



EDB Postgres Advanced Server Security Features Guide

リリース *14.1.0*

EnterpriseDB

2021 年 12 月 18 日

目次

第 1 章	Protecting Against SQL Injection Attacks	3
1.1	SQL/Protect Overview	3
1.1.1	SQL インジェクション攻撃の種類	3
1.1.2	SQL インジェクション攻撃の監視	4
1.2	Configuring SQL/Protect	6
1.2.1	保護する役割の選択	9
1.2.2	保護されたロールの監視	11
1.3	Common Maintenance Operations	17
1.3.1	保護された役割リストへの役割の追加	17
1.3.2	保護された役割リストからの役割の削除	18
1.3.3	ロールの保護のタイプを設定する	18
1.3.4	保護された関係リストからの関係の削除	19
1.3.5	統計の削除	20
1.3.6	問題のあるクエリの削除	21
1.3.7	監視の無効化と有効化	22
1.4	Backing Up and Restoring a SQL/Protect Database	22
1.4.1	SQL/ Protect テーブルのオブジェクト識別番号	23
1.4.2	データベースのバックアップ	23
1.4.3	バックアップファイルからの復元	24
第 2 章	Virtual Private Database	29
第 3 章	sslutils	31
3.1	openssl_rsa_generate_key	31
3.2	openssl_rsa_key_to_csr	31
3.3	openssl_csr_to_cert	32
3.4	openssl_rsa_generate_crl	32
第 4 章	Data Redaction	35
4.1	修正ポリシーの作成	36
4.2	修正方針の変更	40
4.3	ドロップ修正ポリシー	42
4.4	システムカタログ	43
4.4.1	edb_redaction_column	43
4.4.2	edb_redaction_policy	44

このガイドでは、EDB Postgres Advanced Server インストールにセキュリティを追加する機能について説明します。Advanced Server に組み込まれて PostgreSQL が提供するセキュリティ機能の包括的なガイドではありません。

- **SQL/Protect** は、SQL インジェクション攻撃からの保護を提供します。
- **Virtual Private Database** は、機密データのきめ細かいアクセス制御を提供します。
- **sslutils** は、SSL 証明書を生成できる Postgres 拡張機能です。
- **Data redaction** 機能により、データの一部を動的にマスクできます。

Postgres の認証およびセキュリティ機能については、次の URL で入手可能な PostgreSQL コア文書を参照してください。

<https://www.postgresql.org/docs/>

第1章 Protecting Against SQL Injection Attacks

</div>

Advanced Server は、SQL インジェクション攻撃に対する保護を提供します。* SQL インジェクション攻撃*は、データベースのコンテンツ、構造、またはセキュリティ結果を攻撃者に提供する SQL ステートメントを実行することにより、データベースを侵害する試みです。

SQL インジェクション攻撃の防止は、通常、アプリケーション開発者の責任です。通常、データベース管理者は、潜在的な脅威をほとんどまたはまったく制御できません。データベース管理者にとっての困難は、アプリケーションが適切にファンクションするためにデータにアクセスできる必要があることです。

SQL/Protect は、データベース管理者がデータベースを SQL インジェクション攻撃から保護できるようにするモジュールです。SQL/Protect は、一般的な SQL インジェクションプロファイルの着信クエリを調べることにより、通常のデータベースセキュリティポリシーに加えてセキュリティのレイヤーを提供します。

SQL/Protect は、潜在的に危険なクエリを管理者に警告し、これらのクエリをブロックすることにより、データベース管理者に制御を戻します。

1.1 SQL/Protect Overview

</div>

このセクションでは、さまざまなタイプの SQL インジェクション攻撃の概要を序文し、SQL/Protect がそれらを防御する方法について説明します。

1.1.1 SQL インジェクション攻撃の種類

SQL インジェクション攻撃を実行するために使用されるさまざまな手法があります。各手法は、特定のシグネチャによって特徴付けられます。SQL/Protect は、次のシグネチャのクエリを調べます。

不正な関係

Advanced Server では、管理者がリレーション（テーブル、ビューなど）へのアクセスを制限できますが、多くの管理者はこの退屈なタスクを実行しません。SQL/Protect は、ユーザがアクセスするリレーションを追跡する学習モードを提供します。

これにより、管理者はアプリケーションのワークロードを調べることができ、SQL/Protect は、特定のユーザまたはロール内のユーザグループに対して、アプリケーションがアクセスできるリレーションを知ることができます。

SQL/ Protect が * passive または active * モードに切り替えられると、受信したクエリは学習したリレーションのリストと照合されます。

ユーティリティコマンド

SQL インジェクション攻撃で使用される一般的な手法は、通常は SQL データ定義言語 (DDL) ステートメントであるユーティリティコマンドを実行することです。例は、他のシステムリソースにアクセスする機能を持つユーザー定義ファンクションを作成することです。

SQL/ Protect は、すべてのユーティリティコマンドの実行をブロックできます。これらのコマンドは、標準のアプリケーション処理では通常必要ありません。

** SQL トートロジー**

SQL インジェクション攻撃で最も頻繁に使用される手法は、トートロジーの WHERE 句条件を発行することです (つまり、常に true の条件を使用します)。

次に例を示します。

```
WHERE password = 'x' OR 'x'='x'
```

攻撃者は通常、この手法を使用してセキュリティの弱点を特定しスタートします。SQL/ Protect は、トートロジー条件句を使用するクエリをブロックできます。

無制限の DML ステートメント

SQL インジェクション攻撃中に取られる危険なアクションは、無制限の DML ステートメントの実行です。これらは、WHERE 句のない UPDATE および DELETE ステートメントです。例、攻撃者はすべてのユーザーのパスワードを既知の値に更新したり、キーテーブル内のすべてのデータを削除してサービス拒否攻撃を開始したりする可能性があります。

1.1.2 SQL インジェクション攻撃の監視

このセクションでは、SQL/ Protect が SQL インジェクション攻撃を監視および報告する方法について説明します。

保護された役割

SQL インジェクション攻撃の監視には、セッションの現在のユーザが保護されたロールであるデータベースセッションで発生した SQL ステートメントの分析が含まれます。**保護されたロール**は、データベース管理者が SQL/ Protect を使用してモニタことを選択した Advanced Server ユーザまたはグループです。(Advanced Server では、ユーザーとグループをまとめて**ロール**と呼びます。)

保護される各ロールは、監視対象の SQL インジェクション攻撃のタイプに合わせてカスタマイズできるため、ロールごとに異なるレベルの保護を提供し、DBA のユーザメンテナンスのロードを大幅に削減できます。

スーパーユーザ権限を持つロールを保護されたロールにすることはできません。その後、保護された非スーパーユーザロールがスーパーユーザになるように変更された場合、そのスーパーユーザがコマンドを発行しようとするたびに特定の動作が発生します。

- 保護されたスーパーユーザによって発行されたすべてのコマンドで、SQL / Protect によってワーニングメッセージが発行されます。
- `edb_sql_protect_stats` の列スーパーユーザーの統計は、保護されたスーパーユーザが発行するコマンドごとに増加します。`edb_sql_protect_stats` ビューの詳細については、**攻撃試行統計**を参照してください。SQL / Protect がアクティブモードの場合、保護されたスーパーユーザが発行したすべてのコマンドは実行できません。

スーパーユーザ権限を持つ保護されたロールは、スーパースーパーユーザでなくなるように変更するか、保護されていないロールに戻す必要があります。

攻撃試行統計

SQL / Protect による攻撃と見なされる保護されたロールによるコマンドの使用が記録されます。* SQL インジェクション攻撃のタイプ*で説明されているように、統計は SQL インジェクション攻撃のタイプ別に収集されます。

これらの統計は、`edb_sql_protect_stats` ビューから簡単にアクセスでき、潜在的な攻撃のスタートを識別するために簡単に監視できます。

`edb_sql_protect_stats` の列は次をモニタします。

- **** username.** ******保護されたロールの名前。
- **** superusers.** ******保護されたロールがスーパーユーザの場合に発行される SQL ステートメントの数。実際、保護されたスーパーユーザによって発行された SQL ステートメントは、この統計を増加させます。保護されたスーパーユーザーについては、**保護された役割**を参照してください。
- **リレーション。** 保護されたロールによって学習されなかったリレーションを参照して発行された SQL ステートメントの数。(つまり、ロールの保護されたリレーションリストにないリレーション。)
- **** commands.** ******保護されたロールによって発行された DDL ステートメントの数。
- **** tautology.** ******トートロジー条件を含む保護されたロールによって発行された SQL ステートメントの数。
- **** dml.** ****** WHERE 句を含まない保護されたロールによって発行された UPDATE および DELETE ステートメントの数。

これにより、データベース管理者は、貴重なデータやその他の悪意のあるアクションの盗難を防止するために積極的に対応する機会を得ることができます。

ロールが複数のデータベースで保護されている場合、各データベースでの攻撃のロールの統計は個別に維持され、それぞれのデータベースに接続されている場合にのみ表示できます。

!!! Note * SQL / Protect 統計は、データベースサーバの実行中にメモリに保持されます。データベースサーバがシャットダウンされると、統計は Advanced Server ホームディレクトリサブディレクトリにある `edb_sqlprotect.stat` 名前付けのバイナリファイルに保存されます。

攻撃試行クエリ

SQL / Protect による攻撃と見なされる保護されたロールによるコマンドの使用は、`edb_sql_protect_queries` ビューに記録されます。

`edb_sql_protect_queries` ビューには次の列が含まれます。

- **** username.** ******データベースサーバへのログに使用される攻撃者のデータベースユーザ名前。
- **** ip_address.** ******攻撃が開始されたマシンの IP アドレス。
- **ポート。** 攻撃の発信元のポート番号。
- **** machine_name.** ******攻撃が発生したマシンの名前（わかっている場合）。
- **** date_time.** ******データベースサーバがクエリーを受信した日時。時間は 1 分の精度で保存されます。
- **** クエリー.** ******攻撃者が送信したクエリー文字列。

`edb_sql_protect_queries` に保存される問題のあるクエリの最大数は、`edb_sql_protect.max_queries_to_save` 設定パラメータによって制御されます。

ロールが複数のデータベースで保護されている場合、各データベースでの攻撃に対するロールのクエリは個別に維持され、それぞれのデータベースに接続している場合にのみ表示できます。

1.2 Configuring SQL/Protect

</ div>

SQL/ Protect を構成する前に、次の前提条件が満たされていることを確認してください。

- SQL/Protect の実行に必要なライブラリファイル (Linux では `sqlprotect.so`、Windows は `sqlprotect.dll`) は、Advanced Server ホームディレクトリサブディレクトリにインストールする必要があります。Windows の場合、これは Advanced Server インストーラで実行する必要があります。Linux の場合、`edb-asxx-server-sqlprotect` RPM パッケージをインストールします。xx は Advanced Server のバージョン番号です。
- Advanced Server ホームディレクトリサブディレクトリの `share/contrib` サブディレクトリにある SQL スクリプトファイル `sqlprotect.sql` も必要です。
- SQL/Protect を使用するようにデータベースサーバを構成し、SQL/Protect でモニタする各データベースを構成する必要があります。
 - データベースサーバ構成ファイル `postgresql.conf` は、SQL/Protect で使用される構成パラメーターを追加して有効にすることで変更する必要があります。-SQL/Protect が使用するデータベースオブジェクトは、SQL/Protect でモニタ各データベースにインストールする必要があります。

ステップ 1： Advanced Server ホームディレクトリサブディレクトリにある `postgresql.conf` ファイルの次の構成パラメーターを編集します。

- **** shared_preload_libraries.** ******ライブラリのリストに `$libdir/sqlprotect` を追加します。

- **** edb_sql_protect.enabled.** ****** SQL/Protect が保護されたロールによって発行された SQL ステートメントを分析し、**edb_sql_protect.level** の設定に従って反応することにより、それらのロールをアクティブにモニタリングかどうかを制御します。SQL/Protect でモニタリングを開始する準備ができれば、このパラメータを **on** に設定します。このパラメータを省略した場合、デフォルトは **off** です。
- **** edb_sql_protect. レベル.** ******保護されたロールによって SQL ステートメントが発行されたときに SQL/Protect によって実行されるアクションを設定します。このパラメータを省略すると、デフォルトの動作は **passive** です。最初に、このパラメータを **learn** に設定します。
 - 詳細については、*Setting the Protection Level* を参照してください。
- **** edb_sql_protect.max_protected_roles.** ******保護できるロールの最大数を設定します。このパラメータを省略すると、デフォルト設定は **64** です。
- **** edb_sql_protect.max_protected_relations.** ******ロールごとに保護できるリレーシヨンの最大数を設定します。このパラメータを省略すると、デフォルト設定は **1024** です。
 - サーバーの保護されたリレーシヨンの総数は、保護されたリレーシヨンの数に保護されたロールの数を掛けたものになることにノートしてください。すべての保護されたリレーシヨンは、共有メモリのスペースを消費します。可能な限り最大の保護されたリレーシヨンのためのスペースは、データベースサーバのスタートアップに予約れます。
- **** edb_sql_protect.max_queries_to_save.** **** edb_sql_protect_queries** ビューに保存する問題のあるクエリの最大数を設定します。このパラメータを省略すると、デフォルト設定は **5000** です。問題のあるクエリ数が制限に達すると、追加のクエリはビューに保存されませんが、データベースサーバのログファイルにアクセスできます。
 - このパラメータの最小有効値は **100** であることにノートしてください。 **100** より小さい値が指定された場合、データベースサーバはデフォルト設定の **5000** を使用して起動します。ワーニングメッセージがデータベースサーバのログファイルに記録されます。

次の例は、`postgresql.conf` ファイルのこれらのパラメーターの設定を示しています。

```
shared_preload_libraries = '$libdir/dbms_pipe,$libdir/edb_gen,$libdir/sqlprotect'
*
*                                     # (change requires restart)
*
*                                     .
*                                     .
*                                     .
edb_sql_protect.enabled = off
edb_sql_protect.level = learn
edb_sql_protect.max_protected_roles = 64
edb_sql_protect.max_protected_relations = 1024
edb_sql_protect.max_queries_to_save = 5000
```

ステップ 2: `postgresql.conf` ファイルを変更した後、データベースサーバを再起動します。

**** Linux の場合:** ****** `restart` オプションを使用して Advanced Server サービススクリプトを呼び出します。

Redhat または CentOS 7.x インストールでは、次のコマンドを使用します。

```
systemctl restart edb-as-14
```

**** Windows の場合： **** Windows サービスアプレットを使用して、edb-as-14 名前付けのサービスをリスタートします。

ステップ 3：SQL インジェクション攻撃から保護するデータベースごとに、データベースにスーパーユーザ（インストールオプションに応じて `enterprisedb` または `postgres` のいずれか）として接続し、`share/contrib` サブディレクトリにあるスクリプト `sqlprotect.sql` を実行します。Advanced Server のホームディレクトリ。このスクリプトは、`sqlprotect` 名前付けのスキーマに SQL/Protect データベースオブジェクトを作成します。

次の例は、edb 名前付けのデータベースの保護を設定するこのプロセスを示しています。

```
$ /usr/edb/as14/bin/psql -d edb -U enterprisedb
Password for user enterprisedb:
psql.bin (14.0.0, server 14.0.0)
Type "help" for help.

edb=# \i /usr/edb/as14/share/contrib/sqlprotect.sql
CREATE SCHEMA
GRANT
SET
CREATE TABLE
GRANT
CREATE TABLE
GRANT
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
DO
CREATE FUNCTION
CREATE FUNCTION
DO
CREATE VIEW
GRANT
DO
CREATE VIEW
GRANT
CREATE VIEW
GRANT
CREATE FUNCTION
CREATE FUNCTION
SET
```

1.2.1 保護する役割の選択

SQL/ Protect データベースオブジェクトがデータベースに作成された後、保護のために SQL クエリを監視するロール、および各ロールに割り当てる保護レベルを選択できます。

保護された役割リストの設定

保護するデータベースごとに、モニタするロールを決定し、それらのロールをそのデータベースの* protected roles リスト*に追加する必要があります。

ステップ 1: スーパーユーザとして、psql または Postgres Enterprise Manager Client で保護するデータベースに接続します。

```
$ /usr/edb/as14/bin/psql -d edb -U enterprisedb
Password for user enterprisedb:
psql.bin (14.0.0, server 14.0.0)
Type "help" for help.

edb=#
```

ステップ 2: SQL/ Protect テーブル、関数、およびビューは sqlprotect スキーマの下に構築されるため、SET search_path コマンドを使用して sqlprotect スキーマを検索パスに含めます。これにより、SQL/ Protect データベースオブジェクトを含むオペレーションまたはクエリーをスキーマで修飾する必要がなくなります。

```
edb=# SET search_path TO sqlprotect;
SET
```

ステップ 3: 保護したい各ロールは、保護された役割のリストに追加する必要があります。このリストは、edb_sql_protect テーブルに保持されます。

ロールを追加するには、ファンクション protect_role('rolename') を使用します。次の例は、appuser 名前付けのロールを保護します。

```
edb=# SELECT protect_role('appuser');
 protect_role
-----
(1 row)
```

次のクエリーを発行することにより、保護されたロールのリストに追加されたロールをリストできます。

```
edb=# SELECT * FROM edb_sql_protect;
 dbid  | roleid | protect_relations | allow_utility_cmds | allow_tautology |
allow_empty_dml
-----+-----+-----+-----+-----+-----
-----
```

(次のページに続く)

(前のページからの続き)

13917		16671		t			f			f			f
(1 row)													

オブジェクト識別番号 (OID) の代わりにオブジェクト名を使用して同じ情報を提供するビューも提供されます。

```
edb=# \x
Expanded display is on.
edb=# SELECT * FROM list_protected_users;
- [ RECORD 1 ]-----+-----
dbname          | edb
username        | appuser
protect_relations | t
allow_utility_cmds | f
allow_tautology  | f
allow_empty_dml  | f
```

</div>

保護レベルの設定

`edb_sql_protect.level` 設定パラメータは保護レベルを設定します。保護レベルは、保護されたロールが SQL ステートメントを発行したときの SQL / Protect の動作を定義します。定義された動作は、データベースサーバで SQL / Protect を使用して構成されたすべてのデータベースの保護されたロールリスト内のすべてのロールに適用されます。

`edb_sql_protect.level` 設定パラメータ (`postgresql.conf` ファイル) を次の値のいずれかに設定して、`learn` モード、`passive` モード、または `active` モードのいずれかを使用できます。

- **学習**保護されたロールのアクティビティを追跡し、ロールで使用されるリレーションを記録します。これは、保護されたアプリケーションの予想される動作が学習されるように、最初に SQL / Protect を構成するときに使用されます。
- **** passive. ****保護されたロールが定義されたルールに違反している場合に警告を発行しますが、SQL ステートメントの実行を停止しません。これは、SQL / Protect が保護されたロールの予想される動作を学習した後の次のステップです。これは基本的に侵入検知モードで動作し、適切に監視されている場合は稼働環境で実行できます。
- **** active. ****保護されたロールの無効なステートメントをすべて停止します。これは、危険なクエリの実行を防ぐ SQL ファイアウォールとして動作します。これは、攻撃者がアプリケーションの背後にある脆弱点とデータベースのタイプを特定しようとしている場合の早期侵入テストに対して特に効果的です。SQL / Protect はこれらの脆弱性ポイントを閉じるだけでなく、ブロックされたクエリを追跡して、攻撃者がシステムに侵入する別のメソッドを見つける前に管理者に警告することを可能にします。

`edb_sql_protect.level` パラメータが設定されていないか、構成ファイルから省略されている場合、SQL / Protect のデフォルトの動作は `passive` です。

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。

1.2.2 保護されたロールの監視

データベースで SQL / Protect を構成し、保護されたロールリストにロールを追加し、目的の保護レベルを設定したら、`learn` モード、`passive` モード、または `active` モードで SQL / Protect をアクティブにできます。その後、アプリケーションの実行をスタートできます。

新しい SQL/ Protect インストールでは、最初のステップは、保護されたロールが通常のオペレーション中にアクセスを許可されるリレーションを決定することです。学習モードでは、SQL/ Protect がアクセスされるリレーションを記録している間、ロールがアプリケーションを実行できます。これらは、テーブル `edb_sql_protect_rel` に保存されているロールの保護されたリレーションリストに追加されます。

SQL/ Protect がパッシブモードまたはアクティブモードで実行されると、攻撃に対する保護の監視が開始されます。パッシブモードとアクティブモードでは、ロールは通常の使用時にロールがアクセスできるリレーションであると判断されたため、保護されたリレーションリストのリレーションにアクセスすることが許可されます。

ただし、ロールが保護されたリレーションリストにないリレーションにアクセスしようとすると、SQL / Protect から `WARNING` または `ERROR` 重大レベルメッセージが返されます。リレーションに対するロールの試行されたアクションは、モードがパッシブかアクティブかによって実行される場合とされない場合があります。

</ div>

学習モード

ステップ 1: 学習モードで SQL / Protect をアクティブにするには、以下に示すように `postgresql.conf` ファイルのパラメーターを設定します。

```
edb_sql_protect.enabled = on
edb_sql_protect.level = learn
```

ステップ 2: `postgresql.conf` ファイルをリロードします。

Advanced Server アプリケーションメニューから `Expert Configuration` を選択し、次に `Reload Configuration` を選択します。

構成ファイルを再ロードする別のメソッドについては、`pg_reload_conf` ファンクションを使用します。次の例に示すように、必ずスーパーユーザとしてデータベースに接続し、`function pg_reload_conf` を実行してください。

```
edb=# SELECT pg_reload_conf();
      pg_reload_conf
-----
 t
(1 row)
```

ステップ 3: 保護されたロールがアプリケーションを実行できるようにします。

例として、保護されたロール `appuser` によって `psql` アプリケーションで次のクエリが発行されます。

```
edb=> SELECT * FROM dept;
NOTICE:  SQLPROTECT: Learned relation: 16384
 deptno |  dname   |  loc
-----+-----+-----
*      10 | ACCOUNTING | NEW YORK
*      20 | RESEARCH  | DALLAS
*      30 | SALES      | CHICAGO
*      40 | OPERATIONS | BOSTON
(4 rows)

edb=> SELECT empno, ename, job FROM emp WHERE deptno = 10;
NOTICE:  SQLPROTECT: Learned relation: 16391
 empno | ename  |  job
-----+-----+-----
*  7782 | CLARK  | MANAGER
*  7839 | KING   | PRESIDENT
*  7934 | MILLER | CLERK
(3 rows)
```

SQL / Protect は、ロールの保護されたリレーションリストにリレーションが追加されたことを示す **NOTICE** 重大レベルメッセージを生成します。

SQL/ Protect 学習モードでは、疑いの原因となっている SQL ステートメントの実行は阻止されませんが、次の例に示すように、潜在的に危険なステートメントをユーザに警告するメッセージが発行されます。

```
edb=> CREATE TABLE appuser_tab (f1 INTEGER);
NOTICE:  SQLPROTECT: This command type is illegal for this user
CREATE TABLE
edb=> DELETE FROM appuser_tab;
NOTICE:  SQLPROTECT: Learned relation: 16672
NOTICE:  SQLPROTECT: Illegal Query: empty DML
DELETE 0
```

ステップ 4: 保護されたロールがアプリケーションを実行するとき、ロールの保護されたリレーションリストへのリレーションの追加を監視するために、SQL / Protect テーブルを照会できます。

モニタリングしているデータベースにスーパーユーザとして接続し、検索パスに `sqlprotect` スキーマを含めるように設定します。

```
edb=# SET search_path TO sqlprotect;
SET
```

`edb_sql_protect_rel` テーブルをクエリして、保護されたリレーションリストに追加されたリレーションを確認します。

```
edb=# SELECT * FROM edb_sql_protect_rel;
 dbid   | roleid | relid
-----+-----+-----
 13917  | 16671  | 16384
 13917  | 16671  | 16391
 13917  | 16671  | 16672
(3 rows)
```

`list_protected_rels` ビューは、OID の代わりにオブジェクト名とともにより包括的な情報を提供します。

```
edb=# SELECT * FROM list_protected_rels;
 Database | Protected User | Schema | Name      | Type  | Owner
-----+-----+-----+-----+-----+-----
    edb   | appuser        | public | dept      | Table | enterprisedb
    edb   | appuser        | public | emp       | Table | enterprisedb
    edb   | appuser        | public | appuser_tab | Table | appuser
(3 rows)
```

パッシブモード

役割のアプリケーションが必要とするすべてのリレーションにアクセスしたことを確認したら、SQL/Protect が着信 SQL クエリをアクティブにモニタし、SQL インジェクション攻撃から保護できるように保護レベルを変更できます。

パッシブモードは、パッシブモードとアクティブモードの2つの保護モードの制限が緩和されます。

ステップ 1: パッシブモードで SQL/Protect をアクティブにするには、以下に示すように `postgresql.conf` ファイルで次のパラメーターを設定します。

```
edb_sql_protect.enabled = on
edb_sql_protect.level = passive
```

ステップ 2: *Learn Mode* セクションのステップ 2 に示すように、構成ファイルをリロードします。

これで、SQL/Protect はパッシブモードになりました。前の例の `dept` および `emp` テーブルなどの学習されたリレーションの場合、ユーザ `appuser` によって実行される次のクエリに示すように、SQL/Protect によるクライアントへの特別な通知なしに SQL ステートメントが許可されます。

```
edb=> SELECT * FROM dept;
 deptno |  dname   | loc
-----+-----+-----
*   10  | ACCOUNTING | NEW YORK
*   20  | RESEARCH  | DALLAS
*   30  | SALES     | CHICAGO
*   40  | OPERATIONS | BOSTON
(4 rows)
```

(次のページに続く)

(前のページからの続き)

```

edb=> SELECT empno, ename, job FROM emp WHERE deptno = 10;
 empno | ename  | job
-----+-----+-----
      * 7782 | CLARK  | MANAGER
      * 7839 | KING   | PRESIDENT
      * 7934 | MILLER | CLERK
(3 rows)

```

SQL/Protect は、SQL ステートメントの実行を妨げませんが、学習されなかったリレーションに対して実行された SQL ステートメント、または次の例に示す禁止されたシグネチャを含む SQL ステートメントに対して、重大度レベル **WARNING** のメッセージを発行します。

```

edb=> CREATE TABLE appuser_tab_2 (f1 INTEGER);
WARNING: SQLPROTECT: This command type is illegal for this user
CREATE TABLE
edb=> INSERT INTO appuser_tab_2 VALUES (1);
WARNING: SQLPROTECT: Illegal Query: relations
INSERT 0 1
edb=> INSERT INTO appuser_tab_2 VALUES (2);
WARNING: SQLPROTECT: Illegal Query: relations
INSERT 0 1
edb=> SELECT * FROM appuser_tab_2 WHERE 'x' = 'x';
WARNING: SQLPROTECT: Illegal Query: relations
WARNING: SQLPROTECT: Illegal Query: tautology
f1
----
      * 1
      * 2
(2 rows)

```

ステップ 3: 疑わしいアクティビティの統計を監視します。

ビュー `edb_sql_protect_stats` をクエリすることにより、ロールの保護されたリレーションリストになりリレーション、または SQL インジェクション攻撃シグネチャを含む SQL ステートメントが実行された回数を確認でき被参照。ビュー `edb_sql_protect_stats` の詳細については、**攻撃試行統計**を参照してください。

以下は、`edb_sql_protect_stats` のクエリーです。

```

edb=# SET search_path TO sqlprotect;
SET
edb=# SELECT * FROM edb_sql_protect_stats;
 username | superusers | relations | commands | tautology | dml
-----+-----+-----+-----+-----+-----
 appuser  |          0 |         3 |         1 |         1 |         0

```

(次のページに続く)

(前のページからの続き)

(1 row)

ステップ 4: 特定の攻撃に関する情報を表示します。

`edb_sql_protect_queries` ビューを照会することにより、ロールの保護されたリレーションリストに含まれていない関係、または SQL インジェクション攻撃シグネチャを含んでいたリレーションを被参照して実行された SQL ステートメントを確認できます。ビュー `edb_sql_protect_queries` の詳細については、**攻撃試行クエリ**を参照してください。

次のコードサンプルは、`edb_sql_protect_queries` のクエリーを示しています。

```
edb=# SELECT * FROM edb_sql_protect_queries;
- [ RECORD 1 ]+-----
username      | appuser
ip_address    |
port          |
machine_name  |
date_time     | 20-JUN-14 13:21:00 -04:00
query         | INSERT INTO appuser_tab_2 VALUES (1);
- [ RECORD 2 ]+-----
username      | appuser
ip_address    |
port          |
machine_name  |
date_time     | 20-JUN-14 13:21:00 -04:00
query         | CREATE TABLE appuser_tab_2 (f1 INTEGER);
- [ RECORD 3 ]+-----
username      | appuser
ip_address    |
port          |
machine_name  |
date_time     | 20-JUN-14 13:22:00 -04:00
query         | INSERT INTO appuser_tab_2 VALUES (2);
- [ RECORD 4 ]+-----
username      | appuser
ip_address    |
port          |
machine_name  |
date_time     | 20-JUN-14 13:22:00 -04:00
query         | SELECT * FROM appuser_tab_2 WHERE 'x' = 'x';
```

!!! Note * 攻撃が Unix ドメインソケット（つまり、`pg_hba.conf` 接続タイプ `local`）を使用してデータベースサーバと同じホストで発生した場合、`ip_address` 列と `port` 列は情報を結果ません。

アクティブモード

アクティブモードでは、許可されていない SQL ステートメントは実行できません。また、SQL / Protect によって発行されるメッセージの重大度レベルは、WARNING ではなく ERROR になります。

ステップ1: SQL/Protect をアクティブモードでアクティブにするには、以下に示すように `postgresql.conf` ファイルで次のパラメーターを設定します。

```
edb_sql_protect.enabled = on
edb_sql_protect.level = active
```

ステップ2: *Learn Mode* セクションのステップ2に示すように、構成ファイルをリロードします。

次の例は、*Passive Mode* のステップ2の例に示されているものと似ていますが、`edb_sql_protect.level` が `active` に設定されているときにユーザ `appuser` によって実行される SQL ステートメントを示しています。

```
edb=> CREATE TABLE appuser_tab_3 (f1 INTEGER);
ERROR: SQLPROTECT: This command type is illegal for this user
edb=> INSERT INTO appuser_tab_2 VALUES (1);
ERROR: SQLPROTECT: Illegal Query: relations
edb=> SELECT * FROM appuser_tab_2 WHERE 'x' = 'x';
ERROR: SQLPROTECT: Illegal Query: relations
```

結果の統計を以下に示します。

```
edb=# SELECT * FROM sqlprotect.edb_sql_protect_stats;
 username | superusers | relations | commands | tautology | dml
-----+-----+-----+-----+-----+-----
  appuser  |          0 |         5 |         2 |          1 |    0
(1 row)
```

以下は、`edb_sql_protect_queries` のクエリーです。

```
edb=# SELECT * FROM sqlprotect.edb_sql_protect_queries;
- [ RECORD 1 ]+-----
username      | appuser
ip_address    |
port          |
machine_name  |
date_time     | 20-JUN-14 13:21:00 -04:00
query         | CREATE TABLE appuser_tab_2 (f1 INTEGER);
- [ RECORD 2 ]+-----
username      | appuser
ip_address    |
port          |
machine_name  |
date_time     | 20-JUN-14 13:22:00 -04:00
```

(次のページに続く)

(前のページからの続き)

```

query      | INSERT INTO appuser_tab_2 VALUES (2);
- [ RECORD 3 ]+-----
username    | appuser
ip_address  | 192.168.2.6
port        | 50098
machine_name |
date_time   | 20-JUN-14 13:39:00 -04:00
query      | CREATE TABLE appuser_tab_3 (f1 INTEGER);
- [ RECORD 4 ]+-----
username    | appuser
ip_address  | 192.168.2.6
port        | 50098
machine_name |
date_time   | 20-JUN-14 13:39:00 -04:00
query      | INSERT INTO appuser_tab_2 VALUES (1);
- [ RECORD 5 ]+-----
username    | appuser
ip_address  | 192.168.2.6
port        | 50098
machine_name |
date_time   | 20-JUN-14 13:39:00 -04:00
query      | SELECT * FROM appuser_tab_2 WHERE 'x' = 'x';

```

1.3 Common Maintenance Operations

</div>

以下に、他の一般的な操作の実行方法を説明します。

これらの操作を実行するには、スーパーユーザとして接続し、検索パスに `sqlprotect` スキーマを含める必要があります。

1.3.1 保護された役割リストへの役割の追加

保護されたロールリストにロールを追加するには、次の例に示すように `protect_role('rolename')` を実行します。

```

edb=# SELECT protect_role('newuser');
 protect_role
-----
(1 row)

```

1.3.2 保護された役割リストからの役割の削除

保護された役割のリストからロールを削除するには、次のいずれかの機能を使用します。

```
unprotect_role('rolename')

unprotect_role(roleoid)
```

OID または `DROP USER SQL` ステートメントを使用してロールを削除してから、保護されたロールリストからロールを削除する場合、OID を使用する関数のバリエーションが役立ちます。SQL/Protect リレーションのクエリーがユーザ名前に `unknown` (OID=16458) などの値を返す場合、関数の `unprotect_role(roleoid)` フォームを使用して、保護されたロールリストから削除されたロールのエントリーを削除します。

これらの関数を使用してロールを削除すると、ロールの保護されたリレーションリストも削除されます。

削除されたロールの統計は、`drop_stats function` を使用するまで削除されません。

削除されたロールの問題のあるクエリは、`drop_queries function` を使用するまで削除されません。

以下は、`unprotect_role` ファンクションの例です。

```
edb=# SELECT unprotect_role('newuser');
 unprotect_role
-----
(1 row)
```

または、16693 の OID を指定することにより、ロールを削除できます。

```
edb=# SELECT unprotect_role(16693);
 unprotect_role
-----
(1 row)
```

1.3.3 ロールの保護のタイプを設定する

特定のタイプの SQL インジェクション攻撃からロールを保護するかどうかを変更できます。

ロールの保護を無効または有効にする SQL インジェクション攻撃のタイプに対応する `edb_sql_protect` の列のブール値を変更します。

`edb_sql_protect` を更新するステートメントの `WHERE` 句の次の列を必ず修飾してください。

- `** dbid`。 ******変更を行うデータベースの OID
- `** roleid`。 ******ブール設定を変更するロールの OID

例、特定のロールがユーティリティコマンドを発行できるようにするには、`allow_utility_cmds` 列を次のように更新します。

```
UPDATE edb_sql_protect SET allow_utility_cmds = TRUE WHERE dbid =
13917 AND roleid = 16671;
```

edb_sql_protect または list_protected_users を照会することにより、変更が行われたことを確認できます。次のクエリーでは、列 allow_utility_cmds に t が含まれていることにノートしてください。

```
edb=# SELECT dbid, roleid, allow_utility_cmds FROM edb_sql_protect;
 dbid   | roleid | allow_utility_cmds
-----+-----+-----
 13917  | 16671  | t
(1 row)
```

更新されたルールは、変更が行われてからロールによって開始された新しいセッションで有効になります。

1.3.4 保護された関係リストからの関係の削除

特定のリレーションが特定のロールに対してアクセス可能であることを SQL / Protect が学習した場合、その後、そのリレーションをロールの保護されたリレーションリストから削除できます。

次のいずれかの関数を使用して、edb_sql_protect_rel テーブルからエンTRIESを削除します。

```
unprotect_rel('rolename', 'relname')
unprotect_rel('rolename', 'schema', 'relname')
unprotect_rel(roleoid, reloid)
```

relname で指定されたリレーションが現在の検索パスにない場合、2 番目のファンクションフォーマットを使用してリレーションのスキーマを指定します。

3 番目のファンクションフォーマットでは、テキスト名の代わりに、それぞれロールおよびリレーションの OID を指定できます。

次の例は、ロール appuser の保護されたリレーションリストから public.emp リレーションを削除する方法を示しています。

```
edb=# SELECT unprotect_rel('appuser', 'public', 'emp');
 unprotect_rel
-----
(1 row)
```

次のクエリーは、emp リレーションのエンTRIESがなくなったことを示しています。

```
edb=# SELECT * FROM list_protected_rels;
 Database | Protected User | Schema | Name      | Type  | Owner
-----+-----+-----+-----+-----+-----
 edb      | appuser       | public | dept      | Table | enterprisedb
 edb      | appuser       | public | appuser_tab | Table | appuser
(2 rows)
```

ロールがそのリレーションを利用しようとするたびに、SQL/Protect はワーニングを発行するか、アクセスを完全にブロックします (edb_sql_protect.level の設定に応じて)。

</div>

1.3.5 統計の削除

次の 2 つの関数のいずれかを使用して、ビュー edb_sql_protect_stats から統計を削除できます。

```
drop_stats('rolename')
```

```
drop_stats(roleoid)
```

OID を使用したファンクションのバリエーションは、drop_stats('rolename') を使用してロールの統計を削除する前に、DROP ROLE または DROP USER SQL ステートメントを使用してロールを削除する場合に役立ちます。edb_sql_protect_stats のクエリがユーザ名前に unknown (OID=16458) などの値を返す場合、ファンクションの drop_stats(roleoid) フォームを使用して、削除されたロールの統計を edb_sql_protect_stats から削除します。

以下は、drop_stats ファンクションの例です。

```
edb=# SELECT drop_stats('appuser');
drop_stats
-----
(1 row)

edb=# SELECT * FROM edb_sql_protect_stats;
username | superusers | relations | commands | tautology | dml
-----+-----+-----+-----+-----+-----
(0 rows)
```

以下は、統計を削除する前にロールが削除されたときに、ファンクションの drop_stats(roleoid) フォームを使用する例です。

```
edb=# SELECT * FROM edb_sql_protect_stats;
*      username      | superusers | relations | commands | tautology | dml
-----+-----+-----+-----+-----+-----
unknown (OID=16693) |          0 |          5 |          3 |          1 |          0
appuser              |          0 |          5 |          2 |          1 |          0
(2 rows)

edb=# SELECT drop_stats(16693);
drop_stats
-----
(1 row)
```

(次のページに続く)

(前のページからの続き)

```
edb=# SELECT * FROM edb_sql_protect_stats;
username | superusers | relations | commands | tautology | dml
-----+-----+-----+-----+-----+-----
appuser  |           0 |          5 |          2 |          1 |    0
(1 row)
```

</div>

1.3.6 問題のあるクエリの削除

次の2つの関数のいずれかを使用して、ビュー `edb_sql_protect_queries` から問題のあるクエリを削除できます。

```
drop_queries('rolename')
```

```
drop_queries(roleoid)
```

OID を使用するファンクションのバリエーションは、`drop_queries('rolename')` を使用してロールの問題のあるクエリを削除する前に、`DROP ROLE` または `DROP USER SQL` ステートメントを使用してロールを削除する場合に役立ちます。`edb_sql_protect_queries` のクエリがユーザ名前に `unknown` (OID=16454) などの値を返す場合、ファンクションの `drop_queries(roleoid)` フォームを使用して、削除されたロールの問題のあるクエリを `edb_sql_protect_queries` から削除します。

以下は、`drop_queries` ファンクションの例です。

```
edb=# SELECT drop_queries('appuser');
drop_queries
-----
*           5
(1 row)

edb=# SELECT * FROM edb_sql_protect_queries;
username | ip_address | port | machine_name | date_time | query
-----+-----+-----+-----+-----+-----
(0 rows)
```

以下は、クエリを削除する前にロールが削除された場合に、ファンクションの `drop_queries(roleoid)` フォームを使用する例です。

```
edb=# SELECT username, query FROM edb_sql_protect_queries;
*      username      | query
-----+-----
unknown (OID=16454) | CREATE TABLE appuser_tab_2 (f1 INTEGER);
unknown (OID=16454) | INSERT INTO appuser_tab_2 VALUES (2);
unknown (OID=16454) | CREATE TABLE appuser_tab_3 (f1 INTEGER);
```

(次のページに続く)

(前のページからの続き)

```

unknown (OID=16454) | INSERT INTO appuser_tab_2 VALUES (1);
unknown (OID=16454) | SELECT * FROM appuser_tab_2 WHERE 'x' = 'x';
(5 rows)

edb=# SELECT drop_queries(16454);
 drop_queries
-----
*           5
(1 row)

edb=# SELECT * FROM edb_sql_protect_queries;
 username | ip_address | port | machine_name | date_time | query
-----+-----+-----+-----+-----+-----
(0 rows)

```

1.3.7 監視の無効化と有効化

SQL/Protect モニタリングをオフにする場合は、`postgresql.conf` ファイルを変更し、`edb_sql_protect.enabled` パラメータを `off` に設定します。ファイルを保存した後、サーバー構成をリロードして設定を適用します。

SQL/Protect モニタリングを有効にする場合は、`postgresql.conf` ファイルを変更し、`edb_sql_protect.enabled` パラメータを `on` に設定します。ファイルを保存した後、サーバー構成をリロードして設定を適用します。

1.4 Backing Up and Restoring a SQL/Protect Database

</ div>

SQL / Protect で構成されたデータベースをバックアップしてから、バックアップファイルを新しいデータベースに復元するには、バックアップおよび復元手順に通常関連付けられていることをさらに考慮する必要があります。これは主に、このセクションで説明するように、SQL/Protect テーブルでオブジェクト識別番号 (OID) を使用しているためです。

!!! Note * このセクションは、バックアップおよび復元手順により、`pg_dump` バックアッププログラムを使用する場合など、新しい OID で新しいデータベースにデータベースオブジェクトが再作成される場合に適用されます。

オペレーティングシステムのコピーユーティリティを使用して Advanced Server データファイルのバイナリイメージを作成するだけで Advanced Server データベースサーバをバックアップする場合 (ファイルシステムバックアップメソッド)、このセクションは適用されません。

1.4.1 SQL/Protect テーブルのオブジェクト識別番号

SQL/Protect は、2 つのテーブル (`edb_sql_protect` および `edb_sql_protect_rel`) を使用して、データベース、ロール、リレーションなどのデータベースオブジェクトに関する情報を格納します。これらのテーブル内のこれらのデータベースオブジェクトへの参照は、オブジェクトのテキスト名ではなく、オブジェクトの OID を使用して行われます。OID は、Advanced Server が各データベースオブジェクトを一意に識別するために使用する数値データタイプです。

データベースオブジェクトが作成されると、Advanced Server はオブジェクトに OID を割り当てます。OID は、データベースカタログ内のオブジェクトへのリファレンスが必要ときに使用されます。同じ `CREATE TABLE` ステートメントを持つテーブルなど、2 つのデータベースに同じデータベースオブジェクトを作成する場合、各テーブルには各データベースで異なる OID が割り当てられます。

バックアップされたデータベースオブジェクトが再作成されるバックアップおよび復元オペレーションでは、復元されたオブジェクトは、オリジナルのデータベースで割り当てられたものとは異なる新しいデータベースの OID になります。その結果、`edb_sql_protect` テーブルと `edb_sql_protect_rel` テーブルに格納されているデータベース、ロール、およびリレーションを参照する OID は、これらのテーブルが単にバックアップファイルにダンプされてから新しいデータベースに復元されると無効になります。

次のセクションでは、2 つの関数 `export_sqlprotect` および `import_sqlprotect` について説明します。これらは、SQL/Protect テーブルの OID がテーブルの復元後に正しいデータベースオブジェクトをリファレンス保証にするオーダーに、SQL/Protect テーブルのバックアップと復元に特に使用されます。

1.4.2 データベースのバックアップ

次の手順は、SQL/Protect で構成されたデータベースをバックアップします。

ステップ 1: `pg_dump` を使用してバックアップファイルを作成します。

次の例は、`pg_dump` ユーティリティプログラムを使用してデータベース `edb` から作成された `/tmp/edb.dmp` 名前付けのプレーンテキストバックアップファイルを示しています。

```
$ cd /usr/edb/as14/bin
$ ./pg_dump -U enterprisedb -Fp -f /tmp/edb.dmp edb
Password:
$
```

ステップ 2: スーパーユーザとしてデータベースに接続し、`export_sqlprotect('sqlprotect_file')` ファンクションを使用して SQL/Protect データをエクスポートします (`sqlprotect_file` は SQL/Protect データが保存されるファイルへの完全修飾パスです)。

`enterprisedb` オペレーティングシステムアカウント (Advanced Server を PostgreSQL 互換性モードでインストールした場合は `postgres`) には、`sqlprotect_file` で指定されたディレクトリへの読み取りおよび書き込みアクセスが必要です。

```
edb=# SELECT sqlprotect.export_sqlprotect('/tmp/sqlprotect.dmp');
 export_sqlprotect
-----
(1 row)
```

ファイル/tmp/edb.dmp および/tmp/sqlprotect.dmp は、データベースバックアップ全体を構成します。

1.4.3 バックアップファイルからの復元

ステップ 1: バックアップファイルを新しいデータベースに復元します。

次の例では、psql ユーティリティプログラムを使用して、プレーンテキストバックアップファイル/tmp/edb.dmp を、newdb 名前付けの新しく作成されたデータベースに復元します。

```
$ /usr/edb/as14/bin/psql -d newdb -U enterprisedb -f /tmp/edb.dmp
Password for user enterprisedb:
SET
SET
SET
SET
SET
COMMENT
CREATE SCHEMA
* .
* .
* .
```

ステップ 2: スーパーユーザとして新しいデータベースに接続し、edb_sql_protect_rel テーブルからすべての行を削除します。

このステップは、オリジナルのデータベースからバックアップされた edb_sql_protect_rel テーブル内の既存の行を削除します。これらの行には、バックアップファイルが復元されたデータベースに関連する正しい OID が含まれていません。

```
$ /usr/edb/as14/bin/psql -d newdb -U enterprisedb
Password for user enterprisedb:
psql.bin (14.0.0, server 14.0.0)
Type "help" for help.

newdb=# DELETE FROM sqlprotect.edb_sql_protect_rel;
DELETE 2
```

ステップ 3: edb_sql_protect テーブルからすべての行を削除します。

このステップは、オリジナルのデータベースからバックアップされた edb_sql_protect テーブル内の既存の行を削除します。これらの行には、バックアップファイルが復元されたデータベースに関連する正しい OID が含まれていません。

```
newdb=# DELETE FROM sqlprotect.edb_sql_protect;
DELETE 1
```

ステップ 4: データベースに存在する可能性のある統計を削除します。

この手順は、バックアップの復元先のデータベースに存在する可能性のある既存の統計を削除します。次のクエリーは、既存の統計を表示します。

```
newdb=# SELECT * FROM sqlprotect.edb_sql_protect_stats;
username | superusers | relations | commands | tautology | dml
-----+-----+-----+-----+-----+-----
(0 rows)
```

上記のクエリーに表示される行に対して、エントリーのロール名前を指定する `drop_stats` ファンクションを使用します。

例、`username` 列に `appuser` を持つ行が表示された場合、次のコマンドを発行して削除します。

```
newdb=# SELECT sqlprotect.drop_stats('appuser');
drop_stats
-----
(1 row)
```

ステップ 5: データベースに存在する問題のあるクエリを削除します。

この手順により、バックアップの復元先のデータベースに存在する可能性のある既存のクエリが削除されます。次のクエリーは、既存のクエリを表示します。

```
edb=# SELECT * FROM sqlprotect.edb_sql_protect_queries;
username | ip_address | port | machine_name | date_time | query
-----+-----+-----+-----+-----+-----
(0 rows)
```

上記のクエリーに表示される行に対して、エントリーのロール名前を指定する `drop_queries` ファンクションを使用します。

例、`username` 列に `appuser` を持つ行が表示された場合、次のコマンドを発行して削除します。

```
edb=# SELECT sqlprotect.drop_queries('appuser');
drop_queries
-----
(1 row)
```

ステップ 6: オリジナルのデータベースで SQL/Protect によって保護されたロール名が、新しいデータベースが存在するデータベースサーバに存在することを確認します。

オリジナルのデータベースと新しいデータベースが同じデータベースサーバにある場合、データベースサーバからこれらのロールを削除していないと仮定して、何もニーズはありません。

ステップ 7: ファンクション `import_sqlprotect('sqlprotect_file')` を実行します。 `sqlprotect_file` は、データベースのバックアップのステップ 2 で作成したファイルへの完全修飾パスです。

```
newdb=# SELECT sqlprotect.import_sqlprotect('/tmp/sqlprotect.dmp');
import_sqlprotect
```

(次のページに続く)

(前のページからの続き)

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

テーブル `edb_sql_protect` および `edb_sql_protect_rel` に、新しいデータベースで割り当てられたデータベースオブジェクトの OID を含むエントリが追加されました。統計ビュー `edb_sql_protect_stats` には、オリジナルのデータベースからインポートされた統計も表示されるようになりました。

これで、このデータベースの SQL/Protect テーブルと統計が適切に復元されます。これは、Advanced Server システムカタログでの次のクエリによって検証されます。

```
newdb=# SELECT datname, oid FROM pg_database;
```

```
  * datname | oid
-----+-----
 template1 |    1
 template0 | 13909
   edb     | 13917
  newdb    | 16679
```

```
(4 rows)
```

```
newdb=# SELECT rolname, oid FROM pg_roles;
```

```
  * rolname | oid
-----+-----
 enterprisedb |    10
   appuser    | 16671
   newuser    | 16678
```

```
(3 rows)
```

```
newdb=# SELECT relname, oid FROM pg_class WHERE relname IN
('dept','emp','appuser_tab');
```

```
  * relname | oid
-----+-----
 appuser_tab | 16803
   dept     | 16809
   emp      | 16812
```

```
(3 rows)
```

```
newdb=# SELECT * FROM sqlprotect.edb_sql_protect;
```

```
 dbid | roleid | protect_relations | allow_utility_cmds | allow_tautology |
allow_empty_dml
-----+-----+-----+-----+-----+-----
 16679 | 16671 | t                  | t                  | f                | f
```

```
(1 row)
```

```
newdb=# SELECT * FROM sqlprotect.edb_sql_protect_rel;
```

(次のページに続く)

(前のページからの続き)

```

dbid  | roleid | relid
-----+-----+-----
16679 | 16671 | 16809
16679 | 16671 | 16803
(2 rows)

newdb=# SELECT * FROM sqlprotect.edb_sql_protect_stats;
username | superusers | relations | commands | tautology | dml
-----+-----+-----+-----+-----+-----
appuser  |           0 |          5 |          2 |           1 |    0
(1 row)

newedb=# \x
Expanded display is on.
newedb=# SELECT * FROM sqlprotect.edb_sql_protect_queries;
- [ RECORD 1 ]+-----
username      | appuser
ip_address    |
port          |
machine_name  |
date_time     | 20-JUN-14 13:21:00 -04:00
query         | CREATE TABLE appuser_tab_2 (f1 INTEGER);
- [ RECORD 2 ]+-----
username      | appuser
ip_address    |
port          |
machine_name  |
date_time     | 20-JUN-14 13:22:00 -04:00
query         | INSERT INTO appuser_tab_2 VALUES (2);
- [ RECORD 3 ]+-----
username      | appuser
ip_address    | 192.168.2.6
port          | 50098
machine_name  |
date_time     | 20-JUN-14 13:39:00 -04:00
query         | CREATE TABLE appuser_tab_3 (f1 INTEGER);
- [ RECORD 4 ]+-----
username      | appuser
ip_address    | 192.168.2.6
port          | 50098
machine_name  |
date_time     | 20-JUN-14 13:39:00 -04:00
query         | INSERT INTO appuser_tab_2 VALUES (1);
- [ RECORD 5 ]+-----
username      | appuser

```

(次のページに続く)

(前のページからの続き)

ip_address	192.168.2.6
port	50098
machine_name	
date_time	20-JUN-14 13:39:00 -04:00
query	SELECT * FROM appuser_tab_2 WHERE 'x' = 'x';

テーブル `edb_sql_protect` および `edb_sql_protect_rel` の列については、次のことに注意してください。

- `** dbid`。 `** newdb` の `pg_database` の `oid` 列の値と一致
- `** roleid`。 `** appuser` の `pg_roles` の `oid` 列の値と一致

また、テーブルノートでは、リレーション `dept` および `appuser_tab` の `relid` 列の値が `pg_class` の `oid` 列の値とマッチすることに注意してください。

手順 8：新しいデータベースを実行しているデータベースサーバの `postgresql.conf` ファイルで、SQL / Protect 構成パラメーターが必要に応じて設定されていることを確認します。必要に応じて、データベースサーバを再起動するか、構成ファイルをリロードします。

これで、SQL / Protect を使用してデータベースをモニタできます。

第2章 Virtual Private Database

</div>

- 仮想プライベートデータベースは、セキュリティポリシーを使用したきめ細かいアクセス制御の一種です。Fine-grained access control *は、セキュリティポリシーで定義されているように、データへのアクセスを特定の行まで制御できることを意味します。

セキュリティポリシーをエンコードするルールは、ポリシーファンクションで定義されます。これは、特定の入力パラメーターと結果値を持つ SPL ファンクションです。セキュリティポリシーは、特定のデータベースオブジェクト（通常はテーブル）へのポリシーファンクションの名前付け関連付けです。

Advanced Server では、SPL に加えて、SQL や PL/pgSQL など、Advanced Server がサポートする任意の言語でポリシーファンクションを作成できます。

!!! Note * Advanced Server Virtual Private Database で現在サポートされているデータベースオブジェクトはテーブルです。ポリシーをビューまたはシノニムに適用することはできません。

仮想プライベートデータベースを使用する利点は次のとおりです。

- セキュリティのきめ細かいレベルを提供します。GRANT コマンドによって与えられたデータベースオブジェクトレベルの特権は、データベースオブジェクトのインスタンス全体へのアクセス特権を決定し、仮想プライベートデータベースは、データベースオブジェクトインスタンスの個々の行に対するアクセス制御を提供します。SQL コマンドのタイプ（INSERT, UPDATE, DELETE, or SELECT）に応じて、異なるセキュリティポリシーを適用できます。
- セキュリティポリシーは、データベースオブジェクトにアクセスアプリケーションのセッションユーザなどの要因に応じて、データベースオブジェクトに影響を与える該当する SQL コマンドごとに動的に変化します。
- セキュリティポリシーの呼び出しは、データベースオブジェクトにアクセスするすべてのアプリケーションに対して透過的であるため、セキュリティポリシーを適用するために個々のアプリケーションを変更する必要はありません。
- セキュリティポリシーを有効にすると、以下に示すシステム権限を除いて、アプリケーション（新しいアプリケーションを含む）がセキュリティポリシーを回避することはできません。
- スーパーユーザーであっても、以下に記載されているシステム権限を除き、セキュリティポリシーを回避することはできません。

!!! Note * セキュリティポリシーを回避できる唯一の方法は、EXEMPT ACCESS POLICY システム権限がユーザに付与されている場合です。この権限を持つユーザはデータベース内のすべてのポリシーから免除されるため、EXEMPT ACCESS POLICY 権限は細心の注意を払って付与する必要があります。

DBMS_RLS パッケージは、ポリシーの作成、ポリシーの削除、ポリシーの有効化、およびポリシーの無効化の手順を提供します。

第3章 sslutils

</div>

sslutils は、EDB Postgres Enterprise Manager サーバーで使用する SSL 証明書生成機能を Advanced Server に提供する Postgres 拡張機能です。sslutils は、edb-asxx-server-sslutils RPM パッケージを使用してインストールされます。xx は Advanced Server のバージョン番号です。

sslutils パッケージは、次のセクションに示す機能を提供します。

これらのセクションでは、関数のパラメータリスト内の各パラメータは、例が** Parameters **サブセクションの下に記述されているパラメータリスト。

3.1 openssl_rsa_generate_key

openssl_rsa_generate_key ファンクションは、RSA プライベートキーを生成します。ファンクションのシグネチャは次のとおりです。

```
openssl_rsa_generate_key(<integer>) RETURNS <text>
```

ファンクションを呼び出すときに、ビット数を整数値として渡します。ファンクションは生成されたキーを返します。

3.2 openssl_rsa_key_to_csr

openssl_rsa_key_to_csr ファンクションは、証明書署名要求（CSR）を生成します。シグネチャは次のとおりです。

```
openssl_rsa_key_to_csr(<text>, <text>, <text>, <text>, <text>, <text>,
<text>) RETURNS <text>
```

このファンクションは、証明書署名要求を生成して返します。

パラメータ

parameter 1

RSA キーファイルの名前。

parameter 2

署名要求を使用するエージェントのコモンネーム（agentN など）。

parameter 3

サーバーが存在する国の名前。

parameter 4

サーバーが存在する州の名前。

parameter 5

サーバーが存在する州内の場所（都市）。

parameter 6

証明書を要求する組織単位の名前。

parameter 7

証明書を要求しているユーザのメールアドレス。

3.3 openssl_csr_to_cert

openssl_csr_to_cert ファクションは、自己署名証明書または認証権限証明書を生成します。シグネチャは次のとおりです。

```
openssl_csr_to_cert(<text>, <text>, <text>) RETURNS <text>
```

このファクションは、自己署名証明書または認証権限証明書を返します。

パラメータ

parameter 1

要求に署名する証明書の名前。

parameter 2

認証権限証明書へのパス、または認証権限証明書を生成する場合は NULL。

parameter 3

認証局のプライベートキーへのパス、または（引数 2 が NULL の場合）プライベートキーへのパス。

3.4 openssl_rsa_generate_cert

openssl_rsa_generate_cert ファクションは、デフォルトの証明書失効リストを生成します。シグネチャは次のとおりです。

```
openssl_rsa_generate_cert(<text>, <text>) RETURNS <text>
```

このファンクションは、証明書失効リストを返します。

パラメータ

`parameter 1`

認証権限証明書へのパス。

`parameter 2`

認証権限のプライベートキーへのパス。

第4章 Data Redaction

</div>

- データのリダクション*は、特定のユーザーに対して表示されるデータを動的に変更することにより、機密データの露出を制限します。

例、社会セキュリティ番号（SSN）は 021-23-9567 として保存されます。特権ユーザーは完全な SSN を見ることができますが、他のユーザーは最後の 4 桁の xxx-xx-9567 しか見るできません。

データ改訂は、改訂が適用される各フィールドにファンクションを定義することにより実装されます。このファンクションは、データ修正のサブジェクトとなるユーザーに表示される値を返します。

したがって、例、SSN フィールドの場合、redaction ファンクションは、021-23-9567 の入力 SSN に対して xxx-xx-9567 を結果ます。

給与フィールドの場合、編集ファンクションは、入力給与値に関係なく常に \$0.00 を結果ます。

これらの機能は、CREATE REDACTION POLICY コマンドを使用して編集ポリシーに組み込まれます。このコマンドは、ポリシーが適用されるテーブル、指定されたリダクション関数の影響を受けるテーブル列、影響を受けるセッションユーザーを決定する式、およびその他のオプションを指定します。

postgresql.conf ファイルの edb_data_redaction パラメータは、データ修正を適用するかどうかを決定します。

デフォルトでは、パラメータが有効になっているため、墨消しポリシーが有効になり、次のことが行われます。

- スーパーユーザーとテーブル所有者はデータ改訂をバイパスし、オリジナルデータを見ます。
- 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。

セッション中にパラメータを FALSE に設定して無効にすると、次のことが発生します。

- スーパーユーザーとテーブル所有者はデータ改訂をバイパスし、オリジナルデータを見ます。
- 他のすべてのユーザーが取得すると、エラーが発生します。

リダクションポリシーは、ALTER REDACTION POLICY コマンドを使用して変更するか、DROP REDACTION POLICY コマンドを使用して削除できます。

リダクションポリシーコマンドについては、以降のセクションで詳しく説明します。

4.1 修正ポリシーの作成

CREATE REDACTION POLICY は、テーブルの新しいデータ改訂ポリシーを定義します。

概要

```
CREATE REDACTION POLICY <name> ON <table_name>
* [ FOR ( <expression> ) ]
* [ ADD [ COLUMN ] <column_name> USING <funcname_clause>
* [ WITH OPTIONS ( [ <redaction_option> ]
* [ , <redaction_option> ] )
* ]
* ] [ , ... ]
```

redaction_option は次のとおりです。

```
{ SCOPE <scope_value> |
* * EXCEPTION <exception_value> }
```

説明

CREATE REDACTION POLICY コマンドは、redaction ファンクションを使用して列データを編集することにより、テーブルの新しい列レベルのセキュリティポリシーを定義します。新しく作成されたデータ改訂ポリシーは、デフォルトで有効になります。ポリシーは、ALTER REDACTION POLICY ... DISABLE を使用して無効にできます。

FOR (expression)

このフォームは、墨消しポリシー式を追加します。

ADD [COLUMN]

このオプションのフォームは、テーブルの列をデータ改訂ポリシーに追加します。USING は、墨消しファンクション式を指定します。作成中のデータ改訂ポリシーにテーブルの複数の列を追加する場合は、複数の ADD [COLUMN] フォームを使用できます。オプションの WITH OPTIONS (...) 句は、適用するデータ改訂ポリシーのスコープまたは例外を指定します。スコープや例外が指定されていない場合、スコープと例外のデフォルト値はそれぞれ query と none になります。

パラメータ

name

作成するデータ改訂ポリシーの名前。これは、テーブルの他の既存のデータ改訂ポリシーの名前とは異なる必要があります。

table_name

データ修正ポリシーが適用されるテーブルの名前（オプションでスキーマ修飾）。

expression

データ改訂ポリシー式。この式が false と評価された場合、編集は適用されません。

column_name

データ修正ポリシーが作成されるテーブルの既存の列の名前。

funcname_clause

リダクションされた列の値を計算する方法を決定するデータリダクションファンクション。リダクションファンクションの戻り値の型は、データリダクションポリシーが追加される列の型と同じである必要があります。

scope_value

スコープは、列に編集が適用されるクエリーパートを識別しました。有効範囲の値は **query**, **top_tlist** または **top_tlist_or_error** です。スコープが **query** の場合、クエリーのどこに表示されているかに関係なく、列に編集が適用されます。スコープが **top_tlist** の場合、列の編集はクエリのトップターゲットリストに表示されたときにのみ適用されます。スコープが **top_tlist_or_error** の場合、動作は **top_tlist** と同じになりますが、クエリーのどこかに列が表示されるとエラーがスローされます。

exception_value

例外は、リダクションが免除されるクエリーパートを識別しました。例外値は **none**, **equal** または **leakproof** です。例外が **none** の場合、免除はありません。例外が **equal** の場合、列は等価性テストで使用されるときに編集されません。例外が **leakproof** の場合、リークプルーフファンクションが適用されても列は編集されません。

注：

データリダクションポリシーを作成または変更するには、テーブルの所有者である必要があります。

スーパーユーザとテーブルの所有者は、データ改訂ポリシーの対象外です。

例

以下は、この機能を稼動環境で使用方法の例です。employees テーブルにデータ編集ポリシーのコンポーネントを作成します。

```
CREATE TABLE employees (
  id          integer GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  name        varchar(40) NOT NULL,
  ssn         varchar(11) NOT NULL,
  phone       varchar(10),
  birthday    date,
  salary      money,
  email       varchar(100)
);

-- Insert some data
INSERT INTO employees (name, ssn, phone, birthday, salary, email)
VALUES
( 'Sally Sample', '020-78-9345', '5081234567', '1961-02-02', 51234.34,
  'sally.sample@enterprisedb.com'),
( 'Jane Doe', '123-33-9345', '6171234567', '1963-02-14', 62500.00,
  'jane.doe@gmail.com'),
( 'Bill Foo', '123-89-9345', '9781234567', '1963-02-14', 45350,
```

(次のページに続く)

(前のページからの続き)

```
'william.foe@hotmail.com');

-- Create a user hr who can see all the data in employees
CREATE USER hr;
-- Create a normal user
CREATE USER alice;
GRANT ALL ON employees TO hr, alice;

-- Create redaction function in which actual redaction logic resides
CREATE OR REPLACE FUNCTION redact_ssn (ssn varchar(11)) RETURN varchar(11) IS
BEGIN
    * /* replaces 020-12-9876 with xxx-xx-9876 */
    * return overlay (ssn placing 'xxx-xx' from 1) ;
END;

CREATE OR REPLACE FUNCTION redact_salary () RETURN money IS BEGIN return
0::money;
END;
```

ここで、employees にデータ編集ポリシーを作成し、デフォルトの範囲と例外を使用して、等しい条件と salary でアクセスできる列 ssn を編集します。hr ユーザの墨消しポリシーは免除されます。

```
CREATE REDACTION POLICY redact_policy_personal_info ON employees FOR (session_user !=
→ 'hr')
ADD COLUMN ssn USING redact_ssn(ssn) WITH OPTIONS (SCOPE query, EXCEPTION equal),
ADD COLUMN salary USING redact_salary();
```

hr ユーザの可視データは次のとおりです。

```
-- hr can view all columns data
edb=# \c edb hr
edb=> SELECT * FROM employees;
 id | name          | ssn          | phone          | birthday          |
 salary      | email
-----+-----+-----+-----+-----+-----+-----
 * 1 | Sally Sample | 020-78-9345 | 5081234567 | 02-FEB-61 00:00:00 |
 * $51,234.34 | sally.sample@enterprisedb.com
 * 2 | Jane Doe     | 123-33-9345 | 6171234567 | 14-FEB-63 00:00:00 |
 * $62,500.00 | jane.doe@gmail.com
 * 3 | Bill Foo     | 123-89-9345 | 9781234567 | 14-FEB-63 00:00:00 |
 * $45,350.00 | william.foe@hotmail.com
(3 rows)
```

通常のユーザ alice の可視データは次のとおりです。

```
-- Normal user cannot see salary and ssn number.
edb=> \c edb alice
edb=> SELECT * FROM employees;
id | name          | ssn          | phone          | birthday          | salary |
email
-----+-----+-----+-----+-----+-----+-----
* 1 | Sally Sample | xxx-xx-9345 | 5081234567 | 02-FEB-61 00:00:00 | $0.00 |
* sally.sample@enterprisedb.com
* 2 | Jane Doe     | xxx-xx-9345 | 6171234567 | 14-FEB-63 00:00:00 | $0.00 |
* jane.doe@gmail.com
* 3 | Bill Foo     | xxx-xx-9345 | 9781234567 | 14-FEB-63 00:00:00 | $0.00 |
* william.foe@hotmail.com
(3 rows)
```

ただし、`exception_value` の設定が原因で等値チェックに使用される場合、`ssn` データにアクセスできません。

```
-- Get ssn number starting from 123
edb=> SELECT * FROM employees WHERE substring(ssn from 0 for 4) = '123';
id | name          | ssn          | phone          | birthday          | salary |
email
-----+-----+-----+-----+-----+-----+-----
* 2 | Jane Doe     | xxx-xx-9345 | 6171234567 | 14-FEB-63 00:00:00 | $0.00 |
* jane.doe@gmail.com
* 3 | Bill Foo     | xxx-xx-9345 | 9781234567 | 14-FEB-63 00:00:00 | $0.00 |
* william.foe@hotmail.com
(2 rows)
```

警告

1. 継承階層で作成されたデータ改訂ポリシーはカスケードされません。例、データ編集ポリシーが親に対して作成された場合、それは子テーブルに適用されません。子テーブルはそれを継承し、その逆も同様です。これらの子テーブルにアクセスできる人は、編集されていないデータを見ることができます。継承階層については、次の URL で入手可能な PostgreSQL Core Documentation の* Inheritance *を参照してください。

- <<https://www.postgresql.org/docs/current/ddl-inherit.html>> 静的/ddl-inherit.html >

2. スーパーユーザまたはテーブル所有者がテーブルにマテリアライズドビューを作成し、テーブルのアクセス権 `GRANT SELECT` およびマテリアライズドビューを非スーパーユーザーに提供した場合、非スーパーユーザーは非マテリアライズドビューにアクセスできます。マテリアライズドビューを介してデータを編集しました。

3. 墨消しファンクション本体でアクセスされるオブジェクトは、スキーマ修飾されている必要があります。そうでないと、`pg_dump` が失敗する可能性があります。

互換性

CREATE REDACTION POLICY は EDB 拡張機能です。

参照

ALTER REDACTION POLICY, DROP REDACTION POLICY

4.2 修正方針の変更

ALTER REDACTION POLICY は、テーブルのデータ改訂ポリシーの定義を変更します。

概要

```
ALTER REDACTION POLICY <name> ON <table_name> RENAME TO <new_name>

ALTER REDACTION POLICY <name> ON <table_name> FOR ( <expression> )

ALTER REDACTION POLICY <name> ON <table_name> { ENABLE | DISABLE}

ALTER REDACTION POLICY <name> ON <table_name>
  * ADD [ COLUMN ] <column_name> USING <funcname_clause>
  *   [ WITH OPTIONS ( [ <redaction_option> ]
  *     [, <redaction_option> ] )
  *   ]

ALTER REDACTION POLICY <name> ON <table_name>
  * MODIFY [ COLUMN ] <column_name>
  * {
  *   [ USING <funcname_clause> ]
  * |
  *   [ WITH OPTIONS ( [ <redaction_option> ]
  *     [, <redaction_option> ] )
  *   ]
  * }

ALTER REDACTION POLICY <name> ON <table_name>
  * DROP [ COLUMN ] <column_name>
```

redaction_option は次のとおりです。

```
{ SCOPE <scope_value> |
  * * EXCEPTION <exception_value> }
```

説明

ALTER REDACTION POLICY は、既存のデータ改訂ポリシーの定義を変更します。

ALTER REDACTION POLICY を使用するには、データ改訂ポリシーが適用されるテーブルを所有する必要があります。

FOR (expression)

このフォームは、データ改訂ポリシー式を追加または置換します。

ENABLE

以前に無効化されたテーブルのデータ修正ポリシーを有効にします。

DISABLE

テーブルのデータ改訂ポリシーを無効にします。

ADD [COLUMN]

このフォームは、既存の墨消しポリシーにテーブルの列を追加します。詳細については、**CREATE REDACTION POLICY** を参照してください。

MODIFY [COLUMN]

このフォームは、テーブルの列のデータ編集ポリシーを変更します。列の墨消しファンクション句や墨消しオプションを更新できます。**USING** 句は更新するリダクションファンクション式を指定し、**WITH OPTIONS (...)** 句はスコープまたは例外、あるいはその両方を指定します。墨消しファンクション句、墨消しスコープ、墨消し例外の詳細については、**CREATE REDACTION POLICY** を参照してください。

DROP [COLUMN]

このフォームは、テーブルの列をデータ改訂ポリシーから削除します。

パラメータ**name**

変更する既存のデータ改訂ポリシーの名前。

table_name

データ修正ポリシーが適用されているテーブルの名前（オプションでスキーマ修飾）。

new_name

データ編集ポリシーの新しい名前。これは、テーブルの他の既存のデータ改訂ポリシーの名前とは異なる必要があります。

expression

データ改訂ポリシー式。

column_name

データ修正ポリシーが変更またはドロップされるテーブルの既存の列の名前。

funcname_clause

列のデータ修正ファンクション式。詳細については、**CREATE REDACTION POLICY** を参照してください。

scope_value

スコープは、列に編集が適用されるクエリーパートを識別しました。詳細については、**CREATE REDACTION POLICY** を参照してください。

exception_value

例外は、リダクションが免除されるクエリーパートを識別しました。詳細については、CREATE REDACTION POLICY を参照してください。

例

employees 名前付けのテーブルの redact_policy_personal_info というデータリダクションポリシーを更新します。

```
ALTER REDACTION POLICY redact_policy_personal_info ON employees
FOR (session_user != 'hr' AND session_user != 'manager');
```

また、同じポリシー内の列 ssn のデータ編集ファンクションを更新するには：

```
ALTER REDACTION POLICY redact_policy_personal_info ON employees
MODIFY COLUMN ssn USING redact_ssn_new(ssn);
```

互換性

ALTER REDACTION POLICY は EDB 拡張機能です。

参照

CREATE REDACTION POLICY, DROP REDACTION POLICY

4.3 ドロップ修正ポリシー

DROP REDACTION POLICY は、テーブルからデータ改訂ポリシーを削除します。

概要

```
DROP REDACTION POLICY [ IF EXISTS ] <name> ON <table_name>
* [ CASCADE | RESTRICT ]
```

説明

DROP REDACTION POLICY は、指定されたデータ改訂ポリシーをテーブルから削除します。

DROP REDACTION POLICY を使用するには、墨消しポリシーが適用されるテーブルを所有する必要があります。

パラメータ

IF EXISTS

データ編集ポリシー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。

name

削除するデータ改訂ポリシーの名前。

`table_name`

データ修正ポリシーが適用されているテーブルの名前（オプションでスキーマ修飾）。

CASCADE

RESTRICT

これらのキーワードは、データ修正ポリシーに依存しないため、効果がありません。

例

`employees` 名前付けのテーブルに `redact_policy_personal_info` というデータ編集ポリシーを削除するには：

```
DROP REDACTION POLICY redact_policy_personal_info ON employees;
```

互換性

DROP REDACTION POLICY は EDB 拡張機能です。

参照

CREATE REDACTION POLICY, ALTER REDACTION POLICY

4.4 システムカタログ

このセクションでは、墨消しポリシー情報を保存するシステムカタログについて説明します。

4.4.1 edb_redaction_column

`edb_redaction_column` システムカタログには、テーブルの列に添付されたデータ修正ポリシーに関する情報が格納されます。

Column	Type	References	Description
<i>oid</i>	<i>oid</i>		行識別子（非表示の属性、明示的に選択する必要があります）
<i>rdpolicyid</i>	<i>oid</i>	<i>edb_redaction_policy.oid</i>	データ編集ポリシーは、説明されている列に適用されます
<i>rdrelid</i>	<i>oid</i>	<i>pg_class.oid</i>	記述された列が属するテーブル
<i>rdattnum</i>	<i>int2</i>	<i>pg_attribute.attnum</i>	記載されている列の番号
<i>rdscope</i>	<i>int2</i>		墨消しスコープ：1 = クエリー、2 = top_tlist、4 = top_tlist_or_error
<i>rdexception</i>	<i>int2</i>		墨消しの例外：8 = なし、16 = 等しい、32 = 漏れ防止
<i>rdfuncexpr</i>	<i>pg_node_tree</i>		データ編集ファンクション式
			NA

!!! Note * 表のリダクションポリシー `edb_redaction_column.rdpolicyid` が有効で、リダクションポリシー式 `edb_redaction_policy.rdexpr` が `true` に評価される場合、説明されている列はリダクションされます。

4.4.2 edb_redaction_policy

カタログ `edb_redaction_policy` は、テーブルの墨消しポリシーに関する情報を格納します。

Column	Type	References	Description
<i>oid</i>	<i>oid</i>		行識別子（非表示の属性、明示的に選択する必要があります）
<i>rdname</i>	<i>name</i>		データ編集ポリシーの名前
<i>rdrelid</i>	<i>oid</i>	<i>pg_class.oid</i>	データ編集ポリシーが適用されるテーブル
<i>rdenable</i>	<i>boolean</i>		データ編集ポリシーは有効になっていますか？
<i>rdexpr</i>	<i>pg_node_tree</i>		データ編集ポリシー式
			NA

!!! Note * テーブルが有効で、式が `true` と評価された場合、データリダクションポリシーがテーブルに適用されます。