGATE

CAST

COLLATION

CONVERSION

DATABASE

EVENT TRIGGER

EXTENSION

FOREIGN TABLE

FUNCTION

INDEX

LANGUAGE

LARGE OBJECT

MATERIALIZED VIEW

OPERATOR
OPERATOR CLASS
OPERATOR FAMILY
POLICY
PUBLICATION
ROLE
RULE
SCHEMA
SEQUENCE
SERVER
SUBSCRIPTION
TABLE
TABLESPACE
TEXT SEARCH CONFIGURATION
TEXT SEARCH DICTIONARY
TEXT SEARCH PARSER
TEXT SEARCH TEMPLATE
TRANSFORM
TRIGGER
TYPE
USER MAPPING
VIEW

SQL コマンドで使用するオブジェクトタイプの説明は、次の URL で入手可能な PostgreSQL コアドキュメントに記載されています。

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

パラメーター値から `object_type` が省略されている場合、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。

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

```
{ grant | revoke }
```

以下は、DDL と DCL の例です。

Example 1

以下は、edb_audit_connect と edb_audit_statement に次のデフォルト以外の値が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 enterisedb
Password for user enterisedb:
psql.bin (13.0.0)
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
edb=# \c auditdb adminuser
```

(次のページに続く)

(前のページからの続き)

```

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複数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 12:32:22.799 IST,"enterprisedb","edb",72518,"[local]",5ecb6d7e.
11b46,1,"authentication",2020-05-25 12:32:22 IST,4/19,0,AUDIT,00000,
"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,00000,"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,00000,"statement: ALTER
ROLE adminuser WITH LOGIN SUPERUSER PASSWORD 'password';",,,,,,,,,,"psql",
"client backend","ALTER ROLE","alter"

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

```

(次のページに続く)

(前のページからの続き)

```

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 パラメーターに含まれているため、ALTER ROLE adminuser ステートメントのエラーもログに記録されます。

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

ddl、all、drop など、正常に処理された DROP ステートメントを監査する 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。

Example 2

以下は、edb_audit_connect と edb_audit_statement に次のデフォルト以外の値が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、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 (13.0.0)
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
auditdb=# CREATE MATERIALIZED VIEW managers AS
auditdb-#      SELECT empno, ename, hiredate, sal, comm FROM emp WHERE job =
```

(次のページに続く)

(前のページからの続き)

```
'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複数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中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,00000,
"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,00000,"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,00000,"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,00000,"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,00000,"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,00000,"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、all のいずれの値も edb_audit_statement パラメーターに含まれていないため、前の CREATE TABLE emp ステートメントは監査され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、これらの値が `edb_audit_statement` パラメーターに含まれているため、監査されます。

3.5.2.2 データ操作言語ステートメント

このセクションでは、DML ステートメントを監査するために `edb_audit_statement` パラメーター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` パラメーターに `dml` または `all` のいずれか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` パラメーター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 ステートメントに `edb_audit_statement` パラメーター値を指定します。

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

Example

以下は、`edb_audit_connect` と `edb_audit_statement` に次のデフォルト以外の値が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 (13.0.0)
Type "help" for help.

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

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

(次のページに続く)

(前のページからの続き)

```

UPDATE, DELETE, error
(1 row)

auditdb=# SET search_path TO edb;
SET
auditdb=# INSERT INTO dept VALUES (10,'ACCOUNTING','NEW YORK');
INSERT 0 1
auditdb=# INSERT INTO dept VALUES (20,'RESEARCH','DALLAS');
INSERT 0 1
auditdb=# INSERT INTO dept VALUES (30,'SALES','CHICAGO');
INSERT 0 1
auditdb=# INSERT INTO dept VALUES (40,'OPERATIONS','BOSTON');
INSERT 0 1
auditdb=# INSERT INTO emp VALUES (7369,'SMITH','CLERK',7902,'17-DEC-
80',800,NULL,20);
INSERT 0 1
auditdb=# INSERT INTO emp VALUES (7499,'ALLEN','SALESMAN',7698,'20-FEB-
81',1600,300,30);
INSERT 0 1
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複数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中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,00000,
"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,00000,"statement: UPDATE
dept SET loc = 'BEDFORD' WHERE deptno = 40;",,,,,,,,,,"psql","client backend",
"UPDATE","update"

2020-05-25 15:19:38.524 IST,"adminuser","auditdb",123420,"[local]",5ecb9424.
1e21c,3,"idle",2020-05-25 15:17:16 IST,5/16,0,AUDIT,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、all のいずれの値も edb_audit_statement パラメーターに含まれていないため、以前の INSERT ステートメントはすべて監査されないことに注意してください。

edb_audit_statement パラメーターに select も all も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 が edb_audit_statement パラメーターで指定されているが truncate 値ではないため、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 監査ログを有効にする

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 AdvancedServer を構成する方法について説明します。

1. edb_audit パラメーターを xml または csv に設定して、監査を有効にします。
2. 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。
3. すべての接続を監査するには、パラメーター edb_audit_connect を all に設定します。
4. すべての切断を監査するには、パラメータ edb_audit_disconnect を all に設定します。

5. DDL、DCL、DML およびその他のステートメントを監査するには、監査する SQL ステートメントの選択 `<selecting_sql_statements_to_audit>` の指示に従って、パラメーター `‘edb_audit_statement’` を設定します。

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

`edb_audit_statement` パラメーター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` パラメーターは、`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` に接続したときに実行されるステートメントの構成ファイルの `edb_audit_statement` パラメーターをオーバーライドします。
- `edb_audit_statement` パラメーターは、`ALTER ROLE rolename SET edb_audit_statement` コマンドを使用して、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、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だけでなく、構成ファイルの `edb_audit_statement` パラメーターをオーバーライドします。
- `edb_audit_statement` パラメーターは、`ALTER ROLE rolename IN DATABASE dbname SET edb_audit_statement` コマンドで指定されたデータベースを使用するときに、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` パラメーターと、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 `IN DATABASE` 句なしで `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。前述のとおり。

以下は、この手法の例です。

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

```
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, all
```

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

- データベース `auditdb` と `ddl`、`insert`、`update`、`delete`
- ロール `admin` と `select` および `truncate`
- データベースのロール `admin auditdb` と `create table`、`insert`、`update`

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 (13.0.0)
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
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。

Case 1: ロール enterprisedb によってデータベース auditdb に加えられた変更。ddl, insert, update および delete ステートメントのみが監査されます。

```

$ psql auditdb enterprisedb
Password for user enterprisedb:
psql.bin (13.0.0)
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;
 f1 | f2
----+----
  1 | Row A
(1 row)

```

<== Should not be audited

(次のページに続く)

(前のページからの続き)

```

-----+-----
      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複数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中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,00000,"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,00000,"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,00000,"statement: UPDATE
audit_tbl SET f2 = 'Row A' WHERE f1 = 1;",,,,,,,,,,"psql","client backend",
"UPDATE","", "update"

```

Case 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 および truncate 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 (13.0.0)
Type "help" for help.

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

edb=# CREATE TABLE edb_tbl (f1 INTEGER PRIMARY KEY, f2 TEXT); <== Should not
be audited
CREATE TABLE

```

(次のページに続く)

(前のページからの続き)

```

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、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 と INSERT INTO edb_tbl statements 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,00000,"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,00000,"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,00000,"statement: UPDATE
audit_tbl SET f2 = 'Row A' WHERE f1 = 1;",,,,,,,,,,"psql","client backend",
"UPDATE","", "update"

2020-05-25 16:25:37.509 IST,"admin","edb",71920,"[local]",5ecba3fe.118f0,2,
"idle",2020-05-25 16:24:54 IST,4/33,0,AUDIT,00000,"statement: SELECT * FROM
edb_tbl;",,,,,,,,,,"psql","client backend","SELECT","", "select"

2020-05-25 16:25:46.131 IST,"admin","edb",71920,"[local]",5ecba3fe.118f0,3,
"idle",2020-05-25 16:24:54 IST,4/34,0,AUDIT,00000,"statement: TRUNCATE edb_
tbl;",,,,,,,,,,"psql","client backend","TRUNCATE TABLE","", "truncate"

```

Case 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 (13.0.0)
Type "help" for help.

```

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

auditdb=# TRUNCATE audit_tbl_2;
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 番目のエントリには、CREATE TABLE audit_tbl_2 および INSERT INTO audit_tbl_2 ステートメントのみが表示されます。SELECT * FROM audit_tbl_2 および TRUNCATE audit_tbl_2 ステートメントは監査されませんでした。

```

2020-05-25 15:59:12.332 IST,"enterprisedb","auditdb",41837,"[local]",5ecb9dee.
a36d,2,"idle",2020-05-25 15:59:02 IST,5/7,0,AUDIT,00000,"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,00000,"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,00000,"statement: UPDATE
audit_tbl SET f2 = 'Row A' WHERE f1 = 1;",,,,,,,,,,"psql","client backend",
"UPDATE","", "update"

2020-05-25 16:25:37.509 IST,"admin","edb",71920,"[local]",5ecba3fe.118f0,2,
"idle",2020-05-25 16:24:54 IST,4/33,0,AUDIT,00000,"statement: SELECT * FROM
edb_tbl;",,,,,,,,,,"psql","client backend","SELECT","", "select"

2020-05-25 16:25:46.131 IST,"admin","edb",71920,"[local]",5ecba3fe.118f0,3,
"idle",2020-05-25 16:24:54 IST,4/34,0,AUDIT,00000,"statement: TRUNCATE edb_
tbl;",,,,,,,,,,"psql","client backend","TRUNCATE TABLE","", "truncate"

```

(次のページに続く)

(前のページからの続き)

```
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,00000,"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,"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 監査ログファイル

監査ログファイルは、`edb_audit` 構成パラメーターの設定に応じて、CSV 形式または XML 形式で生成できます。

監査ログの情報は、PostgreSQL コアドキュメントの「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グに基づいています。

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

次の表に、XML 監査ログ形式で表示される順序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.** 以前に参照した PostgreSQL ドキュメントのサンプルテーブル定義に示されているフィールドの名前。
- **XML Element/Attribute.** XML 形式の場合、XML 要素の名前とその属性（使用する場合）、値を参照します XML Element/Attribute. n/a は、このフィールドに XML 表現がないことを示します。
- **Data Type.** 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。
- **Description.** フィールドの説明。特定のフィールドについては、監査によってサポートされていないため、監査ログには出力が生成されません。これらのフィールドは「サポートされていません」で示されます。

ロギングの値を持たないフィールドは、CSV 形式で連続したコンマ (,) として表示されます。

野原	XML 要素/属性	データ型	説明
<code>user_name</code>	イベント/ユーザー	テキスト	ステートメントを実行したデータベースユーザー。
<code>database_name</code>	イベント/データベース	テキスト	ステートメントが実行されたデータベース。
<code>process_id</code>	event/process_id	整数	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。
<code>connection_from</code>	event/remote_host	テキスト	ステートメントが実行されたホストとポートの場所。
<code>session_id</code>	event/session_id	テキスト	ステートメントが実行されたセッション ID。

次のページに続く

表 2 – 前のページからの続き

野原	XML 要素/属性	データ型	説明
session_line_num	event/session_line_num	ビッグ	セッション内のステートメントの順序。
process_status	event/process_status	テキスト	処理ステータス。
session_start_time	event/session_start_time	タイムゾーン付きのタイムスタンプ	セッションが開始された日時。
log_time	イベント/ログ時間	タイムゾーン付きのタイムスタンプ	ステートメントのログ日付/時刻。
virtual_transaction_id	event/virtual_transaction_id	テキスト	ステートメントの仮想トランザクション ID。
transaction_id	event/transaction_id	ビッグ	ステートメントの通常のトランザクション ID。
タイプ	イベント/タイプ	テキスト	監査 event_type を決定して、ログ内のメッセージを識別します。
sql_state_code	event/sql_state_code	テキスト	ステートメントに対して返された SQL 状態コード。XML ログ形式の値が 00000 の場合、sql_state_code は記録されません。
command_tag	event/command_tag	テキスト	ステートメントの SQL コマンド。
audit_tag	event/audit_tag	テキスト	構成ファイルの audit_tag パラメーターで指定された値。
application_name	event/application_name	テキスト	ステートメントが実行されたアプリケーションの名前。(たとえば、psql.bin)。

次のページに続く

表 2 – 前のページからの続き

野原	XML 要素/属性	データ型	説明
backend_type	event/backend_type	テキスト	backend_type は pg_stat_activity.backend_type が示すものに対応し、csv ログに列として追加されます。
error_severity	error_severity	テキスト	ステートメントの重大度。値は、監査されたステートメントの場合は AUDIT、結果のエラーメッセージの場合は ERROR です。
伝言	伝言	テキスト	実行しようとした SQL ステートメント。
細部	細部	テキスト	エラーメッセージの詳細。
ヒント	ヒント	テキスト	エラーのヒント。
internal_query	internal_query	テキスト	エラーにつながった内部クエリ（ある場合）。
internal_query_pos	internal_query_pos	整数	エラー位置の文字カウント。
文脈	文脈	テキスト	エラーコンテキスト。
問い合わせ	問い合わせ	テキスト	エラーにつながったユーザークエリ。For（エラーのみ）
query_pos	query_pos	整数	エラー位置の文字カウント。（エラーのみ）

次のページに続く

表 2 – 前のページからの続き

野原	XML 要素/属性	データ型	説明
場所	場所	テキスト	ソースコード内のエラーの場所。 log_error_verbosity が verbose に設定されている場合、場所フィールドに入力されます。

次の例は、CSV および XML 形式で生成されます。

postgresql.conf ファイルのデフォルト以外の監査設定は次のとおりです。

```
logging_collector = 'on'
edb_audit = 'csv'
edb_audit_connect = 'all'
edb_audit_disconnect = 'all'
edb_audit_statement = 'ddl, dml, select, error'
edb_audit_tag = 'edbaudit'
```

XML 形式を生成すると、edb_audit parameter が 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 (13.0.0)
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;
```

(次のページに続く)

(前のページからの続き)

```

UPDATE 1
edb=# SELECT * FROM dept;
 deptno |   dtype   | loc
-----+-----+-----
      10 | ACCOUNTING | BOSTON
(1 row)

edb=# \q

```

CSV Audit Log File

監査ログファイルの CSV 形式は次のとおりです。

各監査ログエントリは複数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。

```

2020-05-22 16:53:37.817 IST,"enterprisedb","edb",55279,"[local]",5ec7b639.
d7ef,1,"authentication",2020-05-22 16:53:37 IST,4/21,0,AUDIT,00000,
"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,00000,"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,00000,"statement: CREATE
TABLE dept (
      deptno          NUMBER(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
      dtype           VARCHAR2(14) CONSTRAINT dept_dtype_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,00000,"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,00000,"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"

```

(次のページに続く)

(前のページからの続き)

```

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,00000,"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,00000,"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,00000,"disconnection: session
time: 0:01:28.732 user=enterprisedb database=edb host=[local],,,,,,,,,"psql",
"client backend","", "edbaudit","disconnect"

```

XML Audit Log File

以下は、監査ログファイルの XML 形式です。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

```

(次のページに続く)

(前のページからの続き)

```

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>
<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>

```

(次のページに続く)

(前のページからの続き)

```

        <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_
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 エラーコードを使用して監査ログをフィルタリングする

AdvancedServer 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 AdvancedServer 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、まず `postgresql.conf` ファイルを変更して拡張機能を有効にし、`shared_preload_libraries` パラメーター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.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。複数のエラーコードをコンマ区切りのリストで指定します。

たとえば、`edb_filter_log` が有効で、`edb_filter_log.errcode` が `'23505,23502,22012'` に設定されている場合、次の SQLSTATE エラーのいずれかを返すログエントリ:

23505 (一意の制約に違反するため)

23502 (非ヌル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。

AdvancedServer 監査ログフィルタリングでサポートされるエラーコードの完全なリストについては、次のコアダキュメントを参照してください。

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

3.5.6 コマンドタグを使用して監査ログをフィルタリングする

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 形式の例です。この例は、確認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、例では値 CREATE ROLE、ALTER ROLE、および DROP ROLE 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">
  <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"
```

(次のページに続く)

(前のページからの続き)

```

    application_name="psql" backend_type="client backend">
      <error_severity>AUDIT</error_severity>
      <message>statement: DROP ROLE newuser;</message>
</event>

```

以下は、CSV 形式の同じ例です。コマンドタグは、各エントリの最後から 2 番目の列です。リストでは、空 ("/") として表示される列は、edb_audit_tag パラメーターの値になります（指定されている場合）。

各監査ログエントリは複数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 19:09:32.105 IST,"enterprisedb","edb",77212,"[local]",5ecbca7b.
12d9c,2,"idle",2020-05-25 19:09:07 IST,4/30,0,AUDIT,00000,"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,00000,"statement: ALTER
ROLE newuser WITH SUPERUSER;","",,,,,,"psql","client backend","ALTER ROLE","",
"alter"

2020-05-25 19:10:15.959 IST,"enterprisedb","edb",77212,"[local]",5ecbca7b.
12d9c,5,"idle",2020-05-25 19:09:07 IST,4/33,0,AUDIT,00000,"statement: DROP
ROLE newuser;","",,,,,,"psql","client backend","DROP ROLE","", "drop"

```

3.5.7 監査ログからのパスワードの編集

edb_filter_log.redact_password_commands 拡張子を使用して、ログファイルから保存されたパスワードを編集するようにサーバーに指示できます。モジュールは次の構文のみを認識することに注意してください。

```

{CREATE|ALTER} {USER|ROLE|GROUP} identifier { [WITH] [ENCRYPTED] PASSWORD
'nonempty_string_literal' | IDENTIFIED BY { 'nonempty_string_literal' |
bareword } } [ REPLACE { 'nonempty_string_literal' | bareword } ]

```

そのようなステートメントが log_statement で記録されると、サーバーは古いパスワードと新しいパスワードを「x」に編集します。たとえば、次のコマンド:

```
ALTER USER carol PASSWORD '1safepwd' REPLACE 'old_pwd';
```

次のようにログファイルに追加されます。

```
statement: ALTER USER carol PASSWORD 'x' REPLACE 'x';
```

リダクトされたパスワードを含むステートメントがログに記録されると、サーバーはステートメントテキストをリダクトします。ステートメントが他のメッセージのコンテキストとしてログに記録されると、サーバーはそのステートメントをコンテキストから省略します。

パスワードのリダクションを有効にするには、最初に `postgresql.conf` ファイルを変更して拡張機能を有効にし、`shared_preload_libraries` パラメーター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、変更を有効にするにはサーバーを再起動する必要があります。

3.6 Unicode 照合アルゴリズム

*UnicodeCollationAlgorithm** (UCA) は、Unicode データを照合および比較するカスタマイズ可能な方法を定義する仕様 (**UnicodeTechnicalReport # 10*) です。照合*は、“*SELECT ... ORDER BY*” 句と同様にデータをソートする方法を意味します。*比較 は、より小さい、より大きい、または等しい演算子で範囲を使用する検索に関連します。

カスタマイズ性は、次のようなさまざまな理由で重要な要素です。

- Unicode は膨大な数の言語をサポートしています。複数の言語に共通する可能性のある文字は、言語に応じて異なる順序で照合されることが予想されます。
- アクセントやウムラウトなど、特定の言語で文字とともに表示される文字は、言語によっては予想される照合に影響を与えます。
- 一部の言語では、複数の連続した文字の組み合わせは、照合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に関しては、特定の好みがある場合があります。たとえば、小文字の文字が同じ文字の大文字の前に照合されるか、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関して、好みがあるかもしれません。

Unicode でサポートされている膨大な数の言語でこれらのすべてのバリエーションを考えると、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方法が必要です。これが *UnicodeCollationAlgorithm* が定義するものです。

注釈: さらに、ICU 照合 (Unicode 照合アルゴリズムの実装) を使用するもう 1 つの利点は、パフォーマンスです。B ツリーインデックスの作成などの並べ替えタスクは、ICU 以外の照合でかかる時間の半分以下で完了できます。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、*UnifiedCollationAlgorithm* 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 基本的な Unicode 照合アルゴリズムの概念

UnicodeCollationAlgorithm の公式情報は、*UnicodeTechnicalReport # 10* で指定されています。これは、Unicode コンソーシアムの Web サイトにあります。

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

ICU-ユニコード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。照合の概念の説明は、次の Web サイトにあります。

<http://userguide.icu-project.org/collation/concepts>

UnicodeCollationAlgorithm 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順序を決定できる場合、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順序を決定できない場合、セカンダリレベルのレベル 2 が適用されます。二次レベルに基づいて順序を決定できる場合はアルゴリズムが実行され、そ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順序を決定する最終的なタイブレイクレベルがあります。

- **Level 1 – Primary Level for Base Characters.** 文字や数字などの基本文字の順序によって、A < B などの違いが決まります。
- **Level 2 – Secondary Level for Accents.** 主要なレベルの違いがない場合、アクセントやその他の文字の有無によって a < 叩 などの順序が決まります。
- **Level 3 – Tertiary Level for Case.** プライマリレベルまたはセカンダリレベルの違いがない場合、大文字と小文字の違いによって a < A などの順序が決まります。
- **Level 4 – Quaternary Level for Punctuation.** 1 次、2 次、または 3 次のレベルの違いがない場合、空白文字、制御文字、および句読点の有無によって -A < A などの順序が決まります。
- **Level 5 – Identical Level for Tie-Breaking.** プライマリ、セカンダリ、ターシャリ、またはクォータナリレベルの差がない場合、コードポイント値などのその他の差が順序を決定します。

3.6.2 Unicode の国際コンポーネント

UnicodeCollationAlgorithm は、*InternationalComponentsforUnicode* (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。

AdvancedServer を使用して ICU コンポーネントを呼び出して照合を作成する場合、結果は*ICU 照合*と呼ばれます。

3.6.2.1 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 短縮名が提供されます。

ICU ショートフォーム*は、照合のプロパティである*照合属性を指定する方法です。Collation Attributes は、照合属性に関する追加情報を提供します。

ロケールには、事前定義された ICU の短い形式があります。ロケールの ICU 短縮形には、特定のロケールで通常

使用される照合属性設定が組み込まれています。これにより、そのロケールの照合属性のリスト全体を指定する必要がなくなるため、照合作成プロセスが簡素化されます。

システムカタログ `pg_catalog.pg_icu_collate_names` には、ロケールの ICU ショートフォームの名前のリストが含まれています。ICU の短い形式の名前は、`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、照合属性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。

3.6.2.2 照合属性

ICU 照合を作成するときは、照合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、*collationattributes* を使用して照合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シーケンスを決定するための文字の比較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、これらの照合属性は非常に複雑です。

照合属性の完全で正確な意味と使用法については、ICU–International Components for Unicode Web サイトのセクション「Collator Naming Scheme」を参照してください。

<http://userguide.icu-project.org/collation/concepts>

以下は、照合属性の簡単な概要と、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、括弧内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。****X****は、属性をオンにすることを意味します **X**** は、属性をデフォルト値に設定することを意味します。

- ****A – Alternate (N, S, D).** **は、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、可変文字の違いは文字の違いと同じ重要度で扱われます。シフト (S) に設定すると、可変文字の違いはそれほど重要ではありません（つまり、基本文字を比較する場合、可変文字は無視されます）。
- **C – Case First (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 – Case Level (X, O, D).** 強度属性と組み合わせて設定します。大文字と小文字ではなくアクセントを無視する場合に、ケースレベル属性が使用されます。
- **F – French Collation (X, O, D).** をオンに設定すると、二次差異（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 – Hiragana Quaternary (X, O, D).** JISX4061 の日本語文字列の照合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を導入します。
- **N – Normalization Checking (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、そのような文字をソート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 – Strength (1, 2, 3, 4, I, D).** 比較に使用される最大照合レベル。文字列を照合または比較するときに、アクセントまたは大文字小文字を考慮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 – Variable Top (hexadecimal digits).** Alternate 属性が無視できない (N) に設定されていない場合にのみ適用されます。16 進数字は、無視できると見なされる最高の文字シーケンスを指定します。たとえば、空白は無視できるが、表示される可変文字は無視できない場合、変数トップが 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 16 進数の 0020 です。バックスペース、タブ、

ラインフィード、キャリッジリターンなどの他の非表示変数文字の値は 0020 未満です。表示されるすべての句読点の値は 0020 を超えます。

照合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属性文字で構成されるテキスト文字列で表されます。次の例に示すように、各属性/値のペアは、アンダースコア文字で次のペアに結合されます。

```
AN_CX_EX_FX_HX_NO_S3
```

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、ロケールのこれら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、LROOT という名前の ICU ショートフォームが他の多くの照合属性/値ペアで変更された例です。

```
AN_CX_EX_LROOT_NO_S3
```

上記の例では、代替属性 (A) は無視できない (N に設定されています)。CaseFirst 属性 (C) は (X) に設定されています。ケースレベル属性 (E) はオフ (X) に設定されています。正規化属性 (N) は (O) に設定されています。強度属性 (S) は、3 次レベル 3 に設定されます。「10」は、これらの他の属性が変更を適用している ICU の短い形式です。

3.6.3 照合の使用

新しく定義された ICU 照合は、CREATE TABLE コマンドの列指定など、SQL コマンドで COLLATION / "collation_name/" 句を使用できる場所、または SELECT コマンド。

以下は、米国の英語 (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、小文字の文字を大文字 (CL) の前に強制的にソートします。

照合 icu_collate_uppercase 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、ソート中に可変文字 (空白および句読点) が無視されます (AS)。

照合 icu_collate_ignore_white_sp により、ソート時に空白およびその他の不可視の可変文字が無視されますが、可視の可変文字 (句読点) は無視されません (AS、T0020)。

```
CREATE COLLATION icu_collate_lowercase (
    LOCALE = 'en_US.UTF8',
    ICU_SHORT_FORM = 'AN_CL_EX_NX_LROOT'
);

CREATE COLLATION icu_collate_uppercase (
    LOCALE = 'en_US.UTF8',
    ICU_SHORT_FORM = 'AN_CU_EX_NX_LROOT'
```

(次のページに続く)

(前のページからの続き)

```
);

CREATE COLLATION icu_collate_ignore_punct (
    LOCALE = 'en_US.UTF8',
    ICU_SHORT_FORM = 'AS_CX_EX_NX_LROOT_L3'
);

CREATE COLLATION icu_collate_ignore_white_sp (
    LOCALE = 'en_US.UTF8',
    ICU_SHORT_FORM = 'AS_CX_EX_NX_LROOT_L3_T0020'
);
```

注釈: 照合を作成するとき、LROOT 照合を変更するための属性が指定されている場合、ICU は通知および警告メッセージを生成する場合があります。

次の psql コマンドは、照合をリストします。

```
edb=# \dO
                                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)
);

INSERT INTO collate_tbl VALUES (1, 'A');
INSERT INTO collate_tbl VALUES (2, 'B');
INSERT INTO collate_tbl VALUES (3, 'C');
INSERT INTO collate_tbl VALUES (4, 'a');
INSERT INTO collate_tbl VALUES (5, 'b');
```

(次のページに続く)

(前のページからの続き)

```

INSERT INTO collate_tbl VALUES (6, 'c');
INSERT INTO collate_tbl VALUES (7, '1');
INSERT INTO collate_tbl VALUES (8, '2');
INSERT INTO collate_tbl VALUES (9, '.B');
INSERT INTO collate_tbl VALUES (10, '-B');
INSERT INTO collate_tbl VALUES (11, ' B');

```

次のクエリは、デフォルトの照合を使用して c2 列でソートします。id 列の値が 9、10、および 11 の可変文字（空白および句読点）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)

```

次のクエリは、照合 icu_collate_lowercase を使用して c2 列でソートします。これにより、同じベース文字の大文字の前に小文字の文字が強制的にソートされます。また、AN 属性は、ベース文字を比較するときに変数文字を同じレベルでソート順に含めることを強制するため、9、10、および 11 の id 値を持つ行が先頭に表示されることに注意してくださいすべての文字と数字の前のソートリストの。

```

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

```

次のクエリは、照合 `icu_collate_uppercase` を使用して列 `c2` でソートします。これにより、大文字の文字が同じベース文字の小文字の前に強制的にソートされます。

```
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)
```

次のクエリは、照合 `icu_collate_ignore_punct` を使用して `c2` 列でソートします。これにより、可変文字が無視されるため、ID 値が 9、10、11 の行は B の文字でソートされます無視された変数文字の直後の文字。

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

次のクエリは、照合 `icu_collate_ignore_white_sp` を使用して `c2` 列でソートします。照合の `AS` および `T0020` 属性により、16 進数 0020 以下のコードポイントを持つ可変文字は無視されますが、16 進数 0020 より大きいコードポイントを持つ可変文字はソートに含まれます。

スペース値（16 進数の 0020）で始まる 11 の id 値を持つ行は、B の文字でソートされます。16 進数の 0020 よりも大きい目に見える句読点で始まる 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同じレベルのソート順序に含まれているため、ソートリストの先頭に表示されます基本文字を比較するとき。

```
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 リソースマネージャー

EDBResourceManager は、AdvancedServer 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 AdvancedServer 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、EDBResourceManager の使用に関する重要なポイントです。

- EDBResourceManager 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、AdvancedServer インスタンスのすべてのデータベースで利用できる名前付きのグローバルグループであり、さまざまなリソース使用制限を定義できます。特定のリソースグループのメンバーとして割り当てられた AdvancedServer プロセスは、EDBResourceManager 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は、リソースグループを作成、変更、および削除するために使用されます。これらのコマンドは、スーパーユーザー特権を持つデータベースユーザーのみが使用できます。
- リソースグループに属するすべてのプロセスの望ましい総消費レベルは、リソースタイプパラメータで定義されます。EDBResourceManager で現在サポートされているさまざまなタイプのシステムリソースには、さまざまなリソースタイプパラメータがあります。
- 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。
- EDBResourceManager 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ーがある場合、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ー設定よりも大幅に低くなる可能性があります。これは、EDBResourceManager 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。
- 使用可能なリソースグループの定義とそのリソースタイプ設定は、共有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、リソースグループは、特定の AdvancedServer インスタンス内のすべてのデータベースで利用できます。

- `edb_max_resource_groups` 構成パラメータは、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。このパラメーターの変更は、データベースサーバーの再起動時に有効になります。
- `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` ファイルのパラメーターを設定することにより、データベースサーバーインスタンス全体にデフォルトのリソースグループを割り当てることができます。
- データベースサーバー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`、`--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>;
```

Description

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。次に、ALTER RESOURCE GROUP コマンドを使用して、グループにリソース制限を定義できます。リソースグループは、AdvancedServer 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、スーパーユーザー権限が必要です。

Parameters

group_name

リソースグループの名前。

Example

次の例では、resgrp_a、resgrp_b、resgrp_c という名前の 3 つのリソースグループが作成されます。

```
edb=# CREATE RESOURCE GROUP resgrp_a;
CREATE RESOURCE GROUP
edb=# CREATE RESOURCE GROUP resgrp_b;
CREATE RESOURCE GROUP
edb=# CREATE RESOURCE GROUP resgrp_c;
CREATE RESOURCE GROUP
```

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

```
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 番目の形式は、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 番目の形式は、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>;
```

Description

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 番目の形式は、指定されたリテラル値をリソースタイプに割り当てるか、DEFAULT が指定されたときにリソースタイプをリセットします。リソースタイプを DEFAULT にリセットまたは設定すると、リソースグループにはそのリソースタイプの制限が定義されていないことを意味します。

RESET resource_type 句を含む 3 番目の形式は、前述のようにグループのリソースタイプをリセットします。

ALTER RESOURCE GROUP コマンドを使用するには、スーパーユーザー権限が必要です。

Parameters

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 を指定すると、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。

Example

以下は ALTER RESOURCE GROUP コマンドの例です。

```

edb=# ALTER RESOURCE GROUP resgrp_a RENAME TO newgrp;
ALTER RESOURCE GROUP
edb=# ALTER RESOURCE GROUP resgrp_b SET cpu_rate_limit = .5;
ALTER RESOURCE GROUP
edb=# ALTER RESOURCE GROUP resgrp_b SET dirty_rate_limit = 6144;
ALTER RESOURCE GROUP
edb=# ALTER RESOURCE GROUP resgrp_c RESET cpu_rate_limit;
ALTER RESOURCE GROUP

```

次のクエリは、edb_resource_group カタログのエントリに対する ALTER RESOURCE GROUP コマンドの結果を示しています。

```

edb=# SELECT * FROM edb_resource_group;
  rgrpname  | rgrpcpuratelimit | rgrpdirtyratelimit
-----+-----+-----
newgrp      | 0                | 0
resgrp_b    | 0.5              | 6144
resgrp_c    | 0                | 0
(3 rows)

```

4.1.3 ドロップグループ

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>;
```

Description

DROP RESOURCE GROUP コマンドは、指定された名前のリソースグループを削除します。

DROP RESOURCE GROUP コマンドを使用するには、スーパーユーザー権限が必要です。

Parameters

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。

Example

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

SET edb_resource_group TO group_name コマンドを使用して、以下に示すように、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ー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 コアドキュメントを参照してください。

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

デフォルトのリソースグループは、ALTER DATABASE ... SET コマンド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 コアドキュメントを参照してください。

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

以下に示すように、postgresql.conf ファイルで edb_resource_group 構成パラメータを設定することにより、データベースサーバーインスタンス全体にデフォルトのリソースグループを割り当てることができます。

```
# - EDB Resource Manager -
#edb_max_resource_groups = 16           # 0-65536 (change requires restart)
edb_resource_group = 'resgrp_b'
```

postgresql.conf ファイルの edb_resource_group への変更は、データベースサーバーインスタンスで有効になる前に構成ファイルのリロードが必要です。

4.1.5 リソースグループからのプロセスの削除

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 ... RESET` 形式の `ALTER ROLE` コマンドを使用します。

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 ... RESET` 形式の `ALTER DATABASE` コマンドを使用します。

データベースサーバー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` 構成パラメータを `postgresql.conf` ファイルの空の文字列に設定し、構成ファイルを再ロードします。

4.1.6 リソースグループのプロセスの監視

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、ビュー `edb_all_resource_groups` から取得できます。

`edb_all_resource_groups` の列は次のとおりです。

- **group_name.** リソースグループの名前。
- **active_processes.** リソースグループ内のアクティブなプロセスの数。
- **cpu_rate_limit.** リソースグループに割り当てられた CPU レート制限リソースタイプの値。
- **per_process_cpu_rate_limit.** リソースグループ内の個々のアクティブなプロセスに適用される CPU レート制限。
- **dirty_rate_limit.** リソースグループに割り当てられたダーティレート制限リソースタイプの値。
- **per_process_dirty_rate_limit.** リソースグループ内の個々のアクティブプロセスに適用されるダーティレート制限。

注釈: `per_process_cpu_rate_limit` 列と `per_process_dirty_rate_limit` 列は、プロセスが使用する*実際の* リソース消費量を表示しませんが、EDB Resource Manager がリソースグループ内のアクティ

プロセスの数に基づいて個々のプロセスのリソース制限を設定する方法を示します。

以下は、リソースグループ `resgrp_a` にアクティブプロセスが含まれない場合、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つ含まれる場合の `edb_all_resource_groups` を示しています。

```
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、`per_process_cpu_rate_limit` と `per_process_dirty_rate_limit` の値は、おおよそ対応する 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注意してください。

4.2 CPU 使用率の調整

リソースグループの CPU 使用率は、`cpu_rate_limit` リソースタイプパラメータを設定することにより制御されます。

`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実時間以上の 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。

`cpu_rate_limit` パラメーターの有効範囲は 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。

EDBResourceManager 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` パラメーターで指定された制限内に保ちます。グループ内のプロセスは、定義された制限を維持するために、短時間中断され、スリープモードになる場合があります。このような中断がいつどのように発生するかは、EDBResourceManager 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

リソースグループの CPU レート制限を設定するには、`ALTER RESOURCE GROUP` コマンドと `SET cpu_rate_limit` 句を使用します。

次の例では、CPU 使用制限は `resgrp_a` で 50 %、`resgrp_b` で 40 %、`resgrp_c` で 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について、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
```

次のクエリは、カタログの `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
 resgrp_c |                0.3
(3 rows)
```

リソースグループの `cpu_rate_limit` を変更すると、グループに割り当てられた新しいプロセスに影響するだけでなく、グループのメンバーである現在実行中のプロセスはすべて、変更の影響を受けます。つまり、`cpu_rate_limit` を .5 から .3 に変更すると、グループ内の現在実行中のプロセスは、グループの総 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 `SELECT 20000!` によって実行される CPU 集中型の計算 20000 階乗 ($20000 \times 19999 \times 19998$ などの乗算) 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 の `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スナップショットです。

```
$ top
top - 16:37:03 up 4:15, 7 users, load average: 0.49, 0.20, 0.38
Tasks: 202 total, 1 running, 201 sleeping, 0 stopped, 0 zombie
Cpu(s): 42.7%us, 2.3%sy, 0.0%ni, 55.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0
Mem: 1025624k total, 791160k used, 234464k free, 23400k buffers
Swap: 103420k total, 13404k used, 90016k free, 373504k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
28915 enterpri  20   0 195m 5900 4212 S 39.9   0.6   3:36.98 edb-postgres
 1033 root        20   0 171m  77m 2960 S   1.0   7.8   3:43.96 Xorg
 3040 user       20   0 278m  22m 14m S   1.0   2.2   3:41.72 knotify4
    .
    .
    .
```

階乗計算を実行する psql セッションは、COMMAND 列の下に `edb-postgres` が表示される行で示されます。%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、これはリソースグループ `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 使用率が 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一部であったときの 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 つの別々の 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。

Session 1:

```
edb=# SET edb_resource_group TO resgrp_b;
SET
edb=# SHOW edb_resource_group;
   edb_resource_group
-----
```

(次のページに続く)

(前のページからの続き)

```

resgrp_b
(1 row)

edb=# SELECT 20000!;
```

Session 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、edb-postgres という名前の 2 つのプロセスがあり、%CPU の値は 19.9 と 19.6 です。その合計は、リソースグループ resgrp_b に設定された 40 % の CPU 使用率に近い値です。

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 プロセスの合計を表示します。これは、EDBResourceManager 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
37.2
```

(次のページに続く)

(前のページからの続き)

```

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、リソースグループ resgrp_c 内で同じ階乗計算を実行し、cpu_rate_limit を .3（CPU 使用率 30 %）にします。

Session 3:

```

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

edb=# SELECT 20000!;

```

Session 4:

```

edb=# SET edb_resource_group TO resgrp_c;
SET
edb=# SHOW edb_resource_group;
   edb_resource_group
-----
    resgrp_c
(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列の合計は39.5（約40%、resgrp_bの制限）であり、3番目と4番目のedb-postgresプロセスの%CPU列の合計は31.6（約30%、resgrp_cの制限）です。

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使用制限の合計は70%です。次の出力は、4つのプロセスの合計が約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、2つのセッションがresgrp_bのままで3つのセッションが処理されているが、3番目のセッションがリソースグループに属していない場合、topコマンドは次の出力を表示します。

```
$ top
top - 17:24:55 up 5:03, 7 users, load average: 1.00, 0.41, 0.38
Tasks: 199 total, 3 running, 196 sleeping, 0 stopped, 0 zombie
```

(次のページに続く)

(前のページからの続き)

```
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いないため、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 つのセッションの合計が約 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` パラメーターを、グループ内のすべてのプロセスが共有バッファに書き込むか、または共有バッファに「ダーティ」にする必要がある複合レート of 1 秒あたりのキロバイト数に設定します。設定例は、毎秒 3072 キロバイトです。

`dirty_rate_limit` パラメーターの有効範囲は 0 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。

EDBResourceManager は*dirtybufferthrottling*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` パラメーター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。このような中断がいつどのように発生するかは、EDBResourceManager 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` 句を含む 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速度が、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で、共有バッファへの合計書き込み速度は約 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
```

次のクエリは、カタログの `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た新しいプロセスに影響するだけでなく、グループのメンバーである現在実行中のプロセスはすべて、変更の影響を直ちに受けます。つまり、`dirty_rate_limit` を 12288 から 3072 に変更すると、グループで現在実行中のプロセスは、グループの総ダーティレートが 12288 キロバイト/秒ではなく 3072 キロバイト/秒に近くなるように下方に調整されます。

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` 句の結果、`INSERT` コマンドはページごとに最大 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速度を計算し、リソースグループに設定されたダーティレート制限と比較するための情報が提供されます。

`pg_stat_statements` モジュールを使用するには、次の手順を実行します。

Step 1: `postgresql.conf` ファイルで、以下に示すように `shared_preload_libraries` 構成パラメーターに `$libdir/pg_stat_statements` を追加します。

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

Step 2: データベースサーバーを再起動します。

Step 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()` 機能は、各例を明確にするために `pg_stat_statements` ビューをクリアするために使用されます。

これらの例では、前のクエリ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。

4.3.2 例-単一グループ内の単一プロセス

次の一連のコマンドは、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()` 機能を実行すると、`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 の一連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
```

以下に、INSERT コマンドの結果を示します。

```
edb=# SELECT query, rows, total_time, shared_blks_dirtied FROM
pg_stat_statements;
-[ 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) を計算すると、約 $*6072$ キロバイト/秒* になります。

6072 キロバイト/秒の実際のダーティレートは、リソースグループのダーティレート制限である 6144 キロバイト/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
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注意してください。

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) あたりのダーティブロック数は 10003 ブロック/2432.165 ミリ秒で、これは 4.112797 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、 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) を計算すると、約 33692 キロバイト/秒* になります。

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。

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つの異なるテーブルで2つの別々の psql セッションで挿入が同時に実行され、各セッションは dirty_rate_limit が 6144 キロバイト/秒に設定されているリソースグループ resgrp_b に追加されています。

Session 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
```

Session 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
```

注釈: セッション 1 およびセッション 2 の INSERT コマンドは、セッション 2 の SELECT pg_stat_statements_reset () コマンドの実行後に開始されました。

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

まず、セッション 1 の合計時間が 33215.334 ミリ秒、セッション 2 の 30591.551 ミリ秒であったことに注意してください。最初の例に示すように、同じリソースグループで 1 つのセッションのみがアクティブな場合、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* 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、 301.15609 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、 ≈ 2467 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) あたりのダーティブロック数は 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) を計算すると、 ≈ 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。

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 と 2 は、リソースグループ resgrp_b と 6144 キロバイト/秒の dirty_rate_limit を使用して、前の例に示すように繰り返されます。

Session 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
```

Session 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
-----
```

(次のページに続く)

(前のページからの続き)

(1 row)

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

注 釈: 4 つのセッションすべての INSERT コマンドは、セッション 4 の SELECT pg_stat_statements_reset() コマンド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
```

まず、セッション 1 (28407.435) とセッション 2 (31343.458) の時間は、セッション 3 (52727.846) の時間と比較して、dirty_rate_limit が 6144 に設定された同じリソースグループ内にあるため、互いに近いことに注意してください。dirty_rate_limit が 3072 に設定されたリソースグループにあるセッション 4 (56063.697)。

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) あたりのダーティブロック数は 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の数 (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) を計算すると、約*2885 キロバイト/秒* になります。

セッション 2 の実際のダーティレートは、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、これは*0.31914156 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、319.14156 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) を計算すると、約*2614 キロバイト/秒* になります。

セッション 1 (2885 キロバイト/秒) とセッション 2 (2614 キロバイト/秒) 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) あたりのダーティブロック数は 10003 ブロック/52727.846 ミリ秒で、これは*0.18971001 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共有ブロック数 (1 秒=1000 ミリ秒) を求めます。これにより、189.71001 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*1554 キロバイト/秒* 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、これは*0.17842205 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の数 (1 秒=1000 ミリ秒) を求めます。これにより、178.42205 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）を計算すると、約*1462 キロバイト/秒* になります。

セッション 3（1554 キロバイト/秒）とセッション 4（1462 キロバイト/秒）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、EDBResourceManager 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。

4.4 システムカタログ

このセクションでは、EDBResourceManager 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 カタログで利用可能な情報を示します。

コラム	種類	説明
group_name	お名前	リソースグループの名前。
active_processes	整数	リソースグループ内の現在アクティブなプロセスの数。
cpu_rate_limit	float8	リソースグループの最大 CPU レート制限。0 は制限なしを意味します。
per_process_cpu_rate_limit	float8	リソースグループ内の現在アクティブなプロセスごとの最大 CPU レート制限。
dirty_rate_limit	float8	リソースグループの最大ダーティレート制限。0 は制限なしを意味します。
per_process_dirty_rate_limit	float8	リソースグループで現在アクティブなプロセスごとのダーティレートの最大制限。

4.4.2 edb_resource_group

次の表に、edb_resource_group カタログで利用可能な情報を示します。

コラム	種類	説明
rgrpname	お名前	リソースグループの名前。
rgrpcpuratelimit	float8	リソースグループの最大 CPU レート制限。0 は制限なしを意味します。
rgrpdirtyratelimit	float8	リソースグループの最大ダーティレート制限。0 は制限なしを意味します。

第 5 章

libpqC ライブラリ

libpq は、C アプリケーションプログラムの AdvancedServer へのインターフェイスです。libpq は、クライアントプログラムが AdvancedServer 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、および ECPG 用に作成されたものを含む、他のいくつかの EDB アプリケーションインターフェイスの基礎となるエンジンでもあります。したがって、libpq の動作のいくつかの側面は、これらのパッケージのいずれかが使用される場合、ユーザー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、ヘッダーファイル 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 EDBSPL での libpq の使用

EDBSPL 言語は libpq インターフェイスライブラリで使用でき、次のサポートを提供します。

- プロシージャ、関数、パッケージ
- 準備されたステートメント
- REFCURSORS
- 静的カーソル
- structs および typedefs
- 配列
- DML および DDL 操作
- IN / OUT / IN OUT パラメーター

5.2 REFCURSOR のサポート

以前のリリースでは、AdvancedServer 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）関数によって返された REFCURSOR を取得できるようになりました。REFCURSOR は、カーソルの名前を示すヌル終了文字列の形式で返されます。カーソルの名前を取得したら、1 つ以上の FETCH ステートメントを実行して、カーソルを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注意してください。

Returning a Single REFCURSOR

次の例は、タイプ REFCURSOR の値を返す SPL 関数を示しています。

```
CREATE OR REPLACE FUNCTION getEmployees(p_deptno NUMERIC)
RETURN REFCURSOR AS
    result REFCURSOR;
BEGIN
    OPEN result FOR SELECT * FROM emp WHERE deptno = p_deptno;

    RETURN result;
END;
```

この関数は、単一のパラメーター p_deptno を予期し、OPEN ステートメントに示されている SELECT クエリの結果セットを保持する REFCURSOR を返します。OPEN ステートメントはクエリを実行し、結果セットをカーソルに保存します。サーバーはそのカーソルの名前を作成し、その名前を変数（result という名前）に格納します。次に、関数はカーソルの名前を呼び出し元に返します。

libpq を使用して C クライアントからこの関数を呼び出すには、PQexec () と 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);
```

(次のページに続く)

(前のページからの続き)

```

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 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);

```

(次のページに続く)

(前のページからの続き)

```

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);
}

```

コードサンプルには、`getEmployees()` 関数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()` 関数は、C プログラムに結果セットハンドルを返します。結果セットには正確に 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。関数 `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、**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
```

Returning Multiple REFCURSORs

次の例では、2 つの REFCURSORs が返されます。

- 最初の REFCURSOR には、発信者が指定した範囲内の部門で働くすべての従業員を含むカーソル (employees) の名前が含まれます。
- 2 番目の REFCURSOR には、呼び出し元によって指定された範囲内のすべての部門を含むカーソル (departments) の名前が含まれます。

この例では、関数は単一の REFCURSOR を返す代わりに、SETOF REFCURSOR (0 以上 REFCURSORs を意味する) 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 行を期待する必要があることです (最初の行には employees カーソルの名前が含まれ、2 行目には 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
    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 関数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);
```

(次のページに続く)

(前のページからの続き)

```
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);
    PGresult *result;
    int         row;

    sprintf(commandText, "FETCH ALL FROM \"%s\"", cursorName);

    result = PQexec(conn, commandText);
```

(次のページに続く)

(前のページからの続き)

```

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 番目のカーソルを返します。

```

-- employees --
7369,SMITH,CLERK,7902,17-DEC-80 00:00:00,800.00,,20
7499,ALLEN,SALESMAN,7698,20-FEB-81 00:00:00,1600.00,300.00,30
7521,WARD,SALESMAN,7698,22-FEB-81 00:00:00,1250.00,500.00,30
7566,JONES,MANAGER,7839,02-APR-81 00:00:00,2975.00,,20
7654,MARTIN,SALESMAN,7698,28-SEP-81 00:00:00,1250.00,1400.00,30
7698,BLAKE,MANAGER,7839,01-MAY-81 00:00:00,2850.00,,30
7788,SCOTT,ANALYST,7566,19-APR-87 00:00:00,3000.00,,20
7844,TURNER,SALESMAN,7698,08-SEP-81 00:00:00,1500.00,0.00,30
7876,ADAMS,CLERK,7788,23-MAY-87 00:00:00,1100.00,,20

```

(次のページに続く)

(前のページからの続き)

```
7900,JAMES,CLERK,7698,03-DEC-81 00:00:00,950.00,,30
7902,FORD,ANALYST,7566,03-DEC-81 00:00:00,3000.00,,20
-- departments --
20,RESEARCH,DALLAS
30,SALES,CHICAGO
```

5.3 配列バインディング

AdvancedServer の配列バインディング機能を使用すると、1 回の往復でネットワークを介してデータの配列を送信できます。バックエンドはバルクデータを受信すると、そのデータを使用して挿入または更新操作を実行できます。

準備されたステートメントで一括操作を実行します。次の関数を使用してステートメントを準備します。

```
PGresult *PQprepare(PGconn *conn,
                    const char *stmtName,
                    const char *query,
                    int nParams,
                    const Oid *paramTypes);
```

PQprepare() の詳細は、準備済みステートメントのセクションにあります。

次の関数を使用して、一括操作を実行できます。

- PQBulkStart
- PQexecBulk
- PQBulkFinish
- PQexecBulkPrepared

5.3.1 PQBulkStart

PQBulkStart() は、サーバー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、この関数を呼び出す必要があります。PQBulkStart() は、paramFmts で指定された形式でデータを受信するために、stmtName で指定された準備済みステートメントを初期化します。

API Definition

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

5.3.2 PQexecBulk

PQexecBulk() は、PQBulkStart() を使用してバルク操作前に以前に初期化されたステートメントにデータ (paramValues) を提供するために使用されます。

この関数は、PQBulkStart() の後に複数回使用して、複数のデータブロックを送信できます。詳細については例を参照してください。

API Definition

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

5.3.3 PQBulkFinish

この関数は、現在の一括操作を完了します。準備されたステートメントを再準備せずに再度使用できます。

API Definition

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

5.3.4 PQexecBulkPrepared

または、PQexecBulkPrepared() 関数を使用して、単一の関数呼び出しで一括操作を実行できます。PQexecBulkPrepared() は、指定されたパラメーターで準備されたステートメントを実行する要求を送信し、結果を待ちます。この機能は、PQbulkStart()、PQexecBulk()、PQBulkFinish() の機能を組み合わせたものです。この関数を使用する場合、一括操作を初期化または終了する必要はありません。この関数は、一括操作を開始し、データをサーバーに渡し、一括操作を終了します。

stmtName の場所に事前に準備されたステートメントを指定します。繰り返し使用されるコマンドは、実行されるたびにではなく、一度だけ解析および計画されます。

API Definition

```
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、PQBulkStart、PQexecBulk、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);

```

(次のページに続く)

(前のページからの続き)

```

    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

デバッガー

デバッガーにより、開発者と DBA は、グラフィカルで動的な環境を使用して、サーバー側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、および PL/pgSQL 関数とトリガーです。

デバッガーは、*pgAdmin4* および *EDBPostgresEnterpriseManager* と統合されています。Windows ホストに *AdvancedServer* をインストールした場合、*pgAdmin4* は自動的にインストールされます。Windows Start メニューに *pgAdmin4* アイコンがあります。AdvancedServer 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、スーパーユーザー権限を引き受けて、次を入力します。

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

Linux では、xx が AdvancedServer のバージョン番号である edb-asxx-server-pldebugger RPM パッケージもインストールする必要があります。*pgAdmin4* 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。

- **Standalone Debugging.** デバッガーを使用して、テストするプログラムを開始します。プログラムに必要な入力パラメーター値を指定すると、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方法です。
- ****In-Context Debugging.**** 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*グローバルブレイクポイント*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、グローバルブレイ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、その時点でデバッガーが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方法を使用したデバッグの機能について説明します。コンテキスト内デバッグ用のデバッガーを起動する方法については、[コンテキスト内デバッグ用グローバルブレイクポイントの設定](#)で説明されています。

6.1 デバッガーの構成

デバッグを使用する前に、`postgresql.conf` ファイル（AdvancedServer ホームディレクトリの `data` サブディレクトリにあります）を編集し、`shared_preload_libraries` 構成パラメータにリストされているライブラリに `$libdir/plugin_debugger` を追加します。

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

- Linux では、`postgresql.conf` ファイルは `/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

x は AdvancedServer 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ーを再起動する必要があります。

6.2 デバッガーの起動

pgAdmin4 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、pgAdmin4 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関数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。

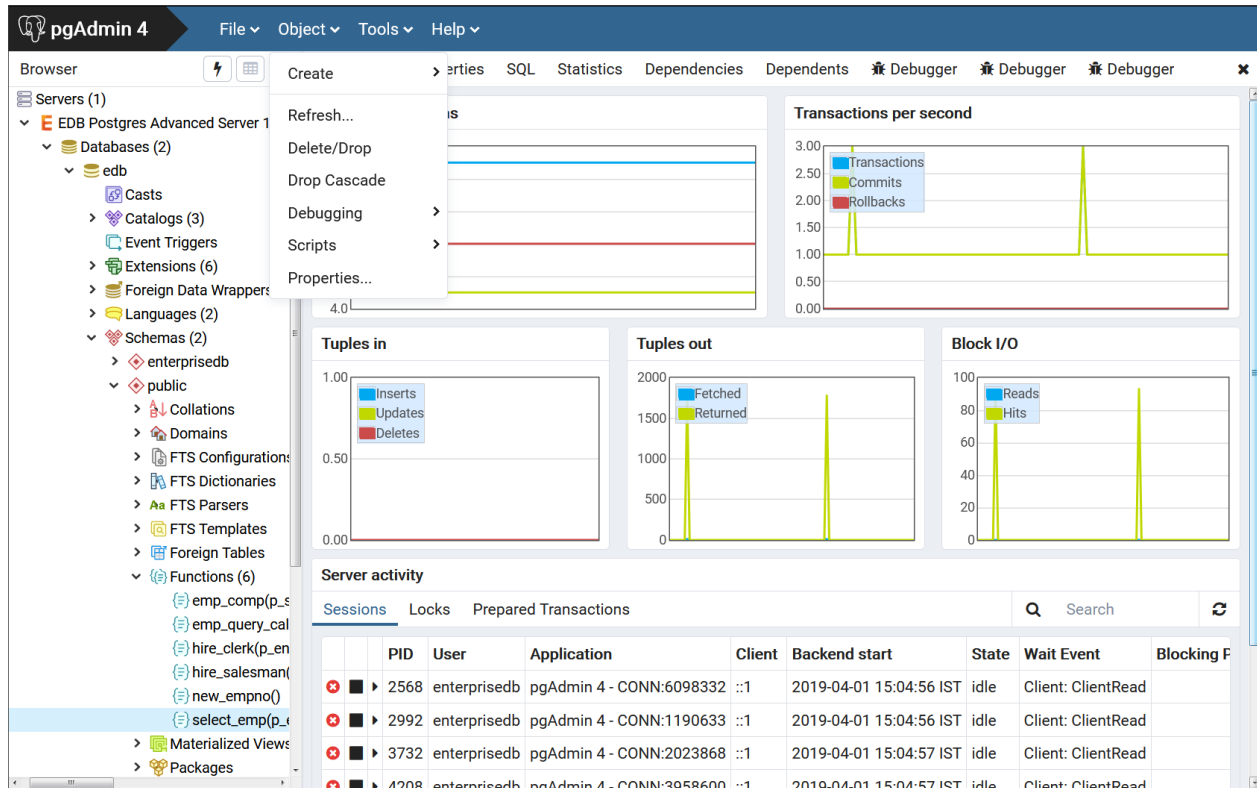


図 1 オブジェクトメニューからデバッガーを起動

pgAdmin4 のストアードプロシージャまたは関数の名前を右クリックして Browser とし、コンテキストメニューから Debugging、Debug を選択することもできます。

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イクポイントの設定については、[コンテキスト内デバッグ用のグローバルブレイクポイントの設定](#) 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の特定のプロシージャまたは関数を強調表示し、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。

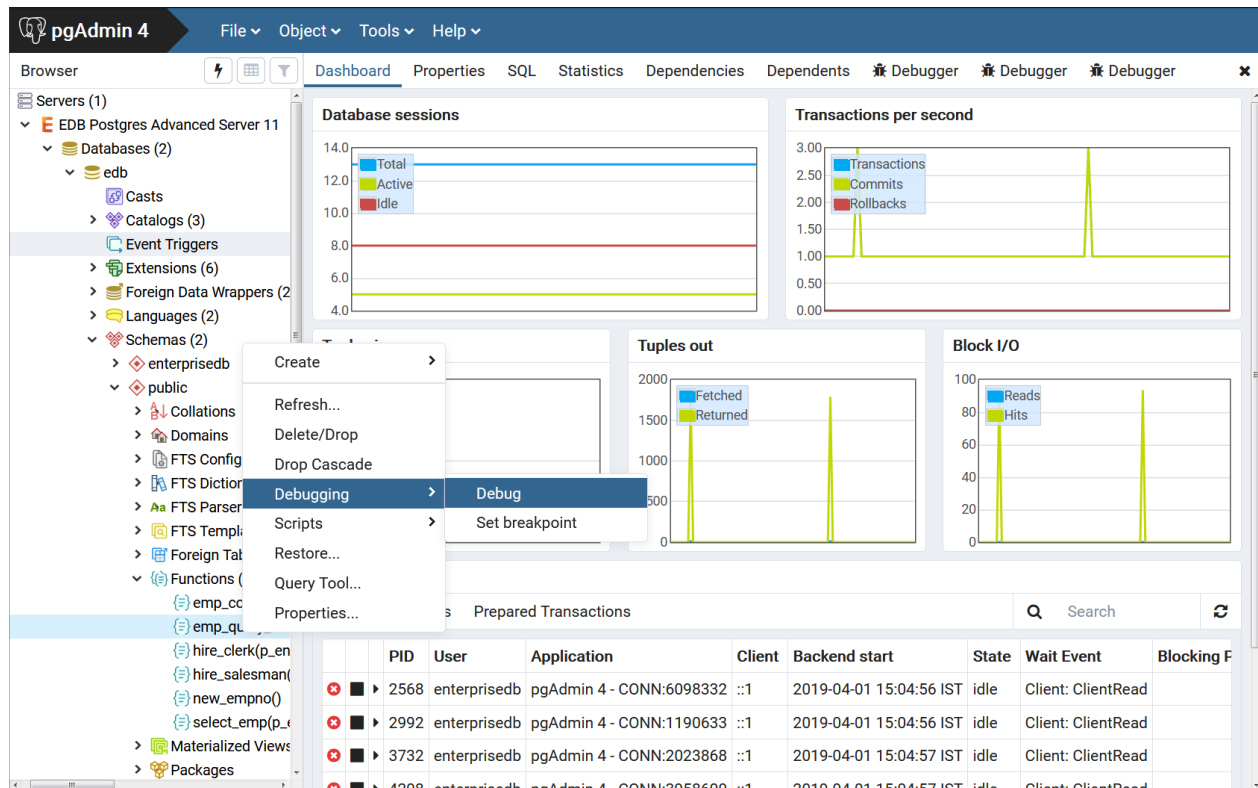


図 2 オブジェクトのコンテキストメニューからデバッガを起動

6.3 デバッガーウィンドウ

パラメーターを必要とするプログラムをスタンドアロンでデバッグしているときに、Debugger ウィンドウを使用してパラメーター値を渡すことができます。デバッガーを起動すると、Debugger ウィンドウが自動的に開き、プログラムで予想される 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 パラメータを宣言しない場合、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>

図 3 デバッガウィンドウ

Debugger ウィンドウのフィールドを使用して、各パラメーターの値を指定します。

- Name フィールドには、正式なパラメータ名が含まれています。
- Type フィールドには、パラメータの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ー値が含まれています。
- Use Default? チェックボックスをオンにして、プログラムが Default Value フィールドの値を使用することを示します。
- 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ています。

Tab キーを押してリスト内のデータ入力用の次のパラメーターを選択するか、Value フィールドをクリックしてデータ入力用のパラメーターを選択します。

初期化セクションを含むパッケージのメンバーであるプロシージャまたは関数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ーに指示し、デバッグ前に初期化セクションのコードをデバッグできるようにします。プロシージャまたは関数。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またはステップスルーすることはできません。

目的のパラメーター値を入力した後、Debug ボタンをクリックしてデバッグプロセスを開始します。Cancel ボタンをクリックして、デバッグを終了します。

注釈: コンテキスト内デバッグ中に 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ー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、新しいパラメーター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、プログラムのソースコードが表示されます。
- 下部の Tabs パネルには、さまざまな情報のタブ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。

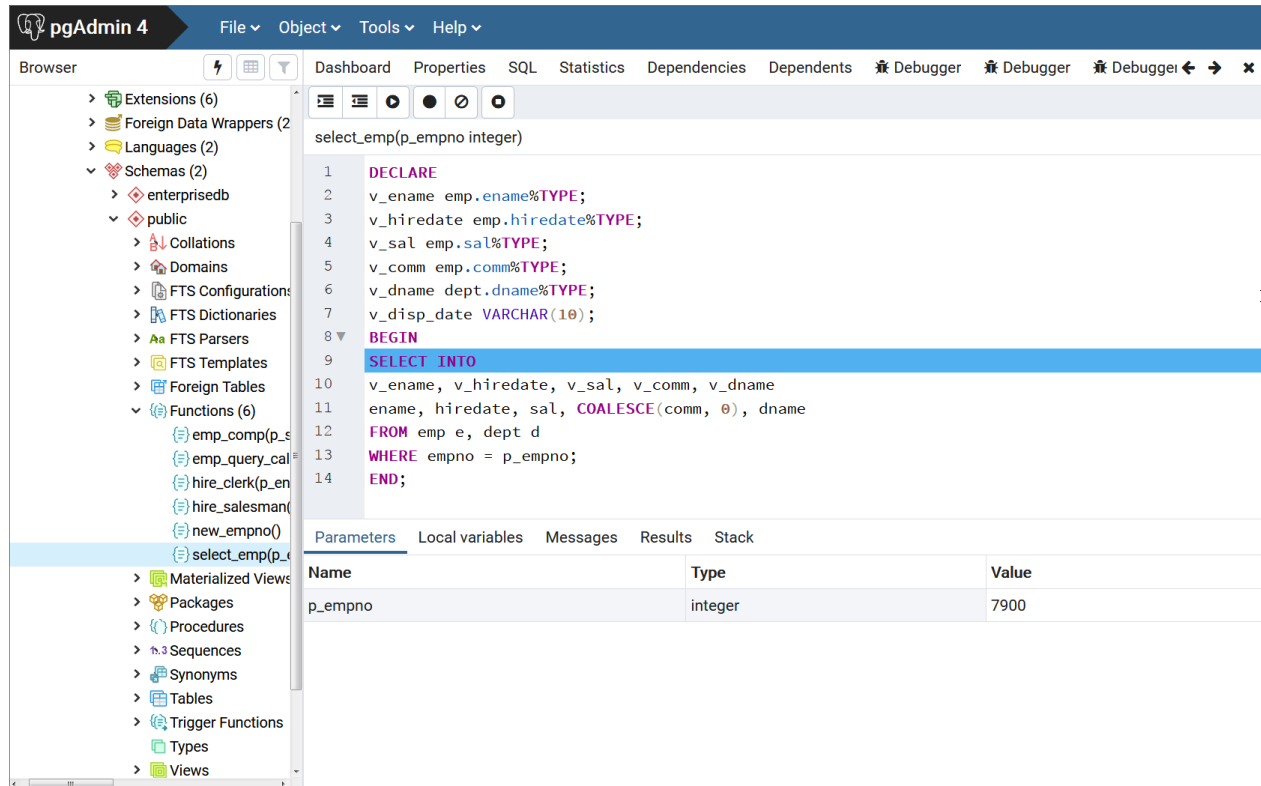


図 4 メインデバッガーウィンドウ

2 つのパネルについては、次のセクションで説明します。

6.4.1 プログラム本体パネル

Program Body パネルには、デバッグ中のプログラムのソースコードが表示されます。

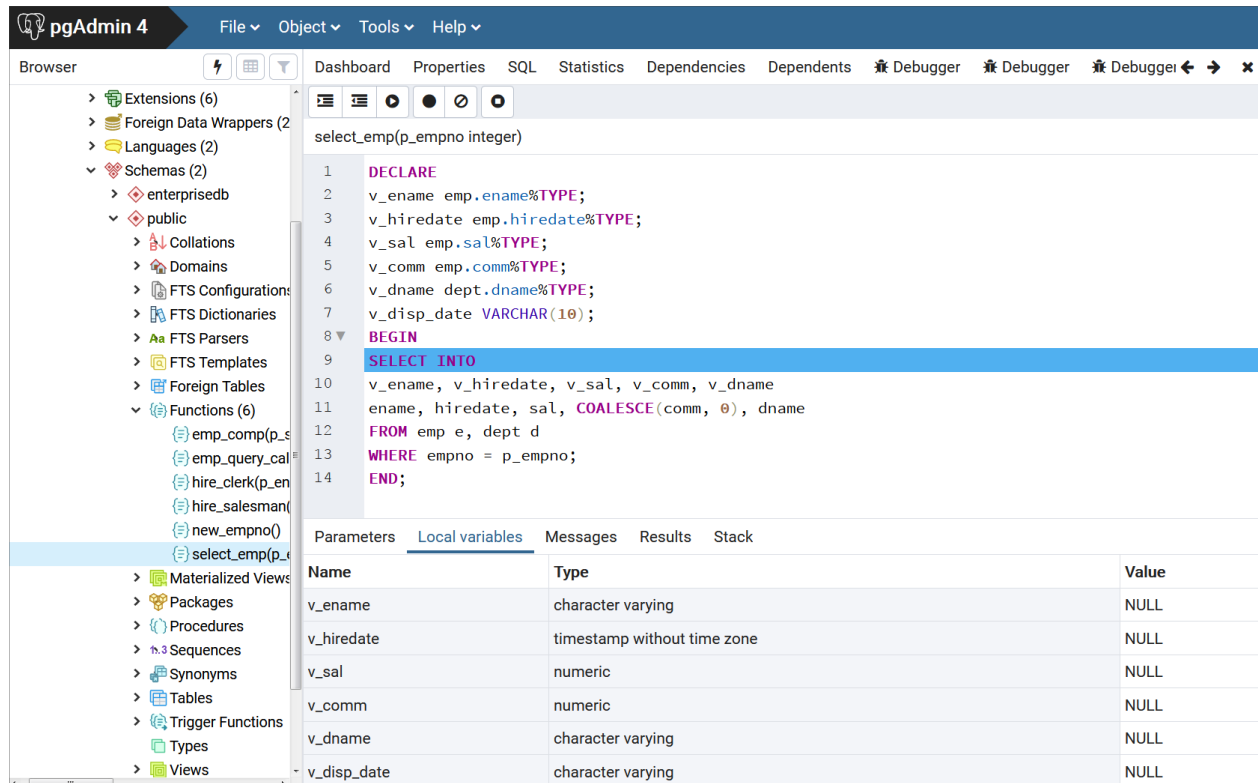


図 5 プログラム本体

この図は、デバッガーが SELECT ステートメントを実行しようとしていることを示しています。プログラム本体の青いインジケータは、実行する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、パラメーター値またはローカル変数を表示または変更したり、RAISE INFO と関数の結果によって生成されたメッセージを表示したりできます。

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、現在のパラメーター値が表示されます。
- Local variables タブには、プログラム内で宣言された変数の値が表示されます。
- Messages タブには、プログラムの実行時に返される結果が表示されます。
- Results タブには、関数の RETURN ステートメントの値など、プログラムの結果（該当する場合）が表示されます。
- Stack tab はコールスタックを表示します。

次の図は、さまざまなタブの結果を示しています。

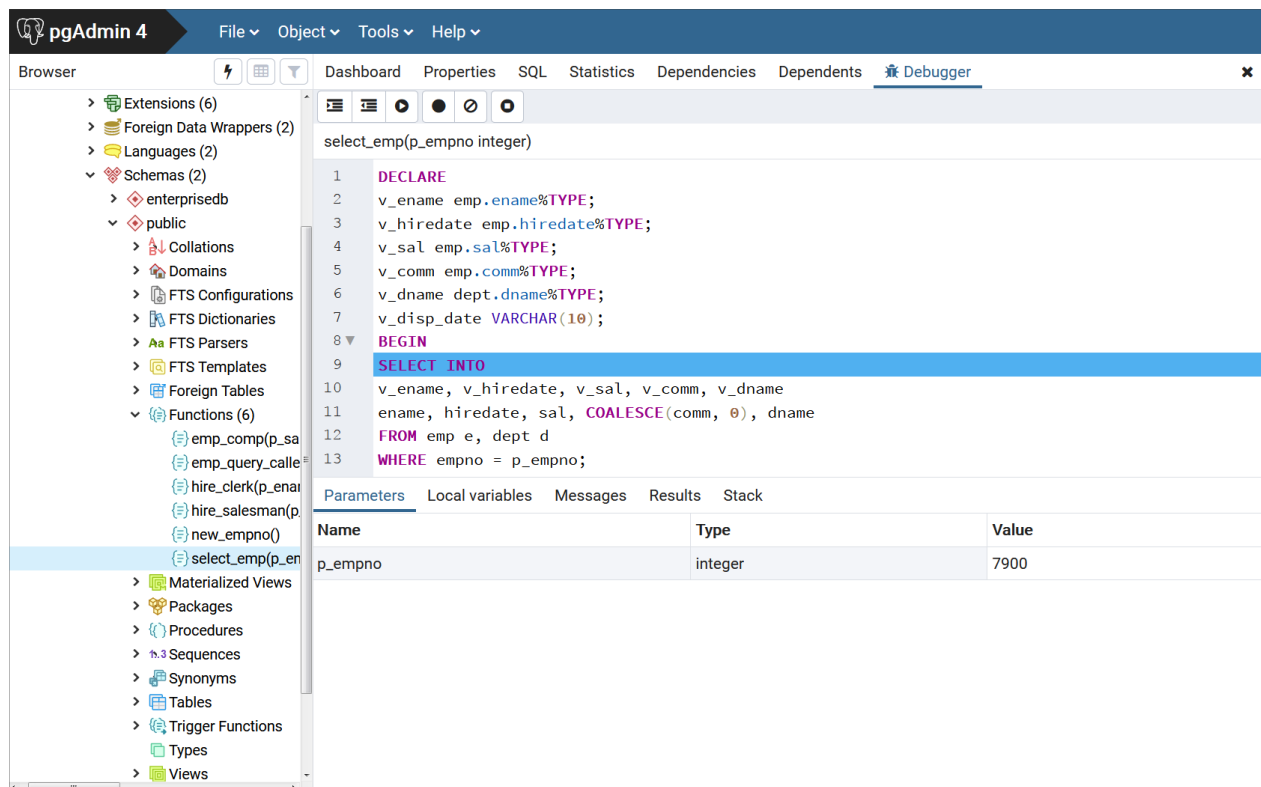


図 6 [パラメータ] タブ

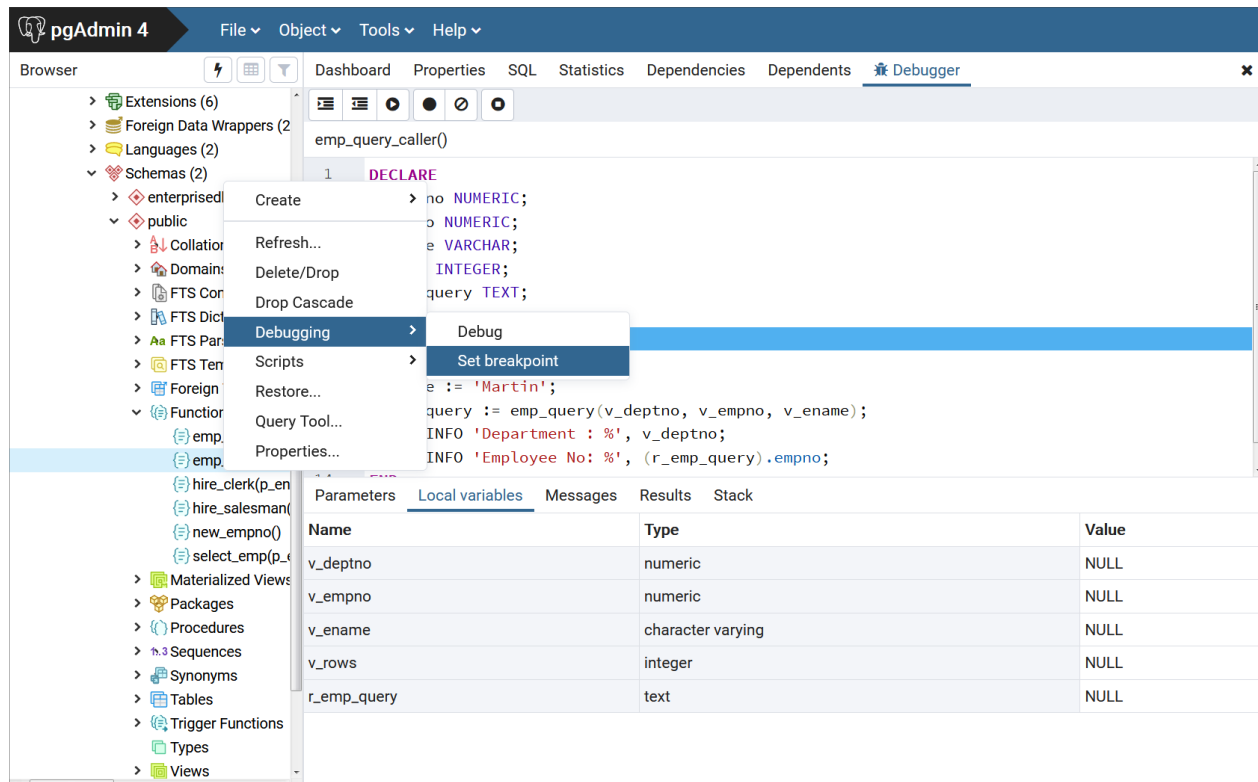


図 7 ローカル変数タブ

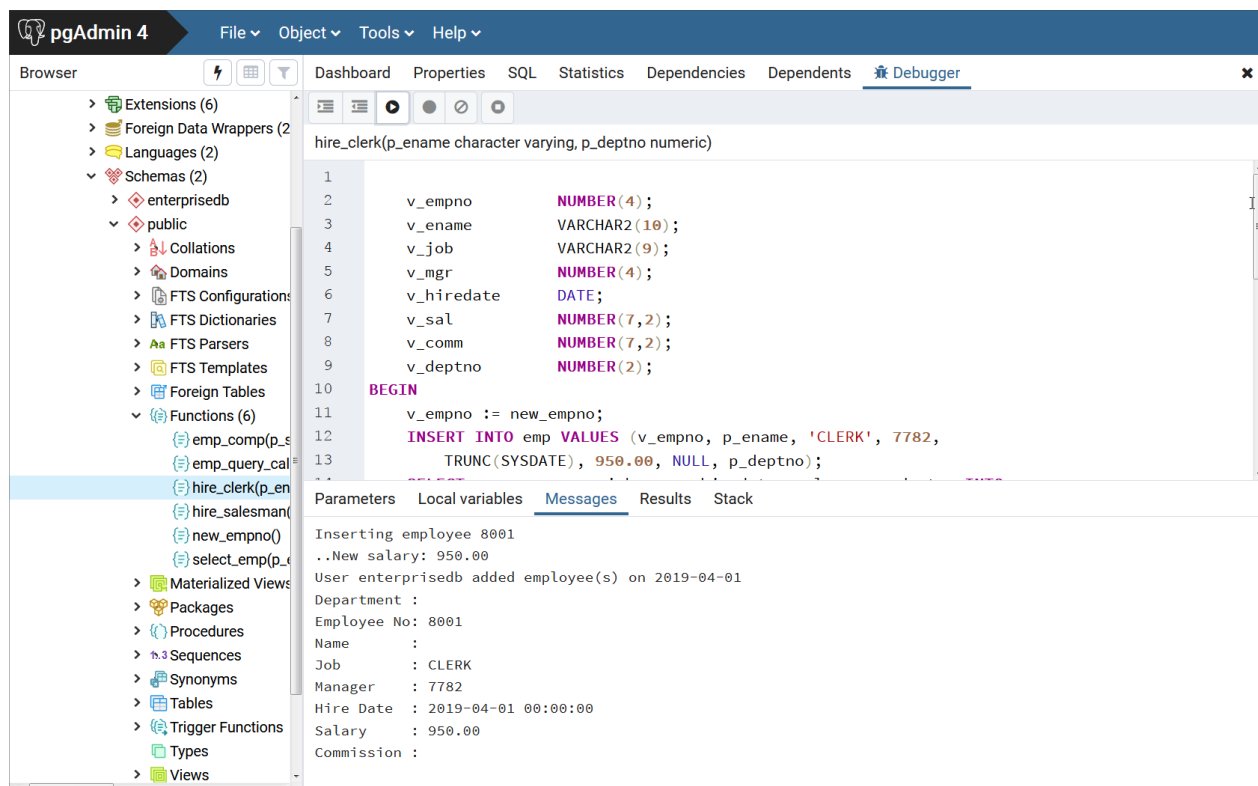


図 8 [メッセージ] タブ

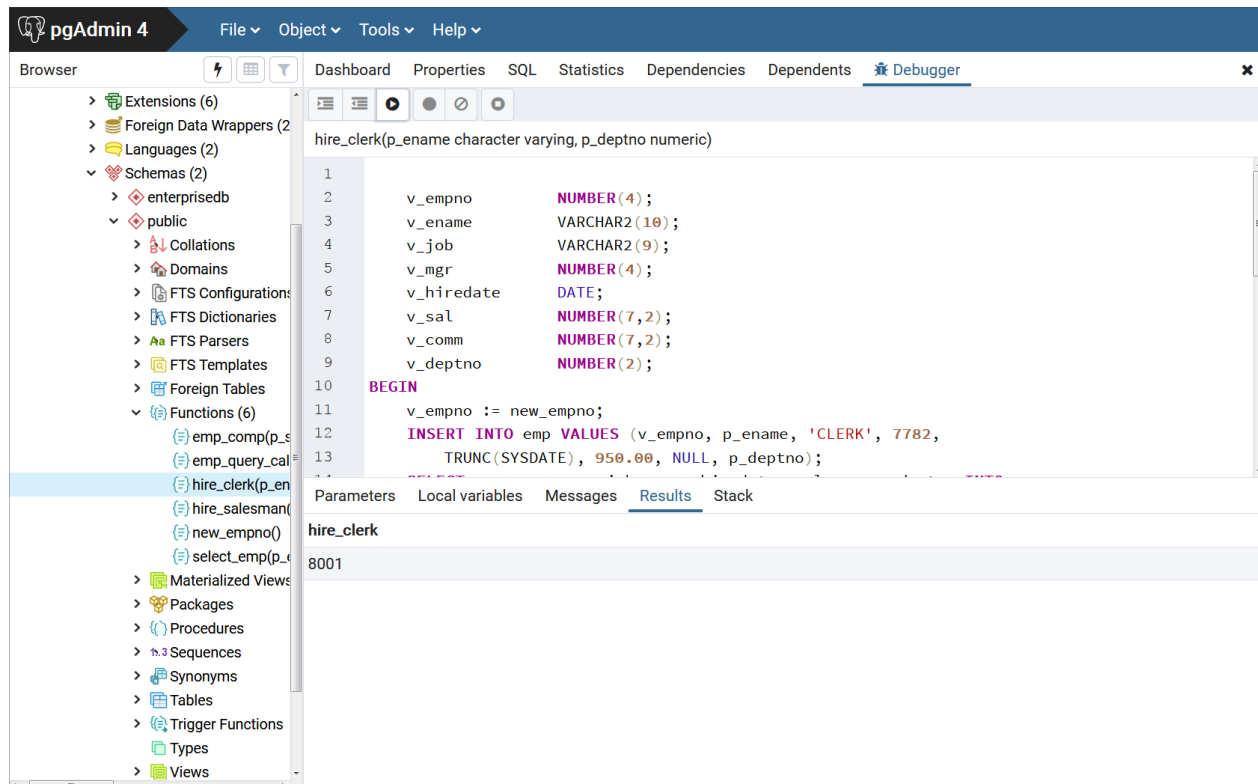


図 9 結果タブ

6.4.3 スタックタブ

Stack タブには、現在コールスタック上にあるプログラム（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、プログラムの名前が Stack タブに表示されるリストの一番上に追加されます。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、一連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役立ちます。

上記の図は、emp_query_caller が emp_query という名前のサブプログラムを呼び出そうとしていることを示しています。emp_query_caller は現在、コールスタックの最上位にあります。

emp_query の呼び出しが実行されると、Stack タブの上部に emp_query が表示され、そのコードが Program Body パネルに表示されます。

サブプログラムの実行が完了すると、制御は Stack タブの上部に表示される呼び出しプログラム

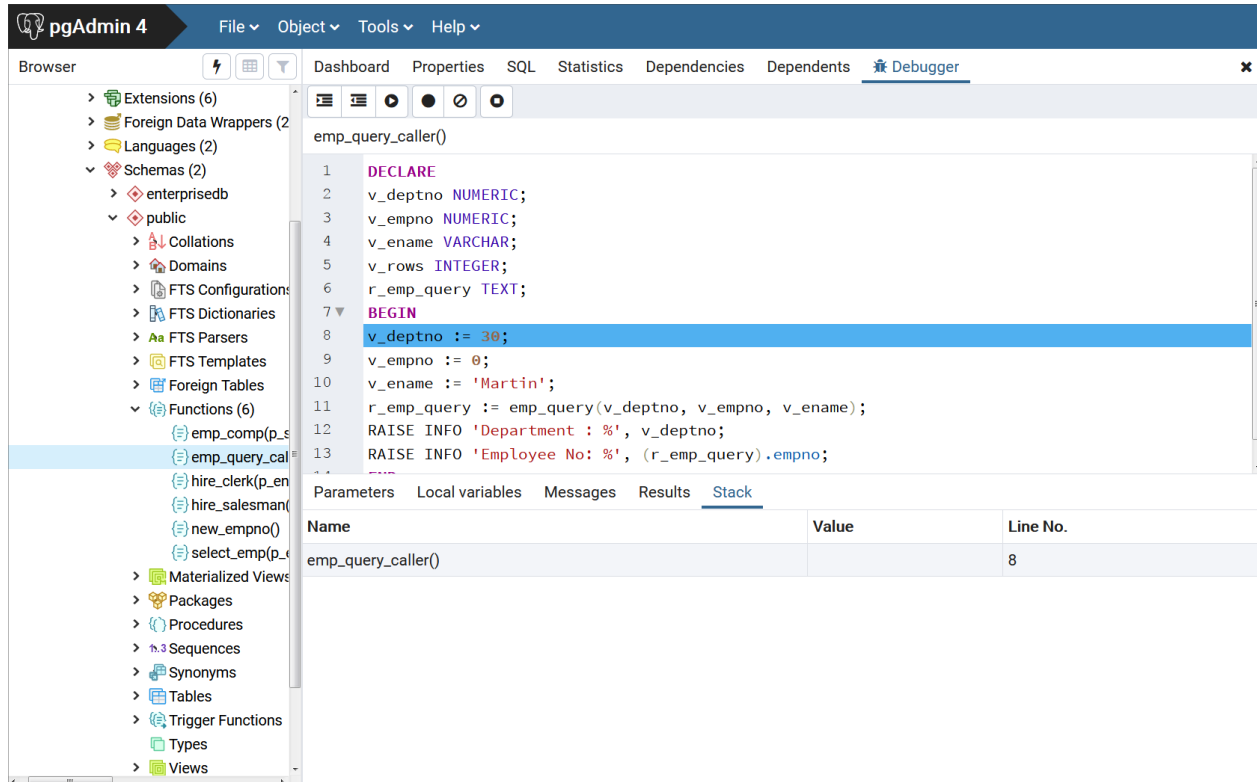


図 10 サブプログラムを呼び出すデバッグプログラム

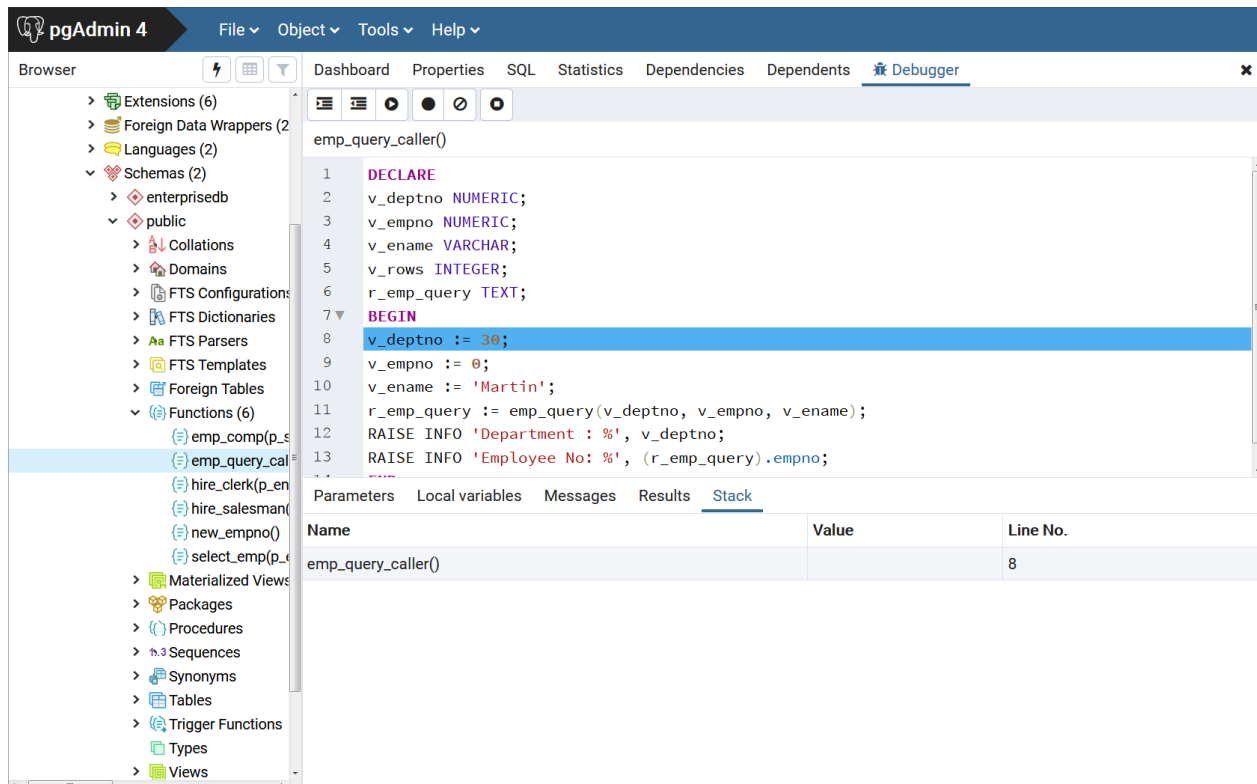


図 11 呼び出されたサブプログラムのデバッグ

(emp_query_caller) に戻ります。

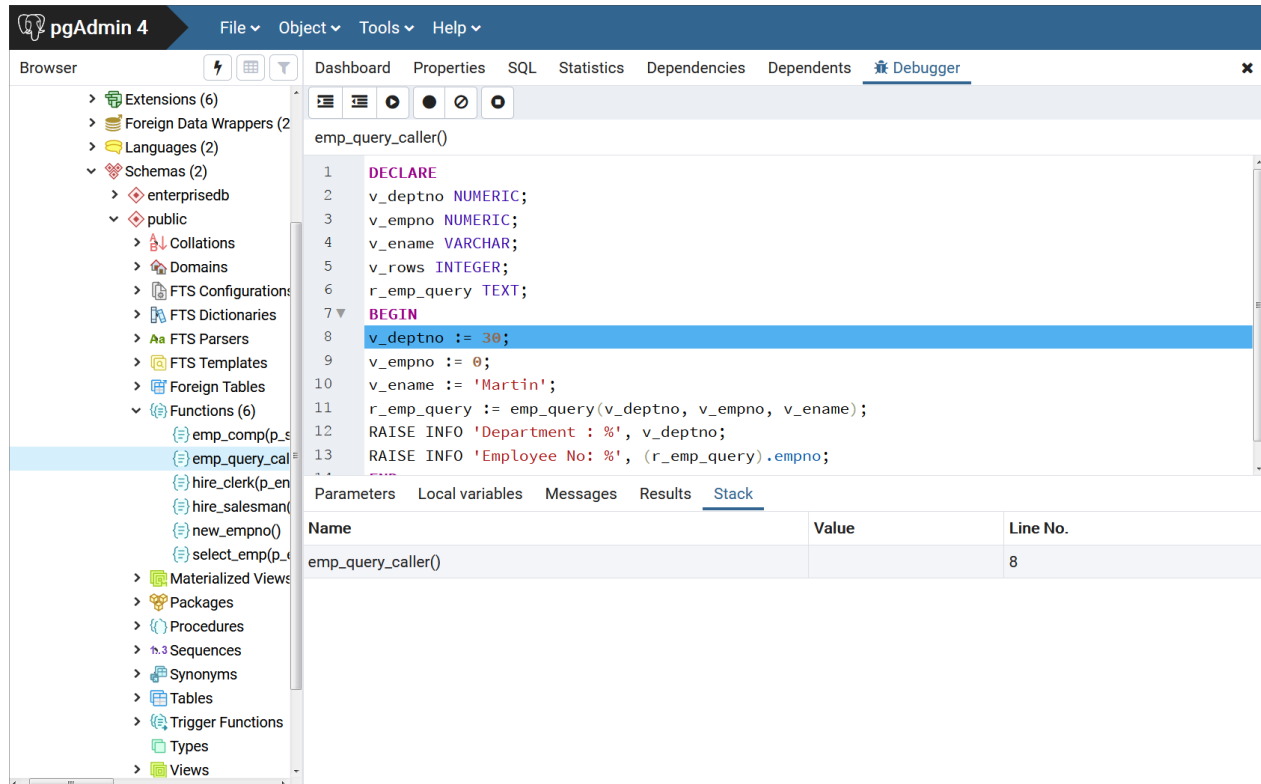


図 12 デバッグされたサブプログラムから制御が戻る

6.5 プログラムのデバッグ

次の操作を実行して、プログラムをデバッグできます。

- プログラムを 1 行ずつステップ実行します
- ブレークポイントに到達するまでプログラムを実行します
- プログラム内のローカル変数値を表示および変更します

6.5.1 コードのステップ実行

ツールバー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。

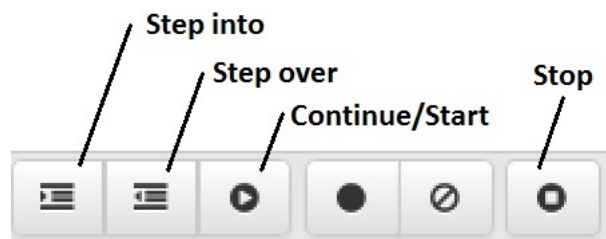


図 13 ツールバーのアイコン

アイコンの目的は次のとおりです。

- **Step into.** 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。
- **Step over.** 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関数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。
- **Continue/Start.** 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。
- **Stop.** 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、呼び出しスタック内のエントリに移動して変数を監視したり、他のブレークポイントを設定したりできます。操作への次のステップ、ステップオーバー、または続行操作により、デバッガーはブレークポイントに続くコードの次の行で実行を再開します。ブレークポイントには 2 つのタイプがあります。

Local Breakpoint - プログラム内の実行可能なコード行にローカルブレイクポイントを設定できます。デバッガーは、ローカルブレイ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。

****Global Breakpoint** - **任意のセッションがそのブレイクポイントに到達すると、グローバルブレイ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、グローバルブレイクポイントを設定します。プログラムにグローバルブレイクポイントが設定されると、グローバルブレイ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。グローバルブレイクポイントは、スーパーユーザーのみが設定できます。

ローカルブレイクポイントを作成するには、ローカルブレイ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 にブレイクポイントドットが表示されているスポットのように、マージンの右側に近いはずです。

デバッガーが作成されると、マージンに暗い点が表示され、選択したコード行にブレイクポイントが設定されたことを示します。

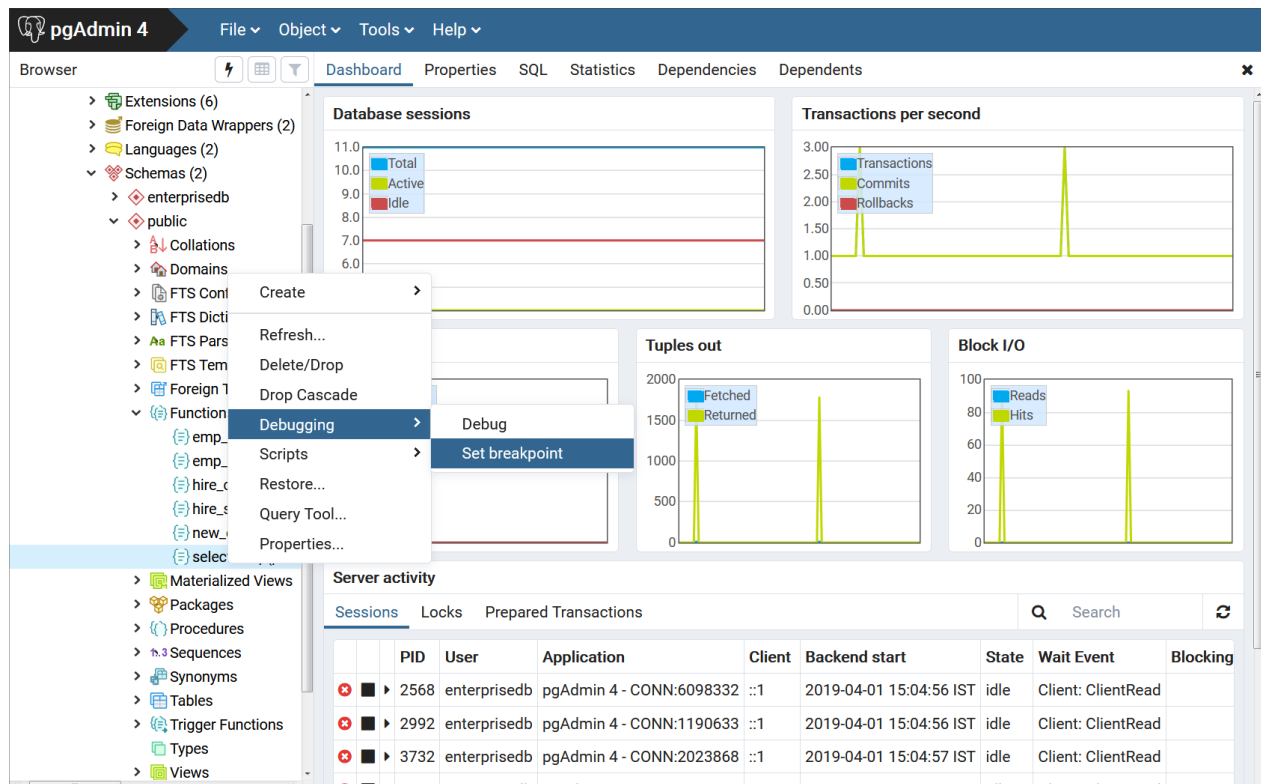


図 14 左マージンをクリックしてブレイクポイントを設定します

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

Removing a Local Breakpoint

ローカルブレイクポイントを削除するには、Program Body パネルのグレーの影付きマージンにあるブレイ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、ブレイクポイントが削除されたことを示します。

Clear all breakpoints アイコンをクリックすると、Program Body フレームに現在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 すべてのブレークポイントアイコンをクリア

注釈: 上記のアクションのいずれかを実行すると、Program Body パネルに現在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 Program Body パネルに現在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

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、関数、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 を選択し、次に Set Breakpoint を選択します。

または、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、関数、またはトリガーの名前を右クリックし、次のようにコンテキストメニューから Debugging を選択してから Set Breakpoint を選択します。

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関数を強調表示し、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、デバッグウィンドウが開き、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。

PSQL クライアントは、select_emp 関数（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 (13.0.0, server 13.0.0)
Type "help" for help.

edb=# SELECT select_emp(7900);
```

select_emp 機能は、デバッガでプログラムをステップスルーするまで完了しません。

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スルーすると、呼び出し側アプリケー

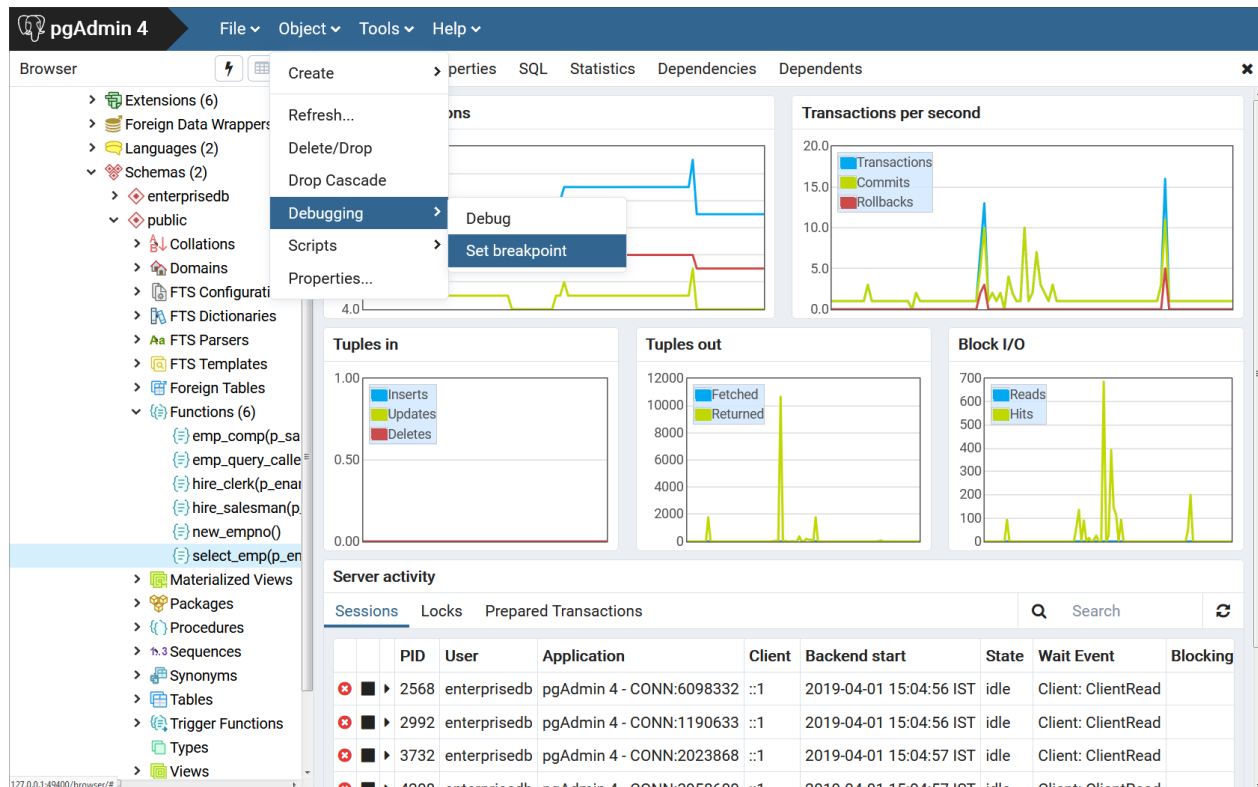


図 16 オブジェクトメニューからグローバルブレイクポイントを設定

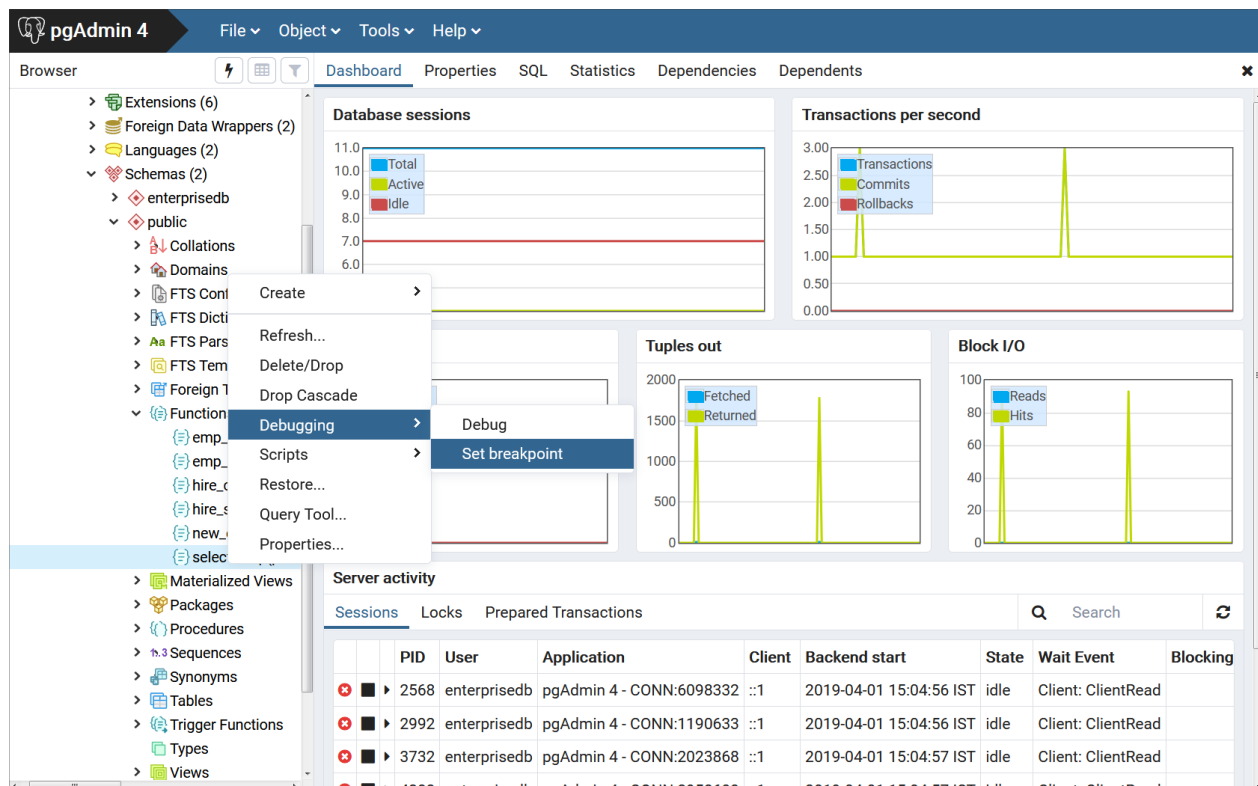


図 17 オブジェクトのコンテキストメニューからグローバルブレイクポイントを設定

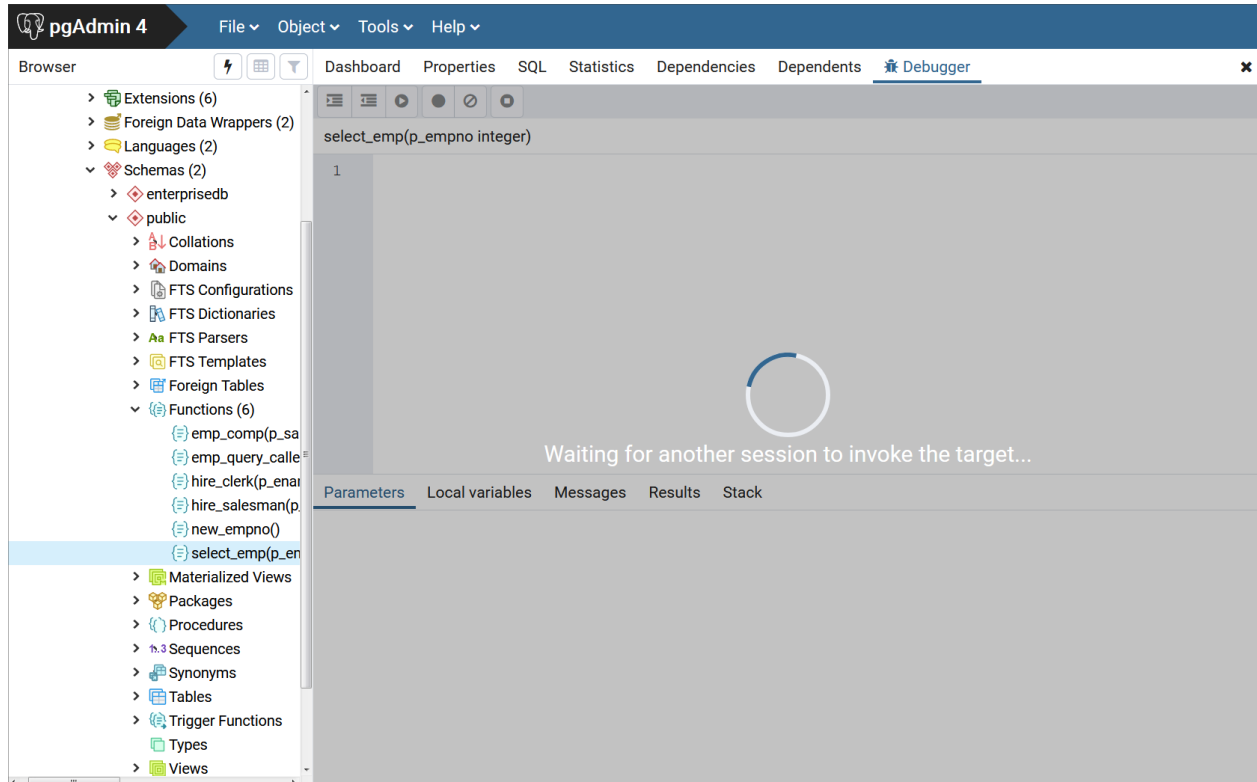


図 18 デバッグするプログラムの呼び出しを待機中

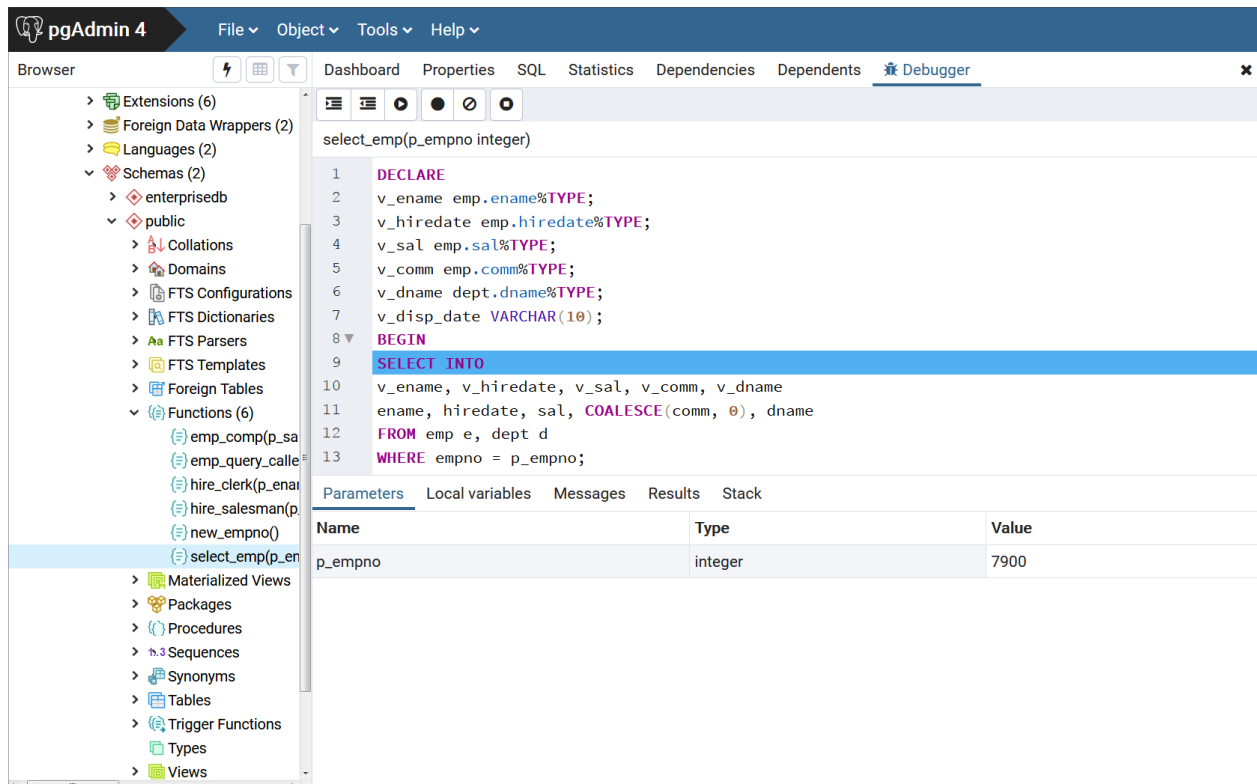


図 19 グローバルブレークポイントが設定されているプログラム

ション (PSQL) が制御を取り戻し、select_emp 関数が実行を完了し、その出力が表示されます。

```
$ psql edb enterprisedb
psql.bin (13.0.0, server 13.0.0)
Type "help" for help.

edb=# SELECT select_emp(7900);
INFO: Number : 7900
INFO: Name : JAMES
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、デバッガーセッションを終了できます。デバッガーセッションを終了しない場合、プログラムを呼び出す次のアプリケーションはグローバルブレークポイントに遭遇し、デバッグサイクルが再び開始されます。

6.5.4 デバッガーの終了

デバッガーセッションを終了してデバッガーを終了するには、右上隅にある閉じるアイコン (x) をクリックしてタブを閉じます。



図 20 デバッガからの終了

第 7 章

パフォーマンス分析とチューニング

AdvancedServer 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

AdvancedServer は、データベースサーバーの動的チューニングをサポートしており、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`

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ー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。

AdvancedServer が最初にインストールされるとき、`edb_dynatune` パラメーターは、インストールされたホストマシン（開発マシン、混合マシン、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、パフォーマンスを向上させるために、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。

AdvancedServer の初期インストール後に `edb_dynatune` パラメーターの値を変更するには、`postgresql.conf` ファイルを編集します。新しい構成を有効にするには、ポストマスターを再起動する必要があります。

`edb_dynatune` パラメーターは、0 から 100 までの任意の整数値に設定できます。値が 0 の場合、動的チューニング機能がオフになり、データベースサーバーのリソース使用量が `postgresql.conf` ファイル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ーに割り当てます。この設定は、他の多くのアプリケーションが使用されている開発マシンで使用されます。

34 から 66 の範囲の値は、適度な量のリソースをデータベースサーバー専用にします。この設定は、AdvancedServer と同じマシン上で実行され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ー専用にします。この設定は、AdvancedServer の実行専用のホストマシンに使用されます。

edb_dynatune の値を選択すると、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ー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 で選択された対応する値を上書きします。パラメーターの値を変更するには、構成パラメーターのコメントを外し、目的の値を指定して、データベースサーバーを再起動します。

7.1.2 edb_dynatune_profile

edb_dynatune_profile パラメータは、データベースサーバーで予想されるワークロードプロファイルに基づいてチューニングの側面を制御するために使用されます。このパラメーターは、データベースサーバーの起動時に有効になります。

edb_dynatune_profile の可能な値は次のとおりです。

価値	使用法
oltp	データベースサーバー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。
報告	大量のデータのレポートに使用されるデータベースサーバーに推奨されます。
混合した	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されます。

7.2 EDB 待ち状態

EDB 待ち状態 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 Wait States Background Worker (EWSBW)

待ち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ー、そのセッションで実行されているクエリ、待機中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` パラメーターで指定されたユーザー設定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ある必要があります。

The following describes the installation process on a Linux system.

Step 1: EDB 待ち状態は、`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。xx は AdvancedServer のバージョン番号です。

Step 2: ワーカーを起動するには、`shared_preload_libraries` パラメーターを使用して `postgresql.conf` ファイルに登録する必要があります。次に例を示します。

```
shared_preload_libraries = '$libdir/edb_wait_states'
```

Step 3: データベースサーバーを再起動します。再起動に成功すると、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。

Step 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;
```

Step 5: EDB 待ち状態ワーカーを終了するには、`shared_preload_libraries` パラメーターから `$libdir/edb_wait_states` を削除し、データベースサーバーを再起動します。

The following describes the installation process on a Windows system.

Step 1: EDB 待ち状態モジュールは、StackBuilderPlus ユーティリティを呼び出して EDB Modules インストーラー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。

Step 2: ワーカーに登録するには、`postgresql.conf` ファイルを変更して、`shared_preload_libraries` 構成パラメーターに待ち状態ライブラリーを含めます。パラメーター値には以下を含める必要があります。

```
shared_preload_libraries = '$libdir/edb_wait_states.dll'
```

EDB 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\as13\lib\
```

Step 3: 変更を有効にするためにデータベースサーバーを再起動します。再起動が成功すると、バックグラウンドワーカーが開始され、データの収集が開始されます。

Step 4: データを表示するには、次の拡張子を作成します。

```
CREATE EXTENSION edb_wait_states;
```

インストーラーは、`edb_wait_states.control` ファイルを次のパスに配置します。

```
C:\Program Files\edb\as13\share\extension
```

Terminating the Wait States Worker

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` 拡張子をドロップします。 `postgresql.conf` ファイルを変更し、 `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ーを再起動します。

The Wait States Interface

このインターフェースには、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。これらの各関数には、共通の入力パラメーターと出力パラメーターがあります。これらのパラメーターは次のとおりです。

- **start_ts and end_ts (IN).** を一緒に使用すると、時間間隔と読み取り対象のデータが指定されます。 `start_ts` のみを指定した場合、 `start_ts` から始まるデータが出力されます。 `end_ts` のみを指定すると、 `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。すべての関数は異なるデータを出力します。以下に、各機能の出力について説明します。
- **query_id (OUT).** 正規化されたクエリを識別します。これは、クエリから計算された内部ハッシュコードです。
- **session_id (OUT).** はセッションを識別します。
- **ref_start_ts and 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、クエリレコード、またはセッションレコードです。

各関数の構文については、次のセクションで説明します。

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

この関数は、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  
)
```

この関数を使用して、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 と end_ts で定義された）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();
```

Parameters

前に説明した共通パラメーターに加えて、出力の各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ー

query

セッションで実行されているクエリ

query_start_time

クエリが開始された時間

sample_time

待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

Example

以下は、`edb_wait_states_data()` 関数からの出力例です。

```
edb=# SELECT * FROM edb_wait_states_data();
-[ RECORD 1 ]-----+-----
session_id         | 4398
dbname             | edb
username           | enterprisedb
query              | SELECT schemaname FROM pg_tables, pg_sleep($1) WHERE
schemaname <> $2
query_start_time   | 17-AUG-18 11:56:05.271962 -04:00
sample_time        | 17-AUG-18 11:56:19.700236 -04:00
wait_event_type     | Timeout
wait_event         | PgSleep
-[ RECORD 2 ]-----+-----
session_id         | 4398
dbname             | edb
username           | enterprisedb
query              | SELECT schemaname FROM pg_tables, pg_sleep($1) WHERE
schemaname <> $2
query_start_time   | 17-AUG-18 11:56:05.271962 -04:00
sample_time        | 17-AUG-18 11:56:18.699938 -04:00
wait_event_type     | Timeout
wait_event         | PgSleep
-[ RECORD 3 ]-----+-----
session_id         | 4398
dbname             | edb
```

(次のページに続く)

(前のページからの続き)

```

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

この関数は、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 timestampz default '-infinity'::timestampz,
  IN end_ts timestampz default 'infinity'::timestampz,
  OUT query_id int8,
  OUT <query> text,
  OUT ref_start_ts timestampz
  OUT ref_end_ts timestampz
)

```

新しいクエリファイルは定期的に作成されるため、特定の間隔に対応して複数のクエリファイルが生成される場合があります。

この関数は、指定された時間間隔と重複するクエリファイル内のすべてのクエリを返します。以下に示すクエリは、start_ts と end_ts の間でサンプリングされたクエリを含むクエリファイル内のすべてのクエリを提供します。

つまり、この関数は、指定された間隔で実行されなかったクエリを出力する場合があります。ユーザーが edb_wait_states_data() を使用する必要があることを正確に知るため。

```
SELECT query FROM edb_wait_states_queries(start_ts, end_ts);
```

Parameters

前に説明した共通パラメーターに加えて、出力の各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

Example

以下は、`edb_wait_states_queries()` 関数からの出力例です。

```
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

この関数は、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
)
```

この機能を使用して、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、この関数は、特定の区画内でサンプリングされたセッションだけでなく、特定の区画内でサンプリングされたセッションを含むセッションファイルに記録されたすべてのセッションを出力します。それを識別するには、`edb_wait_states_data()` を使用する必要があります。

Parameters

前に説明した共通パラメーターに加えて、出力の各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ー

Example

以下は、`edb_wait_states_sessions()` 関数からの出力例です。

```
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
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

この関数は、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,
```

(次のページに続く)

(前のページからの続き)

```

    IN end_ts timestamptz default 'infinity'::timestamptz,
    OUT query_id int8,
    OUT session_id int4,
    OUT <query_start_time> timestamptz,
    OUT <sample_time> timestamptz,
    OUT <wait_event_type> text,
    OUT <wait_event> text
)

```

通常、ユーザーはこの関数を直接呼び出す必要はありません。

Parameters

前に説明した共通パラメーターに加えて、出力の各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

Example

以下は、edb_wait_states_samples() 関数からの出力例です。

```

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

```

(次のページに続く)

(前のページからの続き)

```

wait_event          | PgSleep
-[ RECORD 3 ]-----+-----
query_id            | 4292540138852956818
session_id          | 4340
query_start_time    | 17-AUG-18 11:56:00.39421 -04:00
sample_time         | 17-AUG-18 11:56:02.70001 -04:00
wait_event_type     | Timeout
wait_event          | PgSleep
-[ RECORD 4 ]-----+-----
query_id            | 4292540138852956818
session_id          | 4340
query_start_time    | 17-AUG-18 11:56:00.39421 -04:00
sample_time         | 17-AUG-18 11:56:03.700081 -04:00
wait_event_type     | Timeout
wait_event          | PgSleep
                    .
                    .
                    .

```

7.2.5 edb_wait_states_purge

この関数は、start_ts の後に作成され、end_ts の前にエージング（ローテーション）されたすべてのサンプル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 timestampz default '-infinity'::timestampz,
    IN end_ts timestampz default 'infinity'::timestampz
)

```

通常、ユーザーはこの関数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、この機能を使用できます。

サンプルが保持されている期間を知るには、その関数の前の例で説明したように edb_wait_states_data() を使用します。

Example

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/as13/data
[root@localhost data]# ls -l edb_wait_states
total 12
-rw----- 1 enterprisedb ... 253 Aug 17 11:56
edb_ws_queries_587836358698793_587922758698793

```

(次のページに続く)

(前のページからの続き)

```
-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() を実行した後の \$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/as13/data
[root@localhost data]# ls -l edb_wait_states
total 0
```


第 8 章

EDB クローンスキーマ

EDB クローンスキーマ は、ローカルサーバー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る `AdvancedServer` の拡張モジュールです。

ソースデータベースとターゲットデータベースは、同じ物理データベース、同じデータベースクラスター内の異なるデータベース、または異なるデータベースサーバー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る個別のデータベースにすることができます。

EDB クローンスキーマで次の関数を使用します。

- **localcopyschema.** この関数は、スキーマとそのデータベースオブジェクトのコピーをソースデータベースから元のスキーマ名とは異なるスキーマ名で同じデータベース（ターゲット）にコピーします。元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存在する場合、この関数を使用します。localcopyschema 関数の詳細については、[localcopyschema](#) を参照してください。
- ****localcopyschema_nb.****この機能は 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機能するため、機能を開始した端末を解放します。これは*non-blocking* 関数と呼ばれます。localcopyschema_nb 関数の詳細については、[localcopyschema_nb](#) を参照してください。
- **remotecopyschema.** この関数は、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。元のソーススキーマと結果のコピーが 2 つの別のデータベースに存在する場合、この関数を使用します。個別のデータベースは、同じまたは異なる `AdvancedServer` データベースクラスターに配置できます。remotecopyschema 関数の詳細については、[remotecopyschema](#) を参照してください。
- ****remotecopyschema_nb.****この機能は 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機能するため、機能を開始した端末を解放します。これは*non-blocking* 関数と呼ばれます。remotecopyschema_nb 関数の詳細については、[remotecopyschema_nb](#) を参照してください。
- **process_status_from_log.** この機能は、クローン作成機能のステータスを表示します。この情報は、クローン作成機能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 関数の詳細については、[process_status_from_log](#) を参照してください。

- **remove_log_file_and_job.** この機能は、クローン作成機能によって作成されたログファイルを削除します。この関数は、非ブロッキング形式の関数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 関数の詳細については、[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パーティションテーブルを含むテーブル
- インデックス
- 制約
- シーケンス
- 定義を見る
- 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
- 機能
- 手続き
- パッケージ
- サポートされているすべてのオブジェクトタイプに対するコメント
- 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）
- テーブルの論理複製属性
- 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
- 方針
- 演算子クラス

さらに、次の制限が適用されます。

- EDB クローンスキーマは、インストール中に AdvancedServer の Dialect ダイアログで Compatible with Oracle の方言が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 AdvancedServer 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。
- スキーマ間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機能呼び出す前に、この拡張機能をリモートデータベースのパブリックスキーマに作成する必要があります。
- スキーマを複製するために最大 16 個のコピージョブを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 クローンスキーマ機能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。

さらに、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。

These steps must be performed on any database to be used as the source or target database by an EDB Clone Schema function.

Step 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 拡張子を作成する前に、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 パッケージ（xx は AdvancedServer のバージョン番号）を使用して pgAgent をインストールできます。Windows では、StackBuilderPlus 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 コアドキュメントを参照してください。

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

Step 2: postgresql.conf ファイルを変更します。

次の例に示すように、shared_preload_libraries 構成パラメーターに \$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'
```

Step 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 AdvancedServer のバージョン番号です。Windows の場合、EDBPostgres 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、次の Web サイトで入手可能な*EDBPostgres 言語パックガイド*を参照してください。

‘<https://www.enterprisedb.com/edb-docs><<https://www.enterprisedb.com/resources/product-documentation>>‘ __

Step 4: データベースにスーパーユーザーとして接続し、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 コアドキュメントを参照してください。

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

8.1.2 構成パラメーターの設定

以下のセクションでは、`postgresql.conf` ファイルで変更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。

8.1.2.1 パフォーマンス構成パラメーター

1 つのトランザクションの一部として大きな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、クローン作成機能で参照されるソースデータベースサーバー用です。

調整が必要な可能性がある `postgresql.conf` ファイルの構成パラメーターには、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用されるメモリの量を指定します。
- **maintenance_work_mem.** は、`VACUUM`、`CREATE INDEX`、`ALTER TABLE ADD FOREIGN KEY` など、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。
- **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。
- ****checkpoint_flush_after.**** 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盤となる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.** このパラメーターは、各トランザクションに割り当てられるオブジェクトロックの平均数を制御します。すべてのトランザクションのロックがロックテーブルに収まる限り、個々のトランザクションはより多くのオブジェクトをロックできます。

構成パラメーターについては、次の PostgreSQL コアドキュメントを参照してください。

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

8.1.2.2 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、クローン作成機能呼び出すときに接続しているデータベースサーバーの `postgresql.conf` ファイルの `log_directory` パラメータ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、クローン作成機能を実行する前に存在している必要があります。

ログファイルの名前は、クローン作成機能呼び出すときにパラメーターリストで指定したものによって決まります。

ログファイルからステータスを表示するには、process_status_from_log 機能を使用します。

ログファイルを削除するには、remove_log_file_and_job 機能を使用するか、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。

These steps must be performed on any database to be used as the source or target database by an EDB Clone Schema function.

Step 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 拡張子もドロップします。

Step 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 拡張子を作成する前に、parallel_clone 拡張子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

ローカルクローン作成関数 `localcopyschema` または `localcopyschema_nb` のいずれかを使用する場合、必要なパラメーターの 1 つには、データベースサーバーとそのクローンスキーマのソースおよびレシーバーであるデータベースを識別するための単一の外部サーバーが含まれます。

リモートクローニング機能の 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。最初のパラメーターとして指定された外部サーバーは、複製されたスキーマのプロバイダーであるデータベースと共にソースデータベースサーバーを識別します。2 番目のパラメーターとして指定された外部サーバーは、ターゲットデータベースサーバーと、クローンスキーマの受信側であるデータベースを識別します。

外部サーバーごとに、ユーザーマッピングを作成する必要があります。選択したデータベーススーパーユーザーがクローン作成機能呼び出す場合、その機能呼び出すデータベーススーパーユーザーは、クローン作成機能のパラメーターとして指定された外部サーバーにアクセスできるデータベースユーザー名とパスワードにマップされている必要があります。

外部データ、外部サーバー、およびユーザーマッピングに関する一般情報については、次の PostgreSQL コードキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/ddl-foreign-data.html>

次の 2 つのセクションでは、これらの外部サーバーとユーザーマッピングの定義方法について説明します。

8.1.4.1 ローカルクローニング機能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` 機能の場合、ソーススキーマとターゲットスキーマは両方とも同じデータベースサーバーの同じデータベース内にあります。したがって、これらの機能に対して定義および指定する必要がある外部サーバーは 1 つだけです。この外部サーバーは、「ローカルサーバー」とも呼ばれます。

このサーバーは `localcopyschema` または `localcopyschema_nb` 機能呼び出すときに接続するため、このサーバーはローカルサーバーと呼ばれます。

ユーザーマッピングは、外部サーバーの接続および認証情報を定義します。

This foreign server and user mapping must be created within the database of the local server in which the cloning is to occur.

The database user for whom the user mapping is defined must be a superuser and the user connected to the local server when invoking an EDB Clone Schema function.

次の例では、複製するスキーマを含むデータベースの外部サーバーを作成し、複製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',
```

(次のページに続く)

(前のページからの続き)

```
port '5444',
dbname 'edb'
);
```

CREATE SERVER コマンドの使用の詳細については、次の PostgreSQL コードドキュメントを参照してください。

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

このサーバーのユーザーマッピングは次のとおりです。

```
CREATE USER MAPPING FOR enterprisedb SERVER local_server
OPTIONS (
    user 'enterprisedb',
    password 'password'
);
```

CREATE USER MAPPING コマンドの使用の詳細については、次の PostgreSQL コードドキュメントを参照してください。

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

次の psql コマンドは、外部サーバーとユーザーマッピングを示します。

```
edb=# \des+
List of foreign servers
-[ RECORD 1 ]-----+-----
Name                | local_server
Owner                | enterprisedb
Foreign-data wrapper | postgres_fdw
Access privileges   |
Type                 |
Version              |
FDW options          | (host 'localhost', port '5444', dbname 'edb')
Description          |

edb=# \deu+
                        List of user mappings
   Server   | User name |          FDW options
-----+-----+-----
 local_server | enterprisedb | ("user" 'enterprisedb', password 'password')
(1 row)
```

データベーススーパーユーザー enterprisedb がクローン作成機能呼び出すと、データベースユーザー enterprisedb とそのパスワードを使用して、ポート 5444 で localhost の local_server にデータベース 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ー

enterprisedb とデータベースユーザー enterprisedb は、同じデータベースユーザーであることがありますが、これは絶対的な要件ではありません。

これらの外部サーバーとユーザーマッピングの例の特定の使用方法については、localcopyschema に記載されている例を参照してください。

8.1.4.2 リモートクローニング機能の外部サーバーおよびユーザーマッピング

remotecopyschema および remotecopyschema_nb 機能の場合、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ーと、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ーは、ソースサーバー*または*リモートサーバー と呼ばれます。

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ーとそのデータベースを定義する外部サーバーは、ターゲットサーバー*または*ローカルサーバー と呼ばれます。

ターゲットサーバーは、remotecopyschema または remotecopyschema_nb 機能を呼び出すときに接続する必要があるサーバーであるため、ローカルサーバーとも呼ばれます。

ユーザーマッピングは、外部サーバーの接続および認証情報を定義します。

All of these foreign servers and user mappings must be created within the target database of the target/local server.

The database user for whom the user mappings are defined must be a superuser and the user connected to the local server when invoking an EDB Clone Schema function.

次の例では、クロンスキーマを受信するローカルターゲットデータベースの外部サーバーを作成します。

```
CREATE SERVER tgt_server FOREIGN DATA WRAPPER postgres_fdw
  OPTIONS (
    host 'localhost',
    port '5444',
    dbname 'tgtddb'
  );
```

このサーバーのユーザーマッピングは次のとおりです。

```
CREATE USER MAPPING FOR enterprisedb SERVER tgt_server
  OPTIONS (
    user 'tgtuser',
    password 'tgtpassword'
  );
```

次の例では、クロンスキーマのソースとなるリモートのソースデータベースの外部サーバーを作成します。

```
CREATE SERVER src_server FOREIGN DATA WRAPPER postgres_fdw
  OPTIONS (
    host '192.168.2.28',
    port '5444',
    dbname 'srcdb'
  );
```

このサーバーのユーザーマッピングは次のとおりです。

```
CREATE USER MAPPING FOR enterprisedb SERVER src_server
  OPTIONS (
    user 'srcuser',
    password 'srcpassword'
  );
```

次の psql コマンドは、外部サーバーとユーザーマッピングを表示します。

```
tgtodb=# \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 ]-----+-----
Name                | tgt_server
Owner                | tgtuser
Foreign-data wrapper | postgres_fdw
Access privileges    |
Type                 |
Version              |
FDW options          | (host 'localhost', port '5444', dbname 'tgtodb')
Description          |

tgtodb=# \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ー enterprisedb がクローン作成機能呼び出すと、データベースユーザー tgtuser とパスワード tgtpassword を使用して、localhost の tgt_server にポート 5444 でデータ

ベース tgtddb に接続します。

さらに、データベースユーザー srcuser とパスワード srcpassword は、ホスト 192.168.2.28 の src_server にポート 5444 でデータベース srcdb に接続します。

注釈: ソースデータベース srcdb を実行しているデータベースサーバーの pg_hba.conf ファイルに、ターゲットサーバーの場所（次の例ではアドレス 192.168.2.27）からの接続を許可する適切なエントリが含まれていることを確認してくださいソースサーバーとデータベースを定義する外部サーバー src_server のユーザーマッピングで。

#	TYPE	DATABASE	USER	ADDRESS	METHOD
# "local" is for Unix domain socket connections only					
local	all		all		md5
# IPv4 local connections:					
host	srcdb		srcuser	192.168.2.27/32	md5

これらの外部サーバーとユーザーマッピングの例の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 クローンスキーマ関数

`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 クローンスキーマ関数を使用する前に、次の条件を確認してください。

- ターゲットまたはローカルデータベースの外部サーバーの `CREATE USER MAPPING` コマンドで定義されたデータベーススーパーユーザーとして、ターゲットまたはローカルデータベースに接続します。
- `edb_util` スキーマが検索パスにあるか、`edb_util` プレフィックスを使用してクローン作成関数が呼び出されます。
- 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` パラメーターを `true` に設定すると、ターゲットデータベースクラスターには、ソーススキーマ内のオブジェクトによって参照されるすべてのテーブルスペースが含まれます。ターゲットスキーマで失敗します。これにより、クローン作成プロセスが失敗します。
- リモートコピー機能を使用する場合、`copy_acls` パラメーターを `true` に設定すると、ソーススキーマ内のオブジェクトに対して `GRANT` 権限を持つすべてのロールがターゲットデータベースクラスターに存在し、そうでない場合はそれらに権限を付与しますターゲットスキーマのロールは失敗します。これにより、クローン作成プロセスが失敗します。
- 関数の非ブロッキング形式を使用する場合、`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` のドキュメントを参照してください。

<https://www.pgadmin.org/docs/pgadmin4/dev/pgagent.html>

`pgAgent` は `AdvancedServer` 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

`localcopyschema` 関数は、`source_fdw` 外部サーバー内で指定されたローカルデータベース内のスキーマとそのデータベースオブジェクトを、ソーススキーマから同じデータベース内の指定されたターゲットスキーマにコピーします。

```
localcopyschema(
  <source_fdw> TEXT,
  <source_schema> TEXT,
  <target_schema> TEXT,
  <log_filename> TEXT
  [, <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 ]]]  
)
```

BOOLEAN 値が関数によって返されます。関数が成功すると、true が返されます。関数が失敗すると、false が返されます。

source_fdw, source_schema, target_schema と log_filename は必須パラメーターですが、他のすべてのパラメーターはオプションです。

Parameters

source_fdw

データベースオブジェクトのクローンを作成する postgres_fdw 外部データラッパーによって管理される外部サーバーの名前。

source_schema

データベースオブジェクトの複製元のスキーマの名前。

target_schema

ソーススキーマからデータベースオブジェクトのクローンを作成するスキーマの名前。

log_filename

関数からの情報が記録されるログファイルの名前。ログファイルは、postgresql.conf ファイルの log_directory 構成パラメータで指定されたディレクトリの下に作成されます。

on_tblspace

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する BOOLEAN 値。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、適用可能な 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

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 を印刷するかどうかを指定する BOOLEAN 値。false が指定されている場合、DDL は印刷されません。true を指定すると、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

ターゲットスキーマでオブジェクトを作成するときにアクセス制御リスト (ACL) を含めるかどうかを指定する BOOLEAN 値。アクセス制御リストは、GRANT の特権ステートメントのセットです。false が指定されている場合、アクセス制御リストはターゲットスキーマに含まれません。true を指定すると、アクセ

ス制御リストが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。

Example

次の例は、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。どちらも `local_server` で定義されているデータベース `edb` 内にあります。

この例は、次の環境用です。

- データベースサーバーが実行されているホスト:localhost
- 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/ターゲット:edb
- 外部サーバー (`local_server`) および先行する箇条書きの情報を使用したユーザーマッピング
- ソーススキーマ:edb
- 対象スキーマ:edbcopy
- `localcopyschema:enterprisedb` を呼び出すデータベーススーパーユーザー

関数を呼び出す前に、データベースユーザー `enterprisedb` 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` 機能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
-----
```

(次のページに続く)

(前のページからの続き)

```

-----
(FINISH,"2017-06-29 11:07:36.830783-04",3855,INFO,"STAGE:
FINAL","successfully cloned schema")
(1 row)

```

クローンが完了すると、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が次のように表示されます。

```

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
-------------	------	------

(次のページに続く)

(前のページからの続き)

EDBCOPY	EMP_COMP	FUNCTION
EDBCOPY	HIRE_CLERK	FUNCTION
EDBCOPY	HIRE_SALESMAN	FUNCTION
EDBCOPY	NEW_EMPNO	FUNCTION
EDBCOPY	EMP_ADMIN	PACKAGE
EDBCOPY	EMP_ADMIN	PACKAGE BODY
EDBCOPY	EMP_QUERY	PROCEDURE
EDBCOPY	EMP_QUERY_CALLER	PROCEDURE
EDBCOPY	LIST_EMP	PROCEDURE
EDBCOPY	SELECT_EMP	PROCEDURE
EDBCOPY	EMP_SAL_TRIG	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_a_19991"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_a_19992"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_a_19999"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_a_20000"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_a_20004"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_a_20005"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_c_19993"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_c_19994"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_c_20001"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_c_20002"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_c_20006"	TRIGGER
EDBCOPY	"RI_ConstraintTrigger_c_20007"	TRIGGER
EDBCOPY	USER_AUDIT_TRIG	TRIGGER

(24 rows)

8.2.2 localcopyschema_nb

localcopyschema_nb 機能は、source_fdw 外部サーバー内で指定されたローカルデータベース内のスキーマとそのデータベースオブジェクトを、ソーススキーマから同じデータベース内の指定されたターゲットスキーマにコピーしますが、ジョブが送信されると、非ブロッキング方式で pgAgent。

```
localcopyschema_nb(
  <source_fdw> TEXT,
  <source> TEXT,
  <target> TEXT,
  <log_filename> TEXT
  [, <on_tblspace> BOOLEAN
  [, <verbose_on> BOOLEAN
  [, <copy_acls> BOOLEAN
  [, <worker_count> INTEGER ]]]
)
```

INTEGER 値のジョブ ID は、pgAgent に送信されたジョブの関数によって返されます。関数が失敗すると、null が返されます。

source_fdw, source, target と log_filename は必須パラメーターですが、他のすべてのパラメーターはオプションです。

pgAgent ジョブの完了後、remove_log_file_and_job 機能を使用してジョブを削除します。

Parameters

source_fdw

データベースオブジェクトのクローンを作成する postgres_fdw 外部データラッパーによって管理される外部サーバーの名前。

source

データベースオブジェクトの複製元のスキーマの名前。

target

ソーススキーマからデータベースオブジェクトのクローンを作成するスキーマの名前。

log_filename

関数からの情報が記録されるログファイルの名前。ログファイルは、postgresql.conf ファイルの log_directory 構成パラメータで指定されたディレクトリの下に作成されます。

on_tblspace

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する BOOLEAN 値。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、適用可能な 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

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 を印刷するかどうかを指定する BOOLEAN 値。false が指定されている場合、DDL は印刷されません。true を指定すると、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

ターゲットスキーマでオブジェクトを作成するときにアクセス制御リスト (ACL) を含めるかどうかを指定する BOOLEAN 値。アクセス制御リストは、GRANT の特権ステートメントのセットです。false が指定されている場合、アクセス制御リストはターゲットスキーマに含まれません。true を指定すると、アクセス制御リストが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。

Example

localcopyschema の例と同じクローニング操作が実行されますが、ノンブロッキング関数 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、AdvancedServer インストールディレクトリの bin サブディレクトリにあります。

```
[root@localhost bin]# ./pgagent -l 2 -s /tmp/pgagent_edb_log
hostaddr=127.0.0.1 port=5444 dbname=edb user=enterprisedb password=password
```

注釈: pgagent -l 2 オプションは 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、記録する情報を減らします。

localcopyschema_nb 関数は、例で 4 と表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

remotecopyschema 関数は、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 外部サーバー内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。

```
remotecopyschema (
    <source_fdw> TEXT,
    <target_fdw> TEXT,
    <source_schema> TEXT,
    <target_schema> TEXT,
    <log_filename> TEXT
    [, <on_tblspace> BOOLEAN
    [, <verbose_on> BOOLEAN
    [, <copy_acls> BOOLEAN
    [, <worker_count> INTEGER ]]])
```

BOOLEAN 値が関数によって返されます。関数が成功すると、true が返されます。関数が失敗すると、false が返されます。

source_fdw, target_fdw, source_schema, target_schema と log_filename は必須パラメーターですが、他のすべてのパラメーターはオプションです。

Parameters

source_fdw

データベースオブジェクトのクローンを作成する postgres_fdw 外部データラッパーによって管理される外部サーバーの名前。

target_fdw

データベースオブジェクトのクローン先となる postgres_fdw 外部データラッパーによって管理される外部サーバーの名前。

source_schema

データベースオブジェクトの複製元のスキーマの名前。

target_schema

ソーススキーマからデータベースオブジェクトのクローンを作成するスキーマの名前。

`log_filename`

関数からの情報が記録されるログファイルの名前。ログファイルは、`postgresql.conf` ファイルの `log_directory` 構成パラメータで指定されたディレクトリの下に作成されます。

`on_tblspace`

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する BOOLEAN 値。 `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、適用可能な 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。

注釈: `true` が指定され、データベースオブジェクトに TABLESPACE 句があるが、そのテーブルスペースがターゲットデータベースクラスターに存在しない場合、クローン作成機能は失敗します。

`verbose_on`

ターゲットスキーマにオブジェクトを作成するときに `log_filename` に DDL を印刷するかどうかを指定する BOOLEAN 値。 `false` が指定されている場合、DDL は印刷されません。 `true` を指定すると、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`

ターゲットスキーマでオブジェクトを作成するときにアクセス制御リスト (ACL) を含めるかどうかを指定する BOOLEAN 値。アクセス制御リストは、GRANT の特権ステートメントのセットです。 `false` が指定されている場合、アクセス制御リストはターゲットスキーマに含まれません。 `true` を指定すると、アクセス制御リストが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。

注釈: `true` が指定され、GRANT 特権を持つロールがターゲットデータベースクラスターに存在しない場合、クローン作成機能は失敗します。

`worker_count`

クローンを並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。

Example

次の例は、`src_server` で定義されているデータベース `srcdb` 内のスキーマ `srcschema` を、`tgt_server` で定義されているデータベース `tgtdb` 内のターゲットスキーマ `tgtschema` に複製する方法を示しています。

移行元サーバー環境:

- 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) および先行する箇条書きの情報を使用したユーザーマッピング
- ソーススキーマ:srcschema

ターゲットサーバー環境:

- 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:tgtdb
- 外部サーバー (tgt_server) および先行する箇条書きの情報を使用したユーザーマッピング
- 対象スキーマ:tgtschema
- remotecopyschema:enterprisedb を呼び出すデータベーススーパーユーザー

関数を呼び出す前に、データベースユーザー enterprisedb がデータベース tgtdb に接続します。この機能には 4 のうち 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 current:18:05.299953 -04:00",4021,INFO,"STAGE: DATA-
COPY","[0][0] successfully copied data in [tgtschema.pgbench_tellers]
")
(1 row)
```

(次のページに続く)

(前のページからの続き)

```

tgtldb=# SELECT edb_util.process_status_from_log('clone_rmt_src_tgt');
                                     process_status_from_log
-----
--
(RUNNING,"28-JUN-17 current:18:06.634364 -04:00",4022,INFO,"STAGE: DATA-
COPY","[0][1] successfully copied data in [tgtschema.pgbench_history]
")
(1 row)

tgtldb=# SELECT edb_util.process_status_from_log('clone_rmt_src_tgt');
                                     process_status_from_log
-----
--
(RUNNING,"28-JUN-17 current:18:10.550393 -04:00",4039,INFO,"STAGE: POST-
DATA","CREATE PRIMARY KEY CONSTRAINT pgbench_tellers_pkey successful"
)
(1 row)

tgtldb=# SELECT edb_util.process_status_from_log('clone_rmt_src_tgt');
                                     process_status_from_log
-----
--
(FINISH,"28-JUN-17 current: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。

```

tgtldb=# \dt+
                                List of relations
 Schema |      Name      | Type | Owner  | Size  |
-----+-----+-----+-----+-----+
--
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 関数が呼び出されると、4つのバックグラウンドワーカーが指定されました。

ログファイルの次の部分 clone_rmt_src_tgt 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操作のステータスを示しています。

```

Wed Jun 28 current:18:05.232949 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] [0]
table count [4]
Wed Jun 28 current:18:05.233321 2017 EDT: [4019] INFO: [STAGE: DATA-COPY]
[0][0]
worker started to copy data
Wed Jun 28 current:18:05.233640 2017 EDT: [4019] INFO: [STAGE: DATA-COPY]
[0][1]
worker started to copy data
Wed Jun 28 current:18:05.233919 2017 EDT: [4019] INFO: [STAGE: DATA-COPY]
[0][2]
worker started to copy data
Wed Jun 28 current:18:05.234231 2017 EDT: [4019] INFO: [STAGE: DATA-COPY]
[0][3]
worker started to copy data
Wed Jun 28 current:18:05.298174 2017 EDT: [4024] INFO: [STAGE: DATA-COPY]
[0][3]
successfully copied data in [tgtschema.pgbench_branches]
Wed Jun 28 current:18:05.2999current 2017 EDT: [4021] INFO: [STAGE: DATA-
COPY] [0][0]
successfully copied data in [tgtschema.pgbench_tellers]
Wed Jun 28 current:18:06.634310 2017 EDT: [4022] INFO: [STAGE: DATA-COPY]
[0][1]
successfully copied data in [tgtschema.pgbench_history]
Wed Jun 28 current:18:10.477333 2017 EDT: [4023] INFO: [STAGE: DATA-COPY]
[0][2]
successfully copied data in [tgtschema.pgbench_accounts]
Wed Jun 28 current:18:10.477609 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] [0]
all workers finished [4]
Wed Jun 28 current:18:10.477654 2017 EDT: [4019] INFO: [STAGE: DATA-COPY] [0]
copy done [4] tables
Wed Jun 28 current: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つの角かっこ番号（[0][3] など）が含まれていることに注意してください。

最初の数字はジョブインデックスで、2番目の数字はワーカーインデックスです。ワーカーインデックス値の範囲は、4つのバックグラウンドワーカーの0から3です。

2つの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番目のログファイルにはジョブインデックスとして1があります。

8.2.4 remotecopyschema_nb

remotecopyschema_nb 関数は、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 外部サーバー内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とし

ての非ブロッキング方式。

```
remotecopyschema_nb(
  <source_fdw> TEXT,
  <target_fdw> TEXT,
  <source> TEXT,
  <target> TEXT,
  <log_filename> TEXT
  [, <on_tblspace> BOOLEAN
  [, <verbose_on> BOOLEAN
  [, <copy_acls> BOOLEAN
  [, <worker_count> INTEGER ]]]
)
```

INTEGER 値のジョブ ID は、pgAgent に送信されたジョブの関数によって返されます。関数が失敗すると、null が返されます。

source_fdw, target_fdw, source, target と log_filename は必須パラメーターですが、他のすべてのパラメーターはオプションです。

pgAgent ジョブの完了後、remove_log_file_and_job 機能を使用してジョブを削除します。

Parameters

source_fdw

データベースオブジェクトのクローンを作成する postgres_fdw 外部データラッパーによって管理される外部サーバーの名前。

target_fdw

データベースオブジェクトのクローン先となる postgres_fdw 外部データラッパーによって管理される外部サーバーの名前。

source

データベースオブジェクトの複製元のスキーマの名前。

target

ソーススキーマからデータベースオブジェクトのクローンを作成するスキーマの名前。

log_filename

関数からの情報が記録されるログファイルの名前。ログファイルは、postgresql.conf ファイルの log_directory 構成パラメータで指定されたディレクトリの下に作成されます。

on_tblspace

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する BOOLEAN 値。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、適用可能な 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。

注釈: true が指定され、データベースオブジェクトに TABLESPACE 句があるが、そのテーブルスペースがターゲットデータベースクラスターに存在しない場合、クローン作成機能は失敗します。

verbose_on

ターゲットスキーマにオブジェクトを作成するときに log_filename に DDL を印刷するかどうかを指定する BOOLEAN 値。false が指定されている場合、DDL は印刷されません。true を指定すると、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

ターゲットスキーマでオブジェクトを作成するときにアクセス制御リスト (ACL) を含めるかどうかを指定する BOOLEAN 値。アクセス制御リストは、GRANT の特権ステートメントのセットです。false が指定されている場合、アクセス制御リストはターゲットスキーマに含まれません。true を指定すると、アクセス制御リストが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。

注釈: true が指定され、GRANT 特権を持つロールがターゲットデータベースクラスターに存在しない場合、クローン作成機能は失敗します。

worker_count

クローンを並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。

Example

remotecopyschema の例と同じクローニング操作が実行されますが、ノンブロッキング関数 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、AdvancedServer インストールディレクトリの bin サブディレクトリにあります。

```
[root@localhost bin]# ./pgagent -l 1 -s /tmp/pgagent_tgtadb_log
hostaddr=127.0.0.1 port=5444 user=enterprisedb dbname=tgtadb password=password
```

remotecopyschema_nb 関数は、例で 2 と表示されているジョブ ID を返します。

```

tgtodb=# 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。

```

tgtodb=# 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。

```

tgtodb=# 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

process_status_from_log 機能は、ログファイルからクローン作成機能のステータスを提供します。

```

process_status_from_log (
    <log_file> TEXT
)

```

この関数は、ログファイルから次のフィールドを返します。

フィールド名	説明
状態	STARTING、RUNNING、FINISH、または FAILED のいずれかを表示します。
実行時間	コマンドが実行されたとき。タイムスタンプ形式で表示されます。
pid	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。
レベル	INFO、ERROR、または SUCCESSFUL のいずれかを表示します。
舞台	STARTUP、INITIAL、DDL-COLLECTION、PRE-DATA、DATA-COPY、POST-DATA、または FINAL のいずれかを表示します。
伝言	各コマンドまたは失敗に対応する情報。

Parameters

log_file

クローン作成機能が呼び出されたときに指定されたスキーマのクローン作成を記録するログファイルの名前。

Example

以下は、process_status_from_log 機能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

remove_log_file_and_job 機能は、スキーマクローン作成機能によって作成されたログファイルと、非ブロッキング機能によって作成されたジョブを削除することにより、クリーンアップタスクを実行します。

```
remove_log_file_and_job (
  { <log_file> TEXT |
    <job_id> INTEGER |
    <log_file> TEXT, <job_id> INTEGER
  }
)
```

remove_log_file_and_job 関数を呼び出すときに、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 のみを指定すると、関数はログファイルのみを削除します。
- job_id のみが指定されている場合、関数はジョブのみを削除します。
- 両方が指定されている場合、関数はログファイルとジョブを削除します。

Parameters

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。

Example

次の例では、ログファイル名を指定して、ログファイルのみを削除します。

```
edb=# SELECT edb_util.remove_log_file_and_job ('clone_edb_edbcopy');
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、両方の値を指定して、ログファイルとジョブを削除します。

```
tgtdb=# SELECT edb_util.remove_log_file_and_job ('clone_rmt_src_tgt',2);
remove_log_file_and_job
-----
t
(1 row)
```

第 9 章

拡張された SQL およびその他のその他の機能

AdvancedServer 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

PostgreSQL の 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、AdvancedServer は追加のオブジェクトタイプに関するコメントをサポートしています。サポートされ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) |
```

(次のページに続く)

(前のページからの続き)

```

OPERATOR CLASS <object_name> USING <index_method> |
OPERATOR FAMILY <object_name> USING <index_method> |
PACKAGE <object_name>
POLICY <policy_name> ON <table_name> |
[ PROCEDURAL ] LANGUAGE <object_name> |
PROCEDURE <proc_name> [[[[<argmode>] [<argname>] <argtype> [, ...]]]]
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'>

```

ここで、aggregate_signature は次のとおりです。

```

* |
[ <argmode> ] [ <argname> ] <argtype> [ , ... ] |
[ [ <argmode> ] [ <argname> ] <argtype> [ , ... ] ]
ORDER BY [ <argmode> ] [ <argname> ] <argtype> [ , ... ]

```

Parameters

object_name

コメントするオブジェクトの名前。

AGGREGATE aggregate_name (aggregate_signature)

集約に関するコメントを作成するには、AGGREGATE 句を含めます。aggregate_name はアグリゲートの名前を指定し、aggregate_signature は次のいずれかの形式で関連する署名を指定します。

```

* |
[ <argmode> ] [ <argname> ] <argtype> [ , ... ] |
[ [ <argmode> ] [ <argname> ] <argtype> [ , ... ] ]
ORDER BY [ <argmode> ] [ <argname> ] <argtype> [ , ... ]

```

argmode が関数、プロシージャ、または集約引数のモードである場合、argmode は IN、OUT、INOUT、また

は VARIADIC です。省略した場合、デフォルトは IN です。

argname は集約引数の名前です。

argtype は集約引数のデータ型です。

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

列に関するコメントを作成するには、COLUMN 句を含めます。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

CONSTRAINT 句を含めて、制約に関するコメントを追加します。制約に関するコメントを作成する場合、constraint_name は制約の名前を指定します。table_name または domain_name は、制約が定義されているテーブルまたはドメインの名前を指定します。

FUNCTION func_name ([[argmode] [argname] argtype [, ...]])

関数に関するコメントを追加するには、FUNCTION 句を含めます。func_name は、関数の名前を指定します。argmode は機能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、関数、プロシージャ、または集計引数の名前を指定します。argtype は、関数、プロシージャ、または集計引数のデータ型を指定します。

large_object_oid

large_object_oid は、コメントの対象であ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)

演算子に関するコメントを追加するには、OPERATOR 句を含めます。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、オプションでスキーマ修飾された、演算子の引数のデータ型です。

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、演算子クラスの関連付けられたインデックスメソッドを指定します。

```
OPERATOR FAMILY object_name USING index_method
```

演算子族に関するコメントを追加するには、OPERATOR FAMILY 句を含めます。object_name は、コメントを付ける演算子族の（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、演算子族の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 [, ...]))
```

手順に関するコメントを追加するには、PROCEDURE 句を含めます。proc_name は、プロシージャの名前を指定します。argmode は手順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、関数、プロシージャ、または集計引数の名前を指定します。argtype は、関数、プロシージャ、または集計引数のデータ型を指定します。

```
RULE rule_name ON table_name
```

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 はトランスフォームのデータ型の名前を指定し、lang_name はトランスフォームの言語の名前を指定します。

```
TRIGGER trigger_name ON table_name
```

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
```

文字列リテラルまたは NULL として記述されたコメントを削除するコメント。

Notes:

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、演算子族、パッケージ、プロシージャ、シーケンス、テキスト検索オブジェクト、型、およびビューの名前は、スキーマで修飾できます。

Example:

次の例では、new_emp という名前のテーブルにコメントを追加します。

```
COMMENT ON TABLE new_emp IS 'This table contains information about new
employees.';
```

COMMENT コマンドの使用の詳細については、次の PostgreSQL コアドキュメントを参照してください。

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

9.2 Functionversion () の出力

version() 関数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。

AdvancedServer の場合、version() の出力は、最初のテキストワードが AdvancedServer バージョン 10 以前の *EnterpriseDB* ではなく *PostgreSQL* であるという点で、PostgreSQL コミュニティバージョンと同様の形式です。

version () 出力の一般的な形式は次のとおりです。

```
PostgreSQL $PG_VERSION_EXT (EnterpriseDB Advanced Server $PG_VERSION) on $host
```

したがって、現在の AdvancedServer 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 13.0 (EnterpriseDB Advanced Server 13.0.0) 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、AdvancedServer10 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

スタンバイサーバー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ーに論理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 URL で入手可能な PostgreSQL コアドキュメントを参照してください。

<https://www.postgresql.org/docs/current/logicaldecoding-explanation.html>

スタンバイサーバー上の論理スロットが機能するには、スタンバイで `hot_standby_feedback` パラメーターを ON に設定する必要があります。 `hot_standby_feedback` パラメータは、VACUUM がスタンバイサーバー上の既存の論理レプリケーションスロット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した場合、スロットはドロップされます。

スタンバイでの論理デコードを機能させるには、プライマリサーバーとスタンバイサーバーの両方で `wal_level` を `logical` に設定する必要があります。 `wal_level` が `logical` 以外の値に設定されている場合、スロットは作成されません。プライマリで `wal_level` を `logical` 以外の値に設定し、スタンバイに既存の論理スロットがある場合、そのようなスロットはドロップされ、新しいスロットを作成できません。

トランザクションがプライマリサーバーに書き込まれると、アクティビティはスタンバイサーバー上に論理スロットの作成をトリガーします。プライマリサーバーがアイドル状態の場合、スタンバイサーバーに論理スロットを作成するのに時間がかかることがあります。

レプリケーションと論理デコードの例をサポートする関数の詳細については、次の場所にある PostgreSQL ドキュメントを参照してください。

<https://www.postgresql.org/docs/current/functions-admin.html#FUNCTIONS-REPLICATION>

<https://www.postgresql.org/docs/current/logicaldecoding-example.html>

第 10 章

システムカタログ表

次のシステムカタログ表には、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可能性があります。システムテーブルに格納された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エイリアスごとに 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ーの制限付きアクセスを許可するパス名のエイリアスです。

ディレクトリを使用して、ファイルシステム内の特定のディレクトリツリーにユーザーをフェンスできます。たとえば、UTL_FILE パッケージは、ユーザーがホストファイルシステム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パスへのアクセスのみを許可します。

コラム	種類	修飾子	説明
dirname	「名前」	ヌルではない	エイリアスの名前。
ダイロウナー	oid	ヌルではない	エイリアスを所有するユーザーの OID。
dirpath	テキスト		アクセスが許可されるディレクトリ名。
気違い	aclitem[]		エイリアスにアクセスできるユーザーを決定するアクセス制御リスト。

10.2 edb_all_resource_groups

edb_all_resource_groups テーブルには、CREATE RESOURCE GROUP コマンドで作成されたリソースグループごとに 1 行が含まれ、各リソースグループ内のアクティブなプロセスの数が表示されます。

コラム	種類	修飾子	説明
group_name	「名前」		リソースグループの名前。
active_processes	整数		リソースグループ内の現在アクティブなプロセスの数。
cpu_rate_limit	float8		リソースグループの最大 CPU レート制限。0 は制限なしを意味します。
per_process_cpu_rate_limit	float8		リソースグループ内の現在アクティブなプロセスごとの最大 CPU レート制限。
dirty_rate_limit	float8		リソースグループの最大ダーティレート制限。0 は制限なしを意味します。
per_process_dirty_rate_limit	float8		リソースグループで現在アクティブなプロセスごとのダーティレートの最大制限。

10.3 edb_policy

edb_policy テーブルには、ポリシーごとに 1 行が含まれます。

コラム	種類	修飾子	説明
ポリシー名	お名前	ヌルではない	ポリシー名。
ポリシーグループ	oid	ヌルではない	現在使用されていません。
ポリシーオブジェクト	oid	ヌルではない	このポリシーによって保護されたテーブルの OID (object_schema と object_name)。
政策の	チャー	ヌルではない	このポリシーで保護されているオブジェクトの種類: テーブルの 'r' ビューの場合は「v」 =同義語の場合 現在は常に「r」です。
policyproc	oid	ヌルではない	ポリシー関数の OID (function_schema と policy_function)。
ポリシー挿入	ブール値	ヌルではない	ポリシーが INSERT ステートメントによって実施されている場合は true。
ポリシー選択	ブール値	ヌルではない	ポリシーが SELECT ステートメントによって実施されている場合は true。
ポリシー削除	ブール値	ヌルではない	ポリシーが DELETE ステートメントによって実施されている場合は true。
ポリシー更新	ブール値	ヌルではない	ポリシーが UPDATE ステートメントによって実施されている場合は true。
policyindex	ブール値	ヌルではない	現在使用されていません。
ポリシー対応	ブール値	ヌルではない	ポリシーが有効な場合は true。
policyupdatecheck	ブール値	ヌルではない	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。
政策的	ブール値	ヌルではない	現在使用されていません。
ポリシータイプ	整数	ヌルではない	現在使用されていません。
ポリシー	整数	ヌルではない	現在使用されていません。
policyseccols	int2vector	ヌルではない	sec_relevant_cols にリストされている列の列番号。

10.4 edb_profile

edb_profile テーブルには、使用可能なプロファイルに関する情報が格納されます。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。

コラム	種類	参照資料	説明
oid	oid		行識別子（隠し属性、明示的に選択する必要があります）。
prfname	お名前		プロファイルの名前。
prffailedloginattempts	整数		プロファイルによって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。
prfpasswordlocktime	整数		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。
prfpasswordlifetime	整数		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。
prfpasswordgracetime	整数		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。
prfpasswordreusetime	整数		ユーザーがパスワードを再利用するまで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。
prfpasswordreusemax	整数		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。
prfpasswordverifyfuncdb	oid	pg_database.oid	パスワード検証機能が存在するデータベースの OID。
prfpasswordverifyfunc	oid	pg_proc.oid	プロファイルに関連付けられたパスワード検証機能の OID。

10.5 edb_redaction_column

カタログ `edb_redaction_column` には、テーブルの列に添付されたデータ編集ポリシーの情報が格納されます。

コラム	種類	参照資料	説明
oid	oid		行識別子（非表示属性、明示的に選択する必要があります）
rdpolicyid	oid	<code>edb_redaction_policy.oid</code>	データ改訂ポリシーは、説明されている列に適用されます
rdrelid	oid	<code>pg_class.oid</code>	説明された列が属するテーブル
rdattnum	int2	<code>pg_attribute.attnum</code>	記載されている列の番号
rdscope	int2		リダクションスコープ:1=クエリ、2=top_tlist、4=top_tlist_or_error
rdexception	int2		リダクション例外:8=なし、16=等しい、32=漏れ防止
rdfuncexpr	pg_node_tree		データ修正関数式

注釈: 表の墨消しポリシー `edb_redaction_column.rdpolicyid` が有効で、墨消しポリシー式 `edb_redaction_policy.rdexpr` が `true` と評価される場合、説明されている列は墨消しされます。

10.6 edb_redaction_policy

カタログ `edb_redaction_policy` には、テーブルの墨消しポリシーの情報が格納されます。

コラム	種類	参照資料	説明
oid	oid		行識別子（非表示属性、明示的に選択する必要があります）
rdname	お名前		データ編集ポリシーの名前
rdrelid	oid	<code>pg_class.oid</code>	データ編集ポリシーが適用されるテーブル
rdenable	ブール値		データ編集ポリシーは有効になっていますか？
rdexpr	pg_node_tree		データ改訂ポリシー式

注釈: データ編集ポリシーは、テーブルが有効で、式が真と評価された場合にテーブルに適用されます。

10.7 edb_resource_group

edb_resource_group テーブルには、CREATE RESOURCE GROUP コマンドで作成されたリソースグループごとに 1 行が含まれます。

コラム	種類	修飾子	説明
rgrpname	「名前」	ヌルではない	リソースグループの名前。
rgrpcpuratelimt	float8	ヌルではない	リソースグループの最大 CPU レート制限。 0 は制限なしを意味します。
rgrpdirtyratelimt	float8	ヌルではない	リソースグループの最大ダーティレート制限。 0 は制限なしを意味します。

10.8 edb_variable

edb_variable テーブルには、パッケージレベル変数（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。

コラム	種類	修飾子	説明
varname	「名前」	ヌルではない	変数の名前。
varpackage	oid	ヌルではない	パッケージを格納する pg_namespace 行の OID。
vartype	oid	ヌルではない	変数の型を定義する pg_type 行の OID。
varaccess	/"char/"	ヌルではない	変数がパッケージの外部から見える場合は +。 変数がパッケージ内でのみ表示される場合は -。 注:パブリック変数はパッケージヘッダー 内で宣言され、プライベート変数はパッ ケージ本体内で宣言されます。
varsrc	テキスト		変数のデフォルト値式を含む、変数宣言の ソースが含まれます。
varseq	小さい	ヌルではない	パッケージで変数が宣言された順序。

10.9 pg_synonym

pg_synonym テーブルには、CREATE SYNONYM コマンドまたは CREATE PUBLIC SYNONYM コマンドで作成された同義語ごとに 1 行が含まれます。

コラム	種類	修飾子	説明
シンネーム	「名前」	ヌルではない	シノニムの名前。
シンネームスペース	oid	ヌルではない	synowner を置き換えます。シノニムが格納されている pg_namespace 行の OID が含まれています
共オーナー	oid	ヌルではない	シノニムを所有するユーザーの OID。
synobjschema	「名前」	ヌルではない	参照されるオブジェクトが定義されているスキーマ。
synobjname	「名前」	ヌルではない	参照されるオブジェクトの名前。
シンリンク	テキスト		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。

10.10 product_component_version

product_component_version テーブルには機能の互換性に関する情報が含まれます。アプリケーションはインストール時または実行時にこのテーブルを照会して、アプリケーションで使用する機能がこの展開で利用可能であることを確認できます。

コラム	種類	説明
製品	さまざまなキャラクター (74)	製品の名前。
バージョン	さまざまなキャラクター (74)	製品のバージョン番号。
状態	さまざまなキャラクター (74)	リリースのステータス。

第 11 章

高度なサーバーの例外

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
value_error	22000	-6502	例外は、文字列の数値への変換が失敗したときに発生します。
invalid_number	22000	-6502	文字列の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。
datetime_value_out_of_range	22008	-1863	例外は、有効な範囲外の日付形式でフィールドを書き込むときに発生します。
divide_by_zero	22012	-1476	プログラムが数値をゼロで除算しようとする、例外が発生します。
zero_divide	22012	-1476	プログラムが数値をゼロで除算しようとする、例外が発生します。
dup_val_on_index	23505	-1	プログラムが一意の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保存しようとする、例外が発生します。
invalid_cursor	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、無効なカーソルでカーソル操作を試行したときに発生します。
cursor_already_open	42P03	-6511	プログラムが既に開いているカーソルを開こうとすると、例外が発生します。
collection_is_null	P1403	-6531	プログラムが、初期化されていないネストした表または VARRAY の要素に値を割り当てようとする、例外が発生します。
subscript_beyond_count	P1404	-6533	例外は、プログラムがコレクション内の要素の数より大きいインデックス番号を使用してネストした表または VARRAY を参照しようとする、例外が発生します。
subscript_outside_limit	P1405	-6532	例外は、プログラムが範囲外のインデックス番号を使用してネストした表または VARRAY 要素を参照しようとする、例外が発生します。

Redwood Built-in Package Exceptions

DBMS_CRYPTO Package			
ciphersuiteinvalid	00009	-28827	暗号スイートが定義されていない場合、例外が発生します。
ciphersuitenull	00009	-28829	例外は、暗号スイートに値が指定されていないか、NULL 値が含まれている場合に発生します。
keybadsize	00009	0	例外は、指定されたキーサイズが悪い場合に発生します。
keynull	00009	-28239	キーが指定されていない場合、例外が発生します。
UTL_FILE Package			
invalid_filehandle	P00010000	929282	例外は、ファイルハンドルが無効な場合に発生します。
invalid_maxlinesize	P00010000	929287	例外は、最大行サイズが無効であるか、最大行サイズの値が範囲内でない場合に発生します。
invalid_mode	P00010000	929281	FOPEN の open_mode パラメーターが無効な場合、例外が発生します。
invalid_operation	P00010000	929283	例外は、要求時にファイルを開いたり使用したりできなかった場合に発生します。
invalid_path	P00010000	929280	例外は、ファイルの場所またはファイル名が無効な場合に発生します。
read_error	P00010000	929284	例外は、読み取り操作中にオペレーティングシステムエラーが発生したときに発生します。
write_error	P00010000	929285	例外は、書き込み操作中にオペレーティングシステムエラーが発生したときに発生します。
UTL_HTTP Package			
end_of_body	P00010000	929266	HTTP 応答本文の最後に到達すると、例外が発生します。
UTL_URL Package			
bad_fixed_width_charset	00009	-29274	例外は、固定幅のマルチバイト文字が URL 文字セットとして許可されていない場合に発生します。
bad_url	P00010000	929262	例外は、URL に不正な形式のエスケープコードシーケンスが含まれている場合に発生します。

第 12 章

高度なサーバーキーワード

キーワードとは、AdvancedServer パーサーによって特別な意味または関連があると認識される単語です。pg_get_keywords() 関数を使用して、AdvancedServer 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
access	U	unreserved
...		

pg_get_keywords は、AdvancedServer 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。

名前が二重引用符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。特定のカテゴリに属する AdvancedServer キーワードの最新リストを取得するために、pg_get_keywords() 機能を選択的に照会できます。

```
SELECT * FROM pg_get_keywords() WHERE catcode = 'code';
```

code は次のとおりです。

R-単語は予約されています。予約済みのキーワードは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-単語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 - 単語は内部的に使用されますが、関数または型の名前として使用できます。

C - 単語は内部的に使用され、関数または型の名前として使用することはできません。

AdvancedServer の識別子とキーワードの詳細については、次の PostgreSQL コアドキュメントを参照してください。

<https://www.postgresql.org/docs/current/static/sql-syntax-lexical.html>

第 13 章

Conclusion

EDBPostgres™AdvancedServer ガイド

著作権 ©2007-2020EnterpriseDBCorporation。

無断複写・転載を禁じます。

EnterpriseDB®Corporation

34CrosbyDrive、Suite201、Bedford、MA01730、アメリカ

T+17813573390F+19784671307E

info@enterprisedb.com

www.enterprisedb.com

- EnterpriseDB および PostgresEnterpriseManager は、EnterpriseDBCorporation の登録商標です。EDB および EDBPostgres は、EnterpriseDBCorporation の商標です。Oracle は、Oracle、Inc の登録商標です。その他の商標は、それぞれの所有者の商標です。
- EDB は、存在する場合、EDBPostgres 製品のログオン UI の設計、コーディングのベストプラクティスの確立、レビュー、および入力検証の検証を行います。EDB は追加の入力コンポーネントに対して同じアプローチに従いますが、製品の性質上、限定された検証が可能な信頼できるユーザーによって入力および送信されるフリーフォーム SQL、WMI またはその他の文字列を受け入れる必要がある場合があります。このような場合、ユーザーが誤った入力や危険な入力を禁止することはできません。
- EDB は、将来、認証された信頼できるユーザーからのフリーフォーム SQL、WMI またはその他の潜在的に危険な入力を受け入れる製品に機能を追加する権利を留保しますが、そのような機能はすべて、可能な限り最小のリスクを提供するように設計およびテストされます可能であれば、スーパーユーザーまたは同等の特権が必要です。
- EDB は、すべての潜在的な脅威を予測できることを保証しないため、すべての潜在的な脆弱性が対処または検討されたことを完全に保証することはできません。

索引

- About the Examples Used in this Guide, 8
- Advanced Server Exceptions, 265
- Advanced Server Keywords, 269
- Audit Log File, 127
- Audit Logging Configuration Parameters, 106
- Auditing Settings, 56

- bgwriter_delay, 46

- checkpoint_completion_target, 44
- checkpoint_segments, 44
- checkpoint_timeout, 45
- Client Connection Defaults: Locale and Formatting, 61
- Client Connection Defaults: Other Defaults, 62
- Client Connection Defaults: Statement Behavior, 61
- Compatibility Options, 64
- Conclusion, 271
- Configuration Parameters, 31
- Configuration Parameters by Functionality, 40
- Configuring the Debugger, 188
- Conventions Used in Guide, 7
- custom_variable_classes, 72
- Customized Options, 72

- Database Administration, 31
- db_dialect, 70
- dbms_alert.max_alerts, 73
- dbms_pipe.total_message_buffer, 73
- Debugging a Program, 199
- default_heap_fillfactor, 61
- default_with_rowids, 71

- EDB Audit Logging, 106
- EDB Clone Schema, 221
- EDB Resource Manager, 149
- edb_audit, 56
- edb_audit_destination, 60
- edb_audit_directory, 56
- edb_audit_disconnect, 59
- edb_audit_filename, 57
- edb_audit_rotation_day, 57
- edb_audit_rotation_seconds, 58
- edb_audit_rotation_size, 58
- edb_audit_statement, 59
- edb_audit_tag, 60
- edb_data_redaction, 62
- edb_enable_pruning, 54
- edb_max_spins_per_delay, 49
- edb_redwood_date, 64
- edb_redwood_greatest_least, 64
- edb_redwood_strings, 67
- edb_sql_protect.enabled, 74
- edb_sql_protect.level, 75
- edb_sql_protect.max_protected_relations, 75
- edb_sql_protect.max_protected_roles, 76
- edb_sql_protect.max_queries_to_save, 77
- edb_stmt_level_tx, 69
- edb_wait_states.directory, 78
- edb_wait_states.retention_period, 79
- edb_wait_states.sampling_interval, 79
- edbldr.empty_csv_field, 79
- effective_cache_size, 48
- Enabling Audit Logging, 120
- Enhanced Compatibility Features, 21
- Enhanced SQL and Other Miscellaneous Features, 251

- icu_short_form, 61
- Index Advisor, 84
- Index Advisor Components, 86
- Index Advisor Configuration, 88
- Index Advisor Limitations, 99
- index_advisor.enabled, 73
- Introduction, 1

- libpq C Library, 175

- maintenance_work_mem, 43
- max_wal_size, 45
- min_wal_size, 46
- Modifying the pg_hba.conf File, 33
- Modifying the postgresql.conf File, 33

- odbc_lib_path, 63
- optimizer_mode, 72
- oracle_home, 63

- Performance Analysis and Tuning, 207
- pg_prewarm.autoprewarm, 50
- pg_prewarm.autoprewarm_interval, 51
- pgsnmpd, 103

- Query Tuning, 53
- Query Tuning: Planner Method Configuration, 54

- random_page_cost, 47
- Redacting Passwords from Audit Logs, 137
- Reporting and Logging, 55
- Resource Usage: EDB Resource Manager, 52
- Resource Usage: Memory, 51
- Reviewing the Index Advisor Recommendations, 94

- Selecting SQL Statements to Audit, 108
- seq_page_cost, 47
- Setting Configuration Parameters, 31
- shared_buffers, 42
- SQL Profiler, 100
- Starting the Debugger, 189
- Summary of Configuration Parameters, 34
- synchronous_commit, 48
- System Catalog Tables, 257

Top Performance Related Parameters, 42

Ungrouped, 81

Unicode Collation Algorithm, 139

Using Command Tags to Filter Audit Logs, 136

Using Error Codes to Filter Audit Logs, 135

Using Index advisor, 90

utl_encode.uudecode_redwood, 80

utl_file.umask, 80

wal_buffers, 44

What's New, 3

work_mem, 42